Simulation Levels of Detail for Plant Motion

J. Beaudoin and J. Keyser

Texas A&M University

Abstract

In this paper we describe a method for simulating motion of realistically complex plants interactively. We use a precomputation stage to generate the plant structure, along with a set of simulation levels of detail. The levels of detail are made by continuously grouping branches starting from the tips of the branches and working toward the trunk. Grouped branches are simulated as single branches that have similar simulation characteristics to the original branches. During run-time, we traverse the plant and determine the allowable error in the simulation of branch motion. This allows us to choose the appropriate simulation level of detail and we provide smooth transitions from level to level. Our level of detail approach affects only the simulation parameters, allowing geometry to be handled independently. Using this method we can significantly improve simulation times for complex trees.

Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism -- Animation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling – Physically based modeling

1 Introduction

Due to their ubiquity in natural environments, plants and trees are important features in almost all animations of outdoor scenes. Furthermore, because they are flexible objects, the motion of these plants in response to wind or other external forces provides an important, if sometimes subtle, visual cue for establishing the realism of the scene. Unfortunately, plants and trees can be quite complex, and as a result any simulation of the motion is likely to take a great deal of time. For environments with several trees, this is particularly the case. If we are to have any hope of simulating groups of trees for interactive applications, a level of detail approach will be needed.

There has been a great deal of previous work on the modeling of plant growth. Less work has focused on the simulation of plant motion, and less still on methods for increasing the performance of such simulations. Our approach is aimed at this final area: increasing the efficiency of plant motion simulation through the use of simulation levels of detail. A goal is to be able to apply our method to an environment composed of many trees, each defined with realistic complexity (in terms of number of leaves and branches).

Simulation levels of detail (SLODs) are the animation analog to geometric levels of detail. With geometric LODs, simplified geometric representations requiring less rendering time (but providing less detail) replace the highly detailed original model, when this replacement will result in no or minimal visual error**.** Sim[1]ilarly, with simulation LODs, a less complex simulation is used to replace the

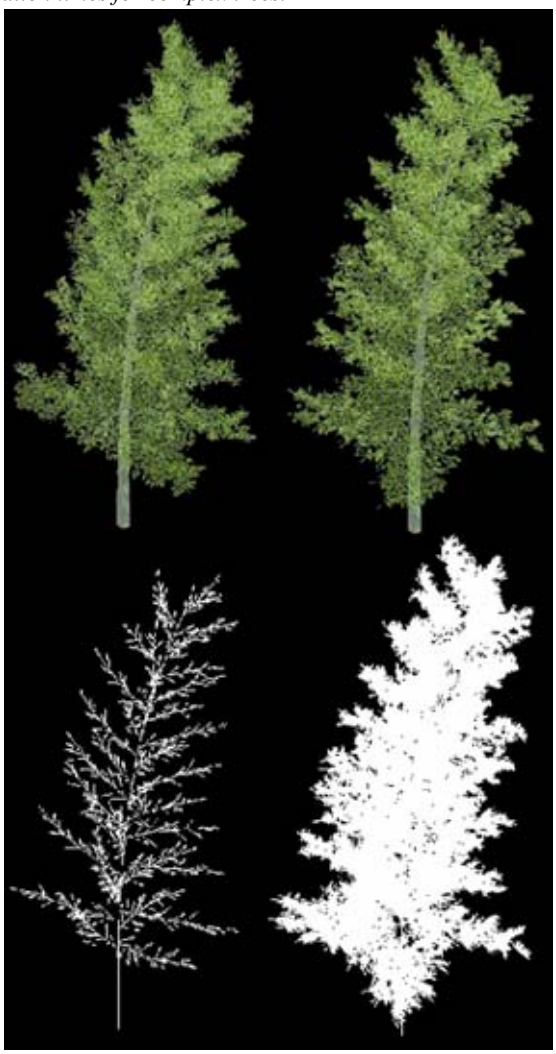

Figure 1. Simplified simulation structures yield similar motions.

This work was supported in part by NSF grant CCR-0220047.

detailed higher-level simulation. With both geometric LODs and SLODs, smooth transitions between the LODs are of key importance. When transitions are not smooth, the "popping" as the LOD changes can be extremely distracting.

