Self Adapting Plant Models

Thomas Ashby

August 18, 2019

1 Abstract

Presented is a method for dynamic interaction of large numbers of unique trees with triangulated meshes at near real-time speeds. The method uses ray collisions, branch transformations, culling, and mass redistribution to provide an approximation of a plants reaction to light, arguably the most important tropism affecting plant growth. This approach is much faster than existing approaches such as space colonisation methods, or biologically based simulation based techniques.

Contents

2 Introduction

Simulation of plants and their interaction with each other and their environment is a complex and open topic in simulation and computer graphics. In this paper I present a method for creating communities of plants which are able to interact with their environment in an efficient manner. Many existing applications extensively model light interaction, resource availability, root systems, and growth over time. I present a method which approximates much of this and uses ray intersections for object avoidance yielding adequate results at near real-time speeds with the potential for further performance increases with the use GPU computation. Ray intersection for plant growth could prove useful in the near future with the advent of real-time ray tracing capabilities on graphics cards. I also present an intuitive method of reducing asset size using the camera position to decide the level of detail when exporting to digital content creation tools.

The motivation behind this project would be to create an efficient means to allow large numbers of trees to interact with meshes created in digital content creation tools. This allows existing scenes to be populated with communities of plants bushes and trees quickly and easily, and also with the flexibility to define any number of type of plants, with the resulting vegetation having minimal storage requirements. The need for for this is ever increasing in the animation industry. With films such as The Lion King (2019) which features 92 distinct species populated around 921 unique assets, and many other recent feature films containing huge amounts of computer generated vegetation, the need for an efficient method to place these assets is ever more apparent and the ability to remove some of this burden from modellers and layout artists is a valuable asset. Simply placing vegetation is not sufficient. Models need to be adapted to suit their environment, the most important adaptation being light availability. This paper focuses on this adaptation as it is the most important in deciding plant structure, and ignore elements such as precipitation levels, ground nutrient contents, and biome simulation, as it can be assumed that the user has already selected suitable plants based on real world examples.

3 Previous work

L-Systems or Lindenmayer-systems are a formalism created by Astrid Lindenmayer, a Biologist, in 1968. They were created as a model of biological development based on axioms, an early attempt at representing plants algorithmically using a turtle graphics style system. They have found uses in many area of computer graphics, not just for plants, and are the basis of many systems which model nature and vegetation growth.

In The Algorithmic Beauty of plants [1] many variations to the classic rewriting are detailed. In order to represent branching axial trees one such modification is suggested, a special data structure using a stack to store the current position of the l-system at a branching point. axial trees can be represented using strings with brackets [Prusinkiewicz et al. 1996, p. 24]. These brackets perform operations to push the current state of a 'turtle' to a stack, or to pop the current state from the stack.

To add to the realism of created structures many implementations use non-deterministic approaches to create variations in the generated structures. Eliminating artificial looking regularity can be achieved by randomising the turtle interpretation, the l-system, or both[Prusinkiewicz et al. 1996, p. 28]. Randomisation of the turtle interpretation changes geometric aspects of the plant such as branch length and branch angles. The stochastic application of string replacements has the effect of changing both the geometry and topology of the plant.

This stochastic application of rules can be defined as the ordered quadruplet $G_\pi = (V, \omega, P, \pi)$. where V is the alphabet, ω is the axiom, P is a set of productions, and π represents the function $\pi : P \rightarrow [0, 1]$. This is the probability distribution which maps the set of productions to the probabilities. The sum of the probabilities for each letter should equal 1. this is based off of a definition by Peter Eichhorst, Walter J. Savitch [2].

The modeling of realistic trees can be achieved using extensions to stochastic l-systems. de Reffye[3] modeled the activity of buds over time using the following rules:

1. do nothing
2. die
3. become flower
4. become an internode (ending with an apex and containing 1+ lateral shoots ending in leaves)

These behaviours can be modeled with string replacements in stochastic l-systems. They are often combined with growth functions to simulate the change in a plant structure over time. Two main growth functions are suggested, Square-root growth and a sigmoidal growth function. It may be possible to estimate the distribution in plant sizes using this function for a specified time without the need for complex and expensive growth simulations.

Extensions were also proposed by Honda [4]. One such extension was a shortening of child segments by $r1$ $r2$ in relation to the parent branch. The origins of computer modeling of trees can be linked to Honda's paper for this recursive module size reduction ad the use of parameters for branching angles. Honda's model formed the basis of Aono and Kunii [5], which added bias in a particular direction to mimic effects of strong wind, or tropisms, without the need for expensive simulations.

Improvements to generated the geometry of these models were made by Bloomenthal [6] by using curves in lieu of straight line segments to represent branches of maple trees, which yields a more natural look.

To model the topology of particular trees we can look at the detailed plant models described in Halle, Oldeman and Tomlinson[7]. therein is a definition of 23 common tree architectures and descriptions of their characteristics. One of which will be referred to in this work, Attims' model, which incorporates plants of the genus *Alnus*.

3.1 Plant Interactions

Literature detailing the interplay of biological phenomena when simulating plants is widely available, and is an ongoing topic in computing. Most recently Makowski et al. [8] describe a multi-scale method for large scale ecosystems which captures tropisms, interactions with objects and other trees, and resource competition.

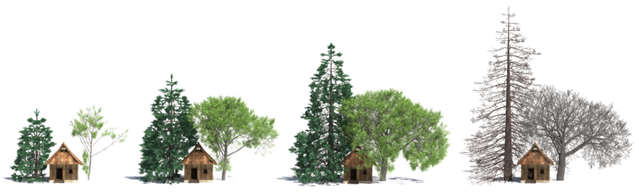

Figure 1: Example from Makowski et al. [8]

The paper mentions the importance of an adequate level of detail algorithm in simulating large amount numbers of plants, and how few approaches both simplify geometry whilst adhering to plant structure. One method it refers to to represent plants at a different level of abstraction is the use of points and lines from Gilet et al[9]. A main focus of the Makowski et al. paper is to use grouping of branch structures to enable instancing on the GPU for greater performance. The paper not only models tree-tree interactions, but also tree-object interactions. Their method for local shadow approximations is taken from the propagation method of Palubicki et al[12] described in section 3.2. Their work is very broad and covers multiple levels of plant simulation across different environments with many environmental factors. Their work is an amalgamation of works across various domains in plant simulation and provides a means of generating large forest systems without direct control over

Attempts at intuitive methods for plant modeling and placement were made in 2009 by Benes et al.[10] Their implementation features intuitive placement and plant generation techniques which don't rely on large numbers of input parameters.

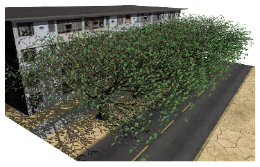

Figure 2: Example from Benes et al. [10]

Their method models interactions between plants and between plants and objects. The 3D meshes are calculated using spline interpolation of growing bud locations, and uses generalised cylinders to represent branches. Collision detection is computed from radii associated with branch apices which are checked against a voxel space. a box of $n * n$ voxels to be used as a new bud location is checked for potential collisions. This voxel space method has $O(n)$ complexity. Collision detection is recognised as the cost critical part of the simulation.

For the simulation of tree - object interactions Pirk et al.[11] offer an approach which builds on the work Benes et al. It uses interacting self adapting botanical tree models. This technique has complex tree models interacting with their environment in real time. It models light distribution, proximity to solid obstacles, and to other trees to approximate biologically motivated changes in the tree structure. This method ensures transformations are only performed when required, thus has a performance advantage over space colonisation based systems. It also has yields a performance increase over previous works and reduces computation time from minutes to near real time speeds. As trees react almost exclusively to changes in local lighting, they model light received by each of the leaf clusters in order to model both the effects of shadows cast by other leaf clusters, and from shadows cast by other objects such as walls. For walls this is achieved by creating a shadow volume which intersects the tree. In order to enable real time placement of trees, the adaptation and deformation is turned off whilst the tree is being placed, and the tree is adapted once placed. The pruning method in this adaptation is based on the amount of light the leaf clusters receive. Wood or plant organs are produced based on the amount of received light. If a bud is in shadow for an extended period of time then it will eventually die off.