Main Results: We present a method for generating and using levels of detail in the simulation of plant motion. We take a plant from its initial representation and automatically create a simplification hierarchy that, when simulating plant/wind interaction, behaves the same as the original in certain key ways. This allows simulation of large groups of plants because those hidden or far away can be significantly simplified yet yield visually realistic results. The simulation is stored as a simple articulated tree structure (generated, for example, by an L-system). The simplified simulation LODs are created by combining either sets of child branches or parent-child branch combinations. The structure for rendering is constructed from the original tree and can be simplified separately from the simulation structure. We also provide for smooth transitions between the SLODs at run-time.

In comparison to other methods for simulation of plant motion, the distinguishing features of our method are:

- We provide a method for automatically building a SLOD hierarchy from the articulated structure.
- We provide an error metric for the simulation error associated with each LOD.
- We provide a method for smooth transitioning between LODs, accounting for issues such as divergent motion among child branches.
- Our method can be applied to portions of a tree or to separate trees, making it fully extensible to arbitrarily large simulation environments.

2 Previous Work

Plant Modeling: Modeling of plant shape has been a major area of research for many years. The most fundamental work in this area has been the work of Lindenmayer and Prusinkiewicz (see, e.g. [PL90]). Lindenmayer systems, or L-systems, form the basis for much of the subsequent plant modeling work. There is a great deal of published work relating to plant modeling, including detailed descriptions of parameters that can be used to define specific trees [Blo85], parameter classes to describe varieties of trees [WP95], and how plants behave together in ecosystems [DHL*98]. This is just a small sampling of the relevant work in these areas. Most of this work, however, has focused primarily on the physical description of static models.

Plant Motion: There has been less work focused on the description of plant motion. Wind is a primary source of force driving plant and tree motion, and has been the subject of study on its own [SF92,WZF*03]. Stam specifically illustrates such wind motion by application to tree branches [Sta97]. Weber and Penn discuss simulating tree motion as a system of oscillators, but do not provide direct examples [WP95]. Sakaguchi and Ohya provide what is probably the most physically accurate model of motion [SO99]. A number of other papers have also dealt with plant motion, either primarily or as a side issue. Because simulation of plant motion is clearly a time-intensive process, several of the papers describing general plant motion have discussed methods for making that motion interactive [PC01,DCF01,EMF03]. A comparison of our method with some of these other methods is provided in section 7.1.

Simulation LODs: SLODs have come into prominence only in the last few years [Ber97,CF97]. They have sometimes taken other names—Endo et al. refer to them as levels of motion detail, or LOmDs [EMF03]. There have been a wide variety of applications for SLODs. These include rigid body dynamics and motion [CIF99,DO01,CAF01], simple collision detection and response [CH97,ODG*03], particle systems [OFL01], and hair [WLL*03]. It is likely that future advances will extend SLOD principles into a number of other areas, as well.

3 Basics and Overview

We give a brief overview of our plant representation and method for simulating plant motion. We follow that with a brief overview of our SLOD method.

Though our implementation is unique, the methods for describing the plants and simulating motion (without any SLODs) are fundamentally the same as what is found in other common implementations. In fact, although we describe our implemented method here, the contributions of this paper would also apply equally well for many possible variations in the form of the model description or the method for simulating motion. For these reasons, we give only a brief review of our implemented approaches, and refer the reader elsewhere for more details of these aspects [WP95,DCF01,SO99].

3.1 Plant Modeling

We describe our plants using an L-system, following the traditional methods for representing plants. The specific L-system grammar we implement is a stochastic turtle interpretation where several parameters can be randomly scaled. From this L-system we generate an articulated plant structure. We also provide a more direct method for generating the articulated structure, following the method of Weber and Penn [WP95]. Our plant structure consists of rigid links connected at joints. Flexible segments are represented as a series of rigid links. We will use standard tree definitions to refer to these segments: a *parent* segment can have one or more *child* segments arising from it. A segment at the edge of the tree (with a leaf attached) will be called *terminal*.

The basic information specified in our grammar includes:

- The lengths of branches
- The orientation of child branches relative to the parent
- The strengths of the joints (equivalent to spring constants, as described below)
- Whether a branch has a leaf attached to the end, and the area of that leaf

This is the information used to generate our SLODs.