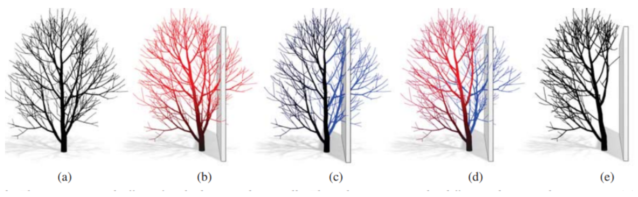

Figure 3: Example from Pirk et al. [11]

Their approach is to use the sum of node distances to all leaf nodes l_t and the amount of resources gathered by child leaf clusters ζ_t. A branch is pruned when the ratio ζ_t/l_t is smaller than some threshold value called pruning factor ψ. For a given branch segment s, the gathered amount of resources ζts is computed from the light that is received on the leaf clusters Cs that are located on the child nodes of a branch segment s[11]:

$$\zeta_{t_s} = \sum_{c \in C} 2\pi l r_c^2 i_c$$

Where r_c is the radius of a given leaf cluster and i_c is the normalized amount of light that the cluster receives.

At every stage of tree growth a local pruning factor is computed for each branch segment: $\psi_s(t)$. The individual pruning factor for each branch is:

$$\psi_{s_{min}} = c_\psi min(\psi_{ref}, \underset{\forall t}{min}(\psi_s(t)))$$

Once they have performed transformations, resource allocation is performed, for all branches of a set age. Finally, all branches and child branches with resources less than $\psi_{s_{min}}$ are deleted.

Input data for this system is taken from laser scans of trees which adds an additional step when compared to previous work. They attempt to reverse the environmental changes of the input tree in order to find out what it would have looked like in perfect conditions, this must be performed before any of the previously mentioned steps to prevent the scanned mesh from effecting results.

Pirk models leaf clusters for efficiency, these clusters are populated with leaves using the GPU once transformations are complete. They define the relationship between normalized cluster density ρ_l and incoming light i is:

$$\rho_l = \frac{\rho_{l_0}}{\rho(il_0)}\rho(i)$$

This is for a cluster l of density ρ_{l_0} for light value i_{l_0}

The translucency values for these clusters γ_c' is updated when changes to the structure are made with $\gamma_c^{\frac{n_c}{n_0}}$

Their method is not perfect as leaf clusters are only approximately filled by the GPU, so some branches may enter obstacles. They allow this as it's effect is not significant, but mention it as a potential future work to prune these branches.

Ray plane intersection for recognising collisions between plants and objects has so far seen use in generating climbing plants[14]. Knutzen uses ray plane intersections with bounding volume hierarchies and achieves similar speeds to the voxel based methods of Greene[15]

3.2 Shadow Approximation

many dynamic plant growth and placement algorithms, including those already covered, account for light levels. An efficient shadow propagation method is implemented in Self-organizing tree models for image synthesis [12]. A voxel space of shadow values is used to estimate the light received at each bud, a method suggested by Palubicki [13].

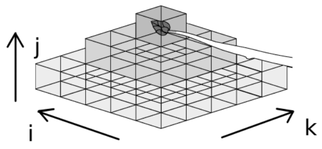

Figure 4: Palubicki et al. [12]

Each voxel (i, j, k) is assigned a shadow value with an initial value zero. a pyramidal penumbra is created which propagates downwards. Voxels in this penumbra have indices $(i, j, k) = (I \pm p, J - q, K \pm p)$ where $q = 0, 1, ..., q_{max}$ and $p = 0, 1, ..., q$. The shadow value in affected voxels in increased by $\Delta s = ab^{-q}$. $a > 0$ and $b > 1$ are defied by the user. Light exposure is calculated as $Q = max(C - s + a, 0)$ where C is complete exposure.

4 Implementation

The implementation is a program written in c++ and openGL. GLFW is used as the OpenGL container and to handle keyboard inputs. The testing framework chosen to ensure the accuracy and robustness of generated trees and meshes is Google test. Tree description files are loaded according to a central JSON scene description file and expanded to form full skeletal trees. These are then placed and reacted with the loaded geometry.

4.1 Tree generation

For the generation of trees the implementation uses a text based description of the skeleton to be generated. The specification for the files are as follows:

1. Number of generations
2. Angle specification (Radians)
3. Thickness
4. Axiom
5. Rule = Replacement : Probability

A full set of rules for describing stochastic l-systems is implemented. The set of available symbols is based loosely on those used in SideFX Houdini[16] and are as follows:

RULE	REPLACEMENT
F	Create a node / move the turtle forward.
[	Create a new branch.
]	End the current branch.
V	Create a leaf.
$+$	Rotate the turtle clockwise.
$-$	Rotate the turtle Anticlockwise.
&	Pitch the turtle up.
ˆ	Pitch the turtle down.
Y	Roll the turtle left.
U	Roll the turtle right.

For each plant type the description file is loaded into memory and rule objects are created and populated. The axiom is then expanded based on these rules. For stochastic rules a replacement is returned based on the specified probability. Probabilities are represented by floating point numbers between 0 and 1.0 and it can be assumed that the sum of probabilities for all replacements of each rule equals 1.0. once the full string is generated we parse this to build our recursive skeleton structure. Each node stores a list of its children and a pointer to its parent. Vectors representing the start and end positions, as well as branch radius are stored. Segment type is decided to be either trunk or lateral segment by checking if we are on a branch.

For our multi scale modeling as in Makowski et al[8] we specify the plants to be loaded and their quantities in a JSON format scene description file. This file enables a user to load any number of plants into the simulation, generating woods and forests. It is in this file where we also specify the .obj collision meshes for the simulation of tree object interaction.

Data in the program is stored in such a manner that is is flexible enough for data to be passed between branches easily and efficiently, and also so that the computers central processing unit and memory can handle the data as quickly as possible. The tree's structure is stored as branch object in a recursive structure. All lists are stored as std::vector to ensure the elements are stored in contiguous memory to help make the code more cache friendly.

4.1.1 Exemplar

In order to more faithfully model real plants an exemplar is taken as a reference. I have chosen a plant which is described by Attim's model in Halle et al [7]. *Alnus glutinosa* or common alder is in the alnus family which has characteristics most similar to Attim's model. The tree architecture is characterised by axes with continuous growth of trunk and branches. Axes are differentiated into a monopoidal trunk and equivalent branches. Branching takes place continuously or diffusely. Flowering is always lateral and does not affect shoot construction [7]. Examples of this tree can be seen to exhibit photo-tropism across the main structure, it will tend to move towards areas with a higher average light intensity and thus avoid obstacles.

The common alder in figure 5 exhibits a 17 degree lean from the vertical axis. It also shows a return to the vertical axis once a height is reached where more light is available.

Another common tree which exhibits leaning characteristics is the London Planetree (*Platanus acerifolia*) [Figure 6]. This archaeophyte is generally thought to be a hybrid of *Platanus orientalis* (oriental plane) and *Platanus occidentalis* (American sycamore), its family *Platanaceae* is not listed in halle

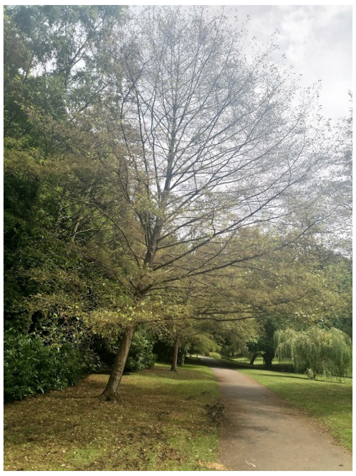

Figure 5: Common Alder. August 3 2019. f/2, 1/850, 3.58mm

et al but seems to exhibit architectural characteristics of either Ruah or attim's model. It again shows a monopoidal trunk and develops tiers of branches which are morphogenetically similar to the trunk. It also has lateral flowers which do not effect the growth of shoots. Studied examples show non-rhythmic growth of meristems which would likely also place this in Attim's model.

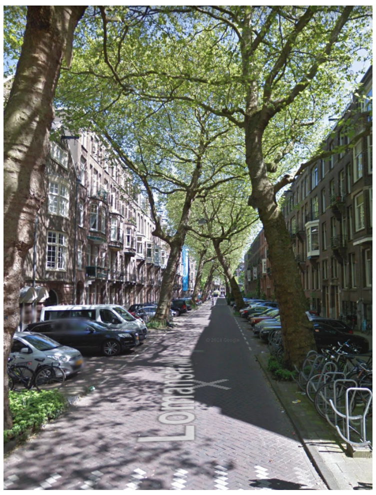

Figure 6: London Planetrees, Lomanstraat, Amsterdam. May 2014. Google.