3.2 Plant Motion

Plant motion is driven through the application of external forces. Generally, this is specified as a wind field, though there is no reason specific impulses within the plant could not also be applied. The joints of our model are treated as angular springs. Force information is propagated

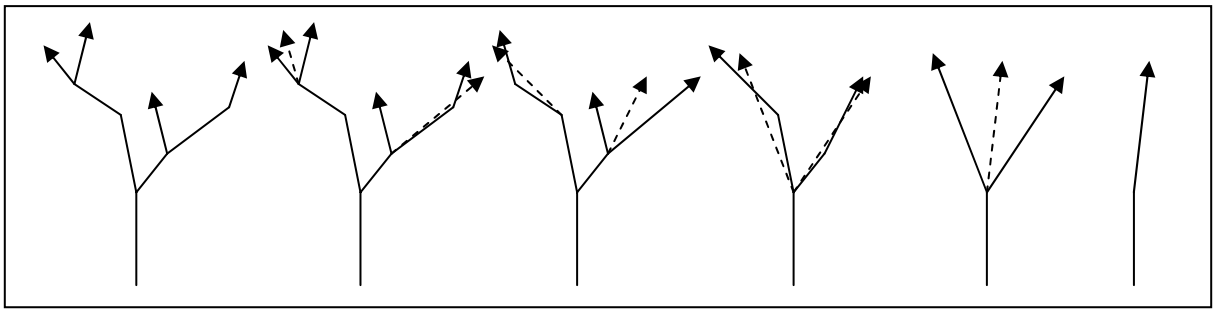

Figure 2. An overview of the SLODs Constructed for a simple tree-like structure. The "given" structure at each stage is shown in solid line, and the simplified structure is shown in dashed line.

throughout the plant. This creates an oscillatory motion governed by the bend strengths defined in our L-system..

Note that we actually specify two springs at each joint. Assuming the parent segment is oriented along the direction **p**, and the child along the direction **c**, one angular spring resists rotation on the axis $\mathbf{p} \times \mathbf{c}$, and the other resists rotation along the axis **p**. (If **p** and **c** are parallel we just use two orthogonal axes perpendicular to **p**.) This effectively allows anisotropic bend strengths, enabling us to model greater resistance to bending in certain directions relative to the parent axis. No third axis is specified, as we do not allow branches to twist about themselves. To be more precise, we should use axes of $\mathbf{p} \times \mathbf{c}$ and $\mathbf{c} \times (\mathbf{p} \times \mathbf{c})$, but this difference is usually minor. In our discussion below and in our implementation, we generally assign both bend strengths to be the same, in order to cut down on the parameter space.

Generally, external force due to wind is applied at the leaves, and is assumed directly proportional to the area of the leaf (and the strength of the wind); direction is also taken into account. This force, applied at the leaf, is then propagated to the parent branch. This propagation continues all the way to the root of the tree. Swaying of the trunk is thus governed by the combined effects of wind at the leaves being propagated to the trunk. We can also add in force due to wind directly hitting branches, but find that the leaf-only approach is better suited for simplification, and we assume that in the following discussions.

We use an Euler integration scheme to update positions at each time step based on the velocity from the previous time step, and update velocities based on the acceleration (force). Euler integration provides good results, but more complex integration schemes could potentially be used.

3.3 Overview of Our Approach

We form a sequence of simulation levels of detail. We follow an approach of working from the outermost branches (connected to leaves) inward toward the root. When we have a parent with a single terminal child, we combine those two segments into a single new (terminal) segment. When we have a parent with multiple terminal children, those children are replaced by a single terminal child. The key to making these replacements is that the substituted segment must behave similarly (within certain error bounds) within the simulation. All of this is done as a precomputation phase. At run-time, the particular amount of error to be allowed in a simulation is determined, and based on this, the precomputed SLOD corresponding to that error level is used. We also take into account how to move smoothly from one LOD to another to avoid popping artifacts in the simulation. Figure 2 demonstrates the SLOD structure for a simplified tree.

It is important to note that the simplification we make is to the *simulation structure*. Geometry can be mapped to the simulation structure (and at the most basic level, they are usually assumed to be directly related), but simplifications in the simulation structure do not change the geometric representation of the object. Thus, the simulation could be simplified to a single segment, while the number of polygons rendered for the plant need not change.

We note that there is no fundamental reason that we must simplify from the outermost node inward. We could allow simplification at any level of the tree, and build up a simplification hierarchy in this way. However, such an approach would entail a significantly more complex data structure than can be achieved by simplifying from the leaves toward the root.

4 Generating SLODs

For a given plant, described as an articulated tree structure, we will generate a set of simulation levels of detail.

4.1 Branch Simplification

At the heart of the simulation level of detail process are the local and global methods of branch simplification. We use two operators on the simulation structure, similar to vertex decimation and vertex combinations used for geometric levels of detail. The first is parent/child combination. Parent branches with only one terminal child branch can be combined to form a single simplified branch. The second operator is child/child combination. Here all terminal children of a branch are simplified to form one new (terminal) child branch. The global simplification scheme requires branches to simplify from the tip to the root of the tree hence both operators require that all child branches be terminal. Figure 2 shows an example of the operators.