4.2 Tree Placement

4.2.1 Growth

My implementation is for producing trees or groups of trees as a climax community of mostly fully grown trees. For this reason it is not necessary to simulate any type of tree growth, somewhat simplifying generation. In order to simulate variation in the size of mature plants scale is varied using a random floating point number giving around a 5 percent variation in plant size. A sigmoid-like function is also used to vary the scale and number of generations in a small number of plants. The function used is $2 * (x/(1 + x))$. Where x is a random floating point number between 0 and 1, and the result is clamped to 1 The function represents a fast approximation of a sigmoid curve for positive numbers, as we do not have the need for negative tree sizes. The result of this function is a population of trees which are mostly fully grown with a few which are mid growth, and even fewer adolescent trees. With large tree populations this presents a feature seen in Makowski et al[8], where trees dying and leaving space for new growth are simulated. By creating a very small number of small trees in a large population, this gives an effect similar to Makowski et al at any fixed point in their simulation, a population of mature trees with a few saplings taking advantage of renewed space. It is important to note that simulation is not involved in this process and thus it is only an approximation.

4.2.2 Placement Algorithm

the algorithm used for tree placement is random scattering approach where a tree is moved to another position if it gets too close to another object. This is more efficient for populations of different sized trees than other methods such as mass-spring or particle systems. Since for most of the placement operation there are very few collisions the complexity of the operation is close to O(n2).

The domain for tree placement is decided in the JSON scene description file. Once the placement domain is decided a random position is decided on the plane. This position is checked against all existing plants to make sure it is not within the bounding cylinder of any plant. If a collision is found then it will try again. This will repeat up to 50 times. If no collision is found then the tree is generated in that position. This still allows for some overlapping of the trees as a free point may be found on the edge of a bounding cylinder. This just ensures a denser forest can be generated can easily be altered to yield thinning forests. It if following these steps when a random rotation is added to the plant. The placement for that plant is now complete and the steps are repeated for the remaining plants.

4.2.3 User Placement

After the initial plant placement is complete the results are displayed to the user. For small numbers of plants, plants can be selected with the square bracket keys and the currently selected plant is highlighted in orange, the plant can then be moved with the arrow keys and with update according to surrounding objects once the arrow keys are released. For larger numbers of plants the meshes can be exported and altered in a digital content creation tool.

4.3 Dynamic Interaction

4.3.1 Collision detection

For collisions with objects the branches are treated as rays and are checked for collisions against a triangulated collision mesh. The MollerTrumbore intersection algorithm is used for its efficiency. This method can be easily parralelised and runs on multiple threads using OpenMp. It also has potential to be calculated on the GPU for even greater speeds. The branches are checked for ray triangle collisions against the meshes and hits are noted. The sum of the direction of these collisions is then computed, and the direction reversed in order to find the bending angle.

4.3.2 bending

We bend the skeletal mesh by recursively transforming the branches by the defined maximum bending angle and multipliers.

The first step in the process is to straighten the tree by removing any previous transformations. This allows for repeated bending of the tree without losing the original shape.

After this is completed a collision manager class will check collisions of each branch and store pointers to the colliding branches.

If any collisions have occurred then the average direction of those collisions from the root of the plant is calculated. This is then used to calculate an approximation of the phototropism direction and to bend the plant. The tree is then rotated branch by branch away from the collided object. The rotation is started at the root of the plant with all children inheriting the transformation of the parent. The rotation amount can be modified branch by branch using the bend multiplier to gradually reduce the amount of bending through the plant. With more bending seen in the lower branches and the trunk similar results are produced to the photographed examples in figures 5 and 6. This product also aligns with biological rules where new shoots are less likely to bend towards light as they have had less time to adjust to light availability.

4.3.3 pruning

Once the transformations have been performed the initial colliding branches once again checked for collisions. Branches which still collide are rendered invisible along with their children. The use of a recursive structure simplifies this to a single function call. It is possible that with some convex meshes, branches which did not initially collide with any mesh will be colliding after the plant has been transformed. In most cases this is not noticeable and only checking colliding branches has been chosen for performance reasons.