Using either of the operators requires that new simplified branches be created. In parent/child combinations the process involves creating a single terminating branch with a leaf area that matches the original. Given the approximate amplitude and frequency of the parent/child segment, the

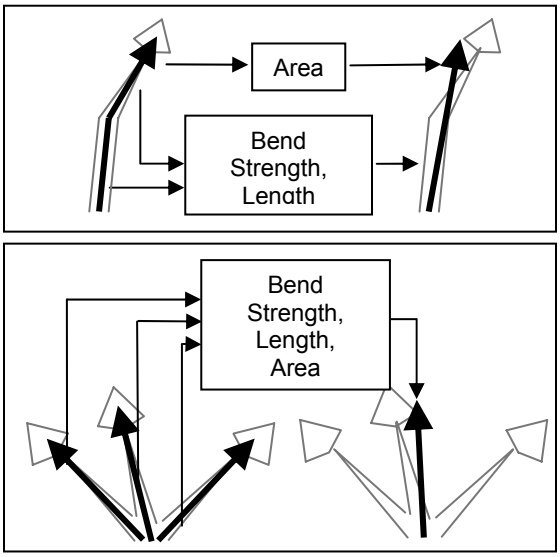

Figure 3. The parent/child (top) and child/child (bottom) combinations. At left is original data, at right is the simplified data. The thick structures show the geometry, the black lines the simulation structure.

lookup table created previously can be used to search for the best single branch parameters (bend strength, and leaf area) that match that amplitude and frequency. That is, we fix the length of the new branch, and determine a new leaf area and bend strength that will closely match the frequency and amplitude of the original pair of branches. Figure 3 (top) illustrates this combination.

Child/child combinations are more difficult. Parameters cannot be directly acquired from the lookup table. The simulation parameters of the children are combined to determine the leaf area, bend strength, and position (including length) of the simplified branch. Leaf areas and length/position are averaged. The bend strengths are determined by a length-weighted average among the children branches. Note that the error introduced in these child/child simplifications is often significantly more than for parent/child combinations (see section 4.3 below). Figure 3(bottom) illustrates this combination.

The force propagated down from a simplified branch must also be modified to make it match that of the branches it replaced. We store a SLOD "propagation factor" (*pf*) for each simplified branch separately from the original *pf*, which is directly derived from bend strengths. When branches are combined to form a new simplified branch, the amount of force passed down to the parents must remain the same. A simplified branch from a child/child combination will pass down a force scaled by the original *pf*'s, taking into account the number of children it replaced. This gives a *pf* equal to the sum of the *pf*'s of the children, multiplied by the original *pf*. A simplified branch from a parent/child combination has a *pf* given by the product of the child's *pf* with the parent's original *pf*.

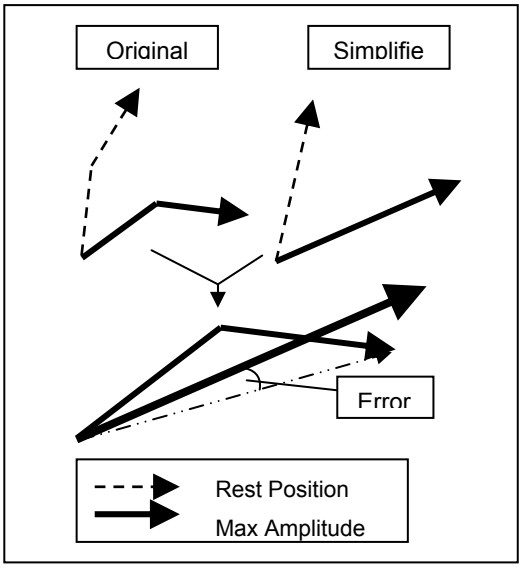

Figure 4. Error computation for parent/child combinations as a function

4.2 Lookup Table Generation

These simulations are performed one time only (not once per plant), and the data is stored for all future simplifications. We thus generate a table, indexed by leaf area and bend strength, that lets us look up single segments with similar amplitude/frequency characteristics for a given pair of segments. This information is used later for determining simplified branches and errors. Since we bound our error under maximal wind force conditions, our error, at least in amplitude, will be even smaller under less force. Although average frequency might vary under different wind conditions, this is not as wide a variation, nor is it as noticeable.