4.3.4 energy redistribution

using Honda's model we can redistribute energy elsewhere in the simulation when branches are pruned. We add up the total lengths of the pruned branches and divide by the number of remaining branches. We can then distribute this saved energy by 'growing' the remaining branches towards the sunlight by transforming their lengths. This represents an approximation of the model which is believable enough for our purposes. The process is as follows:

```
grow():
for all visible branches:
    growthLength = energySaved / numberOfVisibleBranches;
    branchVector = endOfBranch - m_startOfBranch;
    normalize(branchVector);
    branchVector *= growthLength;
    branch4x4Matrix[3] = branchVector;
    for all children:
        grow();
```

This method ensures that all energy saved from pruning a plant is redistributed elsewhere in the structure, conserving the amount of plant mass.

The results of the collision detection, bending, pruning and energy redistribution steps are shown in figures 7 and 8. This angle of deviation shown is the result of a max bed of 0.1 and a bend multiplier of 0.9 as specified in the scene description file. The trunk angle in figure 8 is comparable to the trunk of the exemplar in figure 5.

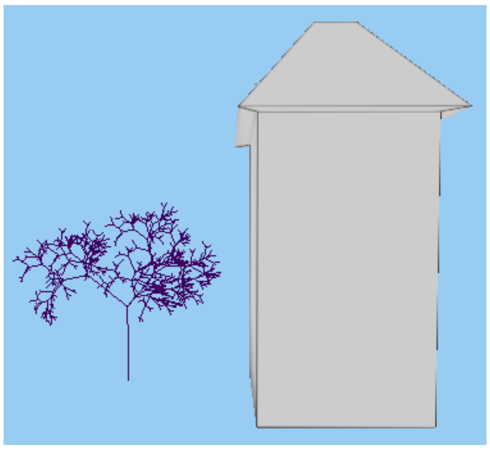

Figure 7: Before

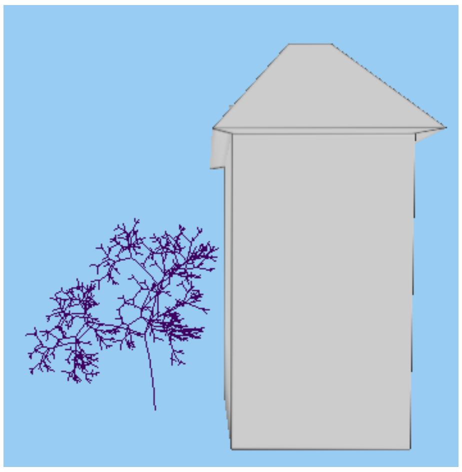

Figure 8: After

4.4 mesh generation

Meshes are exported in Wavefront Obj format for use in digital content creation tools. One .obj file is exported containing multiple objects. Branches are turned into cylinders with the user defined thickness adapted according to the recursion depth. Thickness t is defined as $t = i * m^b$. Where i is the initial thickness, m is the thickness multiplier, and b is the number of branching points before the current branch.

To construct cylinders of branch length a vector perpendicular to the branch direction is found and set to 1/2 thickness. A quaternion is then created with the branch direction as the axis and is repeatedly created according to $360.0f/a$ where a is the axial resolution.

4.4.1 Exporting

In order to reduce the size of the exported meshes the program has an automatic level of detail algorithm implemented. Using the OpenGL camera as the view direction, level of detail is calculated based on the distance of each tree to the camera. Closer meshes have an axial resolution of 8, meaning there are 8 faces to each cylinder, further meshes have an axial resolution of 3. This greatly reduces storage costs for exported meshes and is ideal for use in scenes with a fixed camera position. To reduce mesh size we reduce the axial resolution(the number of faces on each cylinder) which in turn reduces the number of vertices and indices which need to be exported.

5 Evaluation

Seen in figures 9 and 10 the results are similar to those of Pirk et al. The trees in Pirk et al were obtained through laser scans of real trees. Comparing their results with my own prove challenging due to the difficulties of creating their scanned trees with my l-system. Figure 10 shows a roughly similar tree being deformed in similar environments.