To save time in this process, we first generate lookup tables based on simulations of varied segments. The lookup tables allow us to closely match a simplified branch and leaf to a parent/child branch and leaf. We simulate each combination of a single segment with a single leaf and pairs of branches (one parent, one child, with one leaf) with varying bend strengths and leaf areas. We sample 50 increments of bend strength, and 50 increments of leaf areas. This yields 2500 simulations for branch/leaf and 125,000 simulations for parent/child/leaf. For each simulation, we apply a steady wind force that is fixed as the maximum possible wind force in subsequent simulations. Such simple segments behave as damped oscillators. For each simulation, we determine (and store) the maximum amplitude and the average frequency (period) of the oscillation.

4.3 Error Measurements

Error calculations are the last step of the pre-computation stage. Basically, for each simplified LOD, we need to have a measurement of how much error is incurred in the simulation by using the simplified structure in place of the original. Each of the operators has its own error function. Note that we must accumulate errors as we move through multiple SLODs from leaves to root.

The choice of error metric is somewhat dependent on the goals of the simulation. We have chosen a error involving world space distance between the tips of their branches and their maximum amplitudes. A metric including information such as frequency of oscillation, could also be incorporated. However, we believe the amplitude error on its own is a good metric since it is directly related to the screen space error measured in terms of pixels, which is in turn directly related to the distance of the viewer from the simulation. That is, we have a direct relationship between error and

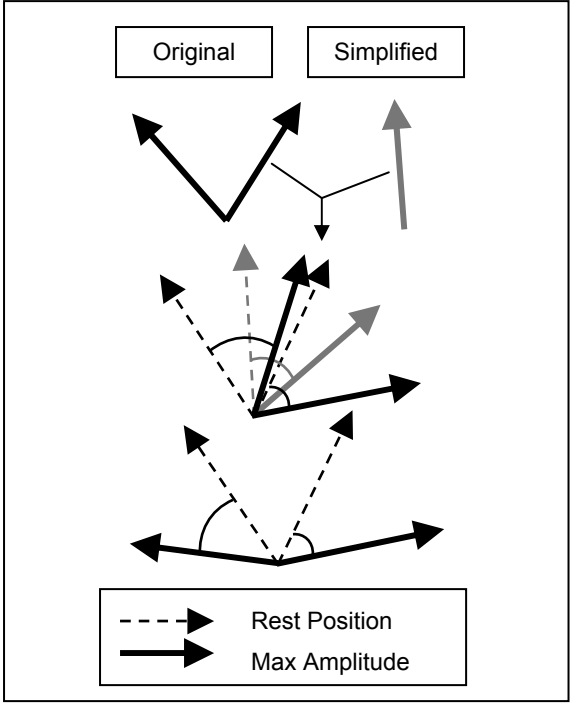

Figure 5. Error measurement for child/child combinations. The maximum amplitude differences for the original and simplified branches (top) are determined as in the middle. Note that children branches can vary independently (bottom).

distance, using a simpler and simpler level as we move farther away. Furthermore, we take frequency into account when determining which simulation parameters to use for the simplified branch (otherwise, we could just match amplitude precisely). Variations in frequency intuitively seem less important than amplitude variations, however, a more involved user study involving perceptual issues (e.g. as in [ODG*03]) would be needed.

Parent/child error is straightforward—we know the difference in amplitude directly from the lookup tables (modified by the segment length). Figure 4 illustrates this process.

Child/child combinations need extra attention. Since one child branch is replacing several children, the error must account for the maximum possible deviation between any two children. Child branches can oscillate out of phase and in opposite directions from each other. Thus, we set the error as follows: Let MO be the maximum amplitude of any of the child branches, MS be the maximum amplitude of the simplified branch. Then the maximum possible error (difference in the simplified position of any point from the original) is 2MO-MS. The error bound thus obtained is an average over time, but is also very conservative (we are extremely unlikely to have branches moving in opposite directions like that). Figure 5 illustrates this further.

Calculating the screen space error at every step for every branch is costly so we usually specify ahead of time a maximum error. This error is often one pixel (very restrictive) though if we desire more rapid progression to lower levels of detail, more error can be allowed—the examples presented below allow 7 pixels to better show off the effects. Given camera parameters, the distance to the camera at which the error is exactly one pixel is stored in the simplified branch and can be quickly checked at run-time. Thus each SLOD links directly to distance from the camera.

5 Using Simulation LODs

At run-time, the tree is traversed and simplified branches are substituted into the simulation structure.

5.1 Run-Time Error Bounds

As previously described, the error associated with each simulation level of detail has already been calculated. It is stored as the distance to the camera in which it is acceptable, using the predefined simulation error of one pixel, to use the simplified branch in place of the original. Note that it is easy to imagine other means of determining acceptable error bounds besides distance. For example, distance outside the view frustum or from a point of focus might be included. Although we focus on distance to the camera, this is not a fundamental limitation of the approach.

Traversing the tree at any branch involves several steps. If the parent has more than one child, the combined children simplification branch must be considered first. If it passes (i.e. the error in that branch is less than the limit) the rest of the tree is represented by this branch. If it fails, the individual children nodes must be examined recursively. If a child is not terminal, the child/parent combination must be considered. Again, if the simplified branch passes, then no further work needs to be done, otherwise you use the parent in the simulation and recursively examine the child.

5.2 Transitioning between LODs

In order to prevent "popping" a smooth transition must occur when switching from the original branches to the simplified ones. The opposite direction is less problematic – replacing a single branch in the simulation with more than one branch merely entails initializing the new branches with the state information (such as current position/orientation/velocity) from the simplified branch.

To describe the transitions, we will describe an idealized situation, however other transitions follow straightforwardly. Assume that the original tree is at the highest level of detail at the minimum camera distance, 0. At some distance D the first simplified branch replaces some number of branches. We thus want to make a smooth transition, from the original simulation set at distance 0, to the first level simplification at distance D. At the distance D, the original branches must be behaving in the same way as the simplified branch.

In the case of a parent/child combination we can make this transition smoothly by interpolating between the key properties:

Node:	Property	Value at distance 0	Value at distance D
Parent	Bend Strength	Original Bend Strength	Simplified Branch Bend Strength
	Angle From Parent	Original Angle From Parent	Simplified Branch Angle From Parent
Child	Bend Strength	Original Bend Strength	MAX Bend Strength

	Angle From Parent	Original Angle From Parent	0 Angle From Parent
Leaf	Area	Original Area	Simplified Leaf Area

At distance D the parent branch and leaf should behave just as the simplified branch and leaf. The child branch should have no angle offset and no angular velocity relative to the parent. At distance D the current position and velocity of the parent branch are transferred to the simplified branch.

For the children combination such an interpolation approach is not possible. Instead, we transition by beginning early simulation of the simplified branch. At distance 0, the simplified branch will be simulated but not processed for rendering. From 0 to D the angle offset of the child branches will be an interpolation of their natural offset and the offset of the simplified branch. At the distance D the simplified branch can now be simulated and processed for rendering in place of the original branches.

Note that although the inverse transition is less problematic, we can have a more cohesive implementation by using this linear interpolation approach in both directions. Thus, moving from a simplified branch toward original branches just interpolates in the other direction.

Also note that if there is rapid motion in the program, we may encounter a problem where we would normally transition between several LODs within a couple of frames. Since this could lead to awkward transition effects, we incorporate hysteresis in the system by forcing transitions from one LOD to another to take place over several frames.

5.3 Combining Simulation and Geometric LODs

Because our SLOD method affects only the simulation structure of the plant, the geometric rendering can be handled as a separate issue. Since the situations where we desire SLODs are often those where geometric LODs are also useful, the combination of these two methods is thus of interest. It would be unfortunate if geometric LODs had to be tied directly to simulation LODs. Optimal geometric LODs might be significantly different from the combined branches in our SLODs. Fortunately, we can treat these two somewhat independently. Note that our current implementation does not incorporate geometric LODs – we present this to highlight the issue.

Usually, the geometric LOD and the simulation LOD are closely tied together at the most detailed level (usually the simulation is based on the geometric information). As the two LOD approaches diverge, however, we need to be able to map from the simulation structure to the geometric structure in order to determine what needs to be drawn on screen. This is relatively easily handled, though at a cost of greater storage overhead, and some additional run-time computation.

Basically, each geometric primitive must be able to update its position from the simulation structure. If we consider the most detailed geometric level, it is easy to determine a new position for each point based on any simplified geometric structure. The unsimplified regions of the model have a clear correspondence, and the simplified portions can be derived directly from the orientation information of the simplified branch (treating the geometry from that point on as static).

The key, then, is to relate the simplified geometric LOD to the original geometric mesh. This will likely involve some additional storage overhead, and some additional run-time computation, such as averaging between points in the original mesh. Nevertheless, it offers an opportunity for the geometric LOD to be dealt with independently.