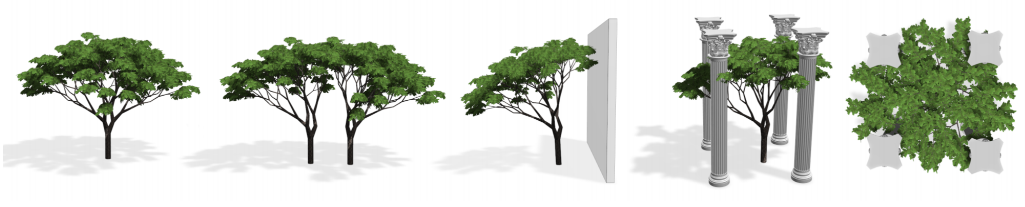

Figure 9: Results from Plastic trees[11]

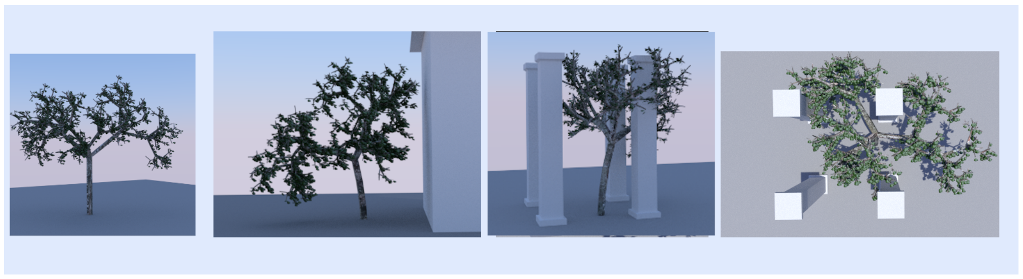

Figure 10: My results

6 Results

The system was implemented in C++ using OpenGL. It was developed and run on a system with an intel i3-5010u CPU @ 2.10GHz an intel HD 5500 integrated GPU. The speed of the algorithm could be improved with geometry instancing but yields quick results due to the simple nature of the displayed geometry. A population of 1000 trees takes 4 minutes and 16 seconds to generate on the specified hardware.

Once exported the trees were rendered in cycles using a particle system to instance leaves. What has been created is the beginnings of a pipeline to generate large amounts of trees interacting with any number of specified objects with believable results, with the option to edit the plants at real time speeds. Figure 10 shows the type of results which can be expected from the pipeline which has been set up. Various phenomena are captured by the program such as phototropism, the competition for space, the direction of energy through the plant structure towards the plant tropisms, and to an extent the senescence of plant branching structures do to the lack of light through the pruning algorithm. Through careful use of the program, human effects on trees can also be simulated. Tree surgery operations such as side pruning, slope pruning, v-pruning, and crown reduction can be performed to direct the shape of the tree. Collision meshes can be imported to selectively remove branches from target plants. This is useful in situations where branch growth is not desirable but cannot be prevented by other means. Examples of this is around power lines or into roads, where humans would typically prune trees. These effects can be achieved by importing a collision mesh representing an area where growth in not desirable, then removing it later in the pipeline so it is not visible in the final render.

The speed of the simulation is currently limited by the complexity of the colliding meshes. Current tests have typically used fewer than 100 triangles. The program can scale to larger meshes but is currently prohibitively expensive, and would benefit from optimisations such as octrees or axis aligned bounding boxes for initial collision checking before comprehensively checking all triangle collisions.

Currently the algorithm does not take into account the height of the collisions between branches and objects. For some cases this can yield inaccurate results. When simulating a tree whose trunk typically returns to the vertical axis once above the obstruction. This model will continue to grow away from the obstruction. This is not unheard of but does not fit all cases. It is however adequate in this case and the speed savings from the removal of light simulation gives its own benefits.

7 Conclusion and Future Work

7.1 Conclusion

I have presented a multi scale tree and forest creation method which integrates plant movement, pruning, and vigor through energy distribution. Through my methods, meshes are exported which react to unknown obstacles efficiently and produce unique meshes with minimal storage costs. With modifications this project could be ideal for use in a film pipeline where the only alternative would be to manually sculpt and deform vegetation. My method is highly scalable and has potential to be used to perform real time dynamic plant positioning.

7.2 Future Work

There are some noteworthy gaps and flaws in my work. One area which is lacking is the user interface. With the addition of an intuitive user interface this program could form an easy to use tool for creating plastic forests with few parameters, and would align the implementation closer to that of Pirk et al[11]. For potential further speed improvements the population of trees could be reduced to a smaller number and these could be repeated with varying scales and rotations which would minimise the computation time required to generate unique trees, something seen in Makowski et al. The application of this method would also reduce storage costs if the unique trees were stored individually and loaded into digital content creation tools using a script to only instance the trees when required. The program currently uses OpenMP to great benefit allowing the collision detection to run on all CPU cores. This has potential to be further accelerated using the GPU to calculate ray triangle intersections for collisions, and would be an interesting application for real-time ray-tracing technologies. Elements have been borrowed from recent literature on the topic of plant creation to create a program which could fit within a vegetation creation pipeline, but the application is certainly not ready for use in a production environment.