6 Implementation and Results

The method that we have described here has been implemented in C, and tested on an AMD Athlon 2700 with 512 MB of RAM and a GeForce Ti 4200 video card. Our implementation allows for the description of a general L-system, with parameters allowing us to define both tree/plant shape and bend strengths. Our system is general, allowing us to bypass the L-system description if desired; e.g. we can (and did) adapt the parameters of Weber and Penn [WP95] to model specific species of trees.

Figures 1, 5 and 6 demonstrate that our approach is an effective method for achieving faster overall simulation while minimizing the visible error in simulation. In addition, the supplemental video demonstrates our approach in action.

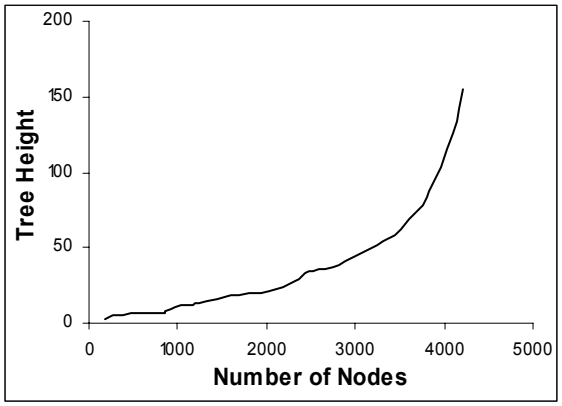

Figure 6. Tree height needed to maintain one pixel error.

As far as timings, we first need to emphasize that plant/tree motion simulations can vary from the very simple to the very complex. Regardless of the simulation method used, it is possible to give enough trees with enough complexity with the right view to make the simulation run slower than any given speed (or conversely, to introduce more error at a fixed speed). Likewise, if simple enough plants are used, any method will seem fast. For this reason, we discuss our SLOD method in terms of the relative reduction in the total amount of simulation required—this should directly reflect performance improvement, regardless of the system described.

To give some sense of the timings, and the speedups obtained through our method, refer to Figure 6. As can be seen there, the use of SLODs allows significant improvements in the simulation frame rates achieved.

Using SLODs does entail additional storage cost. In effect, each branch (or cluster of branches) must store the parameters for a simplified branch anchored at that location. The total amount of storage required for the SLODs, though, is less than the amount of original data –

i.e. an SLOD-enhanced tree uses less than twice the storage of the original tree. This does not include space for the lookup tables, which are not associated with any single plant and are not used at run-time.

7 Conclusion and Future Work

We have presented a method for computing and using simulation levels of detail in the simulation of plant motion. Our method allows us to precompute SLODs with guaranteed error bounds. At run-time, we can create smooth transitions between LODs to achieve faster simulation times, with guaranteed bounds on the error.

7.1 Comparison with Other Methods

We briefly compare our method with some of the more prominent prior methods. Sakaguchi and Ohya provide a more detailed model of motion than we use, however they provide no SLOD-like improvements [SO99]. The work of Perbet and Cani [PC01], while certainly implementing a SLOD approach, is limited to the motion of simple grass. Similarly, the level of detail provided in the work of Endo et al. is geared toward less complex plants, and does not support very complex LODs [EMS03].

Our work is perhaps most similar to that of Di Giacomo et al. [DCF01]. Both approaches use a similar physical motion model, use a blend between LODs, allow for multiple LODs in the same tree, and use a "branch to root" simplification method. There are several major differences, however. We obtain a hierarchical physically-based simulation, whereas they switch from a single physical simulation to procedural (and then to static). We maintain an error measurement for the simplification, giving more rigorous control of the error introduced. Finally, by gradually degrading our physical simulation (as opposed to swapping with a procedural animation) we allow the simulation state to blend over time more smoothly.

7.2 Future Work

Our method is actually very modular—we can substitute a different motion simulator, error metric, or structure generator without affecting the fundamental idea. This leaves a number of avenues open for future work.

- The motion model we have implemented assumes spring-connected rigid bodies. We may be able to achieve better motion results by modeling branch segments as oscillating flexible rods.
- Error computation in our method is conservative. It may be that by considering relationships between errors, we can achieve greater simplification without introducing noticeable simulation changes.
- As stated before, we can take wind effects directly on branches into account in our motion simulator, but this is not currently incorporated into our simplification scheme.
- Exploration of different error metrics and methods for setting error bounds would be interesting.
- We do not incorporate collision detection into our current approach. On complex trees, this may be too computationally intensive, but is worth exploring.
- We do not include procedural animation as an option in our current simulation, however we see no fundamental reason it could not be included in a similar manner as Di Giacomo et al. [DCF01].