8 Bibliography

1. Prusinkiewicz, P., Lindenmayer, A. 1996, The algorithmic beauty of plants. , Springer .

2. Eichhorst, P., Savitch, W., 1980. Growth functions of stochastic Lindenmayer systems, Information and Control, Volume 45, Issue 3, Pages 217-228.

3. de Reffye P., Barthlmy D., Blaise F., Fourcaud T., Houllier F. 1997. A functional model of tree growth and tree architecture. Silva Fennica vol. 31 no. 3

4. Honda, H. 1971. Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body. Journal of Theoretical Biology 31, 331-338.

5. Aono m., Kunii T., Botanical tree image generation. IEEE Computer Graphics and Applications, 4(5):10-34, 1984.

6. Bloomenthal J., Modeling the mighty maple. Proceedings of SIGGRAPH 85 (San Francisco, California, July 22-26, 1985) in Computer Graphics, 19, 3 (July 1985), pages 305-311, ACM SIGGRAPH, New York, 1985.

7. F. Halle, R. A. A. Oldeman, and P. B. Tomlinson. Tropical trees and forests: An architectural analysis. Springer-Verlag, Berlin, 1978.

8. Makowski M., Hadrich T., Scheffczyk J., Michels D. L., Pirk S., Palubicki W. 2019. Synthetic silviculture: multi-scale modeling of plant ecosystems. ACM Trans. Graph. 38, 4, Article 131 ,July 2019.

9. Gilet G., Meyer A., and Neyret F., 2005. Point-based Rendering of Trees (NPH05). 67-73

10. Benes, B., Andrysco, N.,Andstava, O., 2009. Interactive modeling of virtual ecosystems. Eurographics Workshop on Natural Phenomena, Eurographics Association, 9-16.

11. S. Pirk, O. Stava, J. Kratt, M. A. M. Said, B. Neubert, R. Mch, B. Benes, and O. Deussen. 2012. Plastic trees: interactive self-adapting botanical tree models. ACM Trans. Graph. 31, 4, Article 50 (2012), 10 pages.

12. Palubicki, W., Horel, K., Longay, S., Runions, A., Lane, B., Mech, R., Prusinkiewicz, P. 2009. Self-organizing tree models for image synthesis. In Proceedings of SIGGRAPH 09, 1-10.

13. Palubicki, W. 2007. Fuzzy plant modeling with OpenGL. VDM Verlag, Saarbrucken.

14. Knutzen J. 2009. Generating Climbing Plants Using L-Systems.

15. Greene, N. 1989. Voxel space automata: modeling with stochastic growth processes in voxel space. In SIGGRAPH 89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques, pages 175-184.

16. Side Effects Software Inc., Houdini, Version 17.5, March 2019, www.sidefx.com

9 Appendix

9.1 User Manual

The program is built with qMake in qt creator and requires the following dependencies to be installed to run on linux:

-lassimp -fopenmp -lGLEW -lglfw -lGL -lX11 -lXi -lXrandr -lXxf86vm -lXinerama -lXcursor -lrt -lm -pthread -ldl

When starting the program it will hang for a few seconds whilst building the tree skeletons before displaying anything.

The program is controlled using a keyboard and mouse. The camera can be panned by clicking and dragging the left mouse button, and moved using the W,A,S,D keys.

The current selected tree is shown in orange and can be changed using the square bracket keys. The selected tree can be repositioned with the arrow keys and will be updated when the keys are released.

The T key will export tree trees from the current camera view to output.obj in the output folder.

9.2 Glossary

1. Apex - pl. apices: The tip; point furthest from the point of attachment.
2. Archaeophyte: A non-native plant which has been present in the geographic location for an extended period of time.
3. Crown: The upper layer of a plant.
4. Senescence: The biological ageing and deterioration of an organism.
5. Shoot: A young branch growing from the main plant structure.
6. Tropism: The orientation or growth of an organism in response to a stimulus (phototropism: response to light. Gravitropism: response to gravity).
7. Vigor: The vitality and performance of a plant.