References

[Ber97] Berka, R.: Reduction of Computations in Physics-Based Animation Using Level of Detail. 13th Spring Conference on Computer Graphics, ed. Wolfgang Strasser (1997). pp.69-76.

[Blo85] Bertails, F., Kim, T-Y., Cani,, M-P., Neumann, U.: Adaptive Wisp Tree: A Multiresolution Control Structure for Simulating Dynamic Clustering in Hair Motion. *Proc of the 2003 ACM SIGGPRAH/Eurographcis Symposium on Computer Animation* (2003). pp. 207-213.

[Blo85] Bloomenthal, J.: Modeling the Mighty Maple. *Proceedings of ACM SIGGRAPH*, (1985).

[CH97] Carlson, D., Hodgins, J.: Simulation Levels of Detail for Real-Time Animation. *Proc. of Graphics Interface* (1997).

[CAF01] Chenney, S., Arikan, O., Forsyth, D.: Proxy Simulations for Efficient Dynamics. *Proc. of Eurographics 2001*, Short Presentations, (2001).

[CIF99] Chenney, S., Ichnowski, J., Forsyth, D.: Dynamics Modeling and Culling. *IEEE Computer Graphics and Applications (March/April 1999)*, pp. 79-87.

[CF97] Chenney, S., Forsyth, D.: View Dependent Culling of Dynamic Systems in Virtual Environments. Proc. of Symposium on Interactive 3D Graphics. (1997), pp. 55-58.

[DCS*02] Deussen, O., Colditz, C., Stamminger, M., Drettakis, G.: Interactive visualization of Complex Plant Ecosystems. Proc. of conference on Visualization '02 (2002) pp. 219-226.

[DHL*98] Deussen, O., Hanrahan, P., Lintermann, B., Mech, R. Pharr, M. Prusinkiewicz, P.: Realistic Modeling and Rendering of Plant Ecosystems. *Proc. of SIGGRAPH '98* (1998) pp. 275-286.

[DCF01] Di Giacomo, T., Capo, S., Faure, F.: An Interactive Forest. *Proc. of Eurographics Workshop on Computer Animation and Simulation* (2001), pp. 65-74.

[DO01] Dingliana, J., O'Sullivan, C.: Levels of detail in physically based real-time animation. *ERCIM News (special issue on Computer Graphics and Visualization)*, No. 44. (January 2001).

[EMF03] Endo, L., Morimoto, C., Fabris, A.: Real-time Animation of Underbrush. *Proc. of 11th*

International Conference in Central Europe on Computer Graphics, Visualization, and Computer Vision (2003).

[OFL01] O'Brien, D., Fisher, S., Lin, M.: Automatic Simplification of Particle System Dynamics. *Proc. of IEEE International Conference on Computer Animation* (2001), pp. 210-219.

[ODG*03] O'Sullivan, C., Dingliana, J., Giang, T., Kaiser, M.: Evaluating the Visual Fidelity of Physically-Based Animations. *ACM Transactions on Graphics (Proceedings of Siggraph 2003)*, (August 2003), vol. 22, no. 3, pp.

[Ono97] Ono, H: Practical experience in the physical animation and destruction of trees. *Eurographics Workshop on Animation and Simulation* (1997), pp. 149-159.

[PC01] Perbet, F., Cani, M.: Animating Praries in Real-Time. *Proc. of ACM Symposium on Interactive 3D Graphics* (2001).

[PL90] Prusinkiewicz, P., Lindenmayer, A.: *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[SO99] Sakaguchi, T., Ohya, J.: Modeling and Animation of Botanical Trees for Interactive Virtual Environments. Proc. of Symposium on Virtual Reality Software and Technology (1999), pp 139-146.

[SF92] Shinya, M., Fournier, A.: Stochastic Motion – Motion Under the Influence of Wind. Proc. of Eurographics (1992), vol. 11, no. 3 pp. 119-128.

[Sta97] Stam, J.: Stochastic Dynamics: Simulating the Effects of Turbulence on Flexible Structures. Proc. of Eurographics (1997), vol. 16, no. 3, pp. C159-C164.

[WLL*03] Ward, K., Lin, M., Lee, J., Fisher, S., Macri, D.: Modeling Hair Using Level-of-Detail Representations. *Proc. Of Computer Animation and Social Agents* (2003).

[WP95] Weber, J., Penn, J.: 1995. Creation and Rendering of Realistic Trees. *Proc. of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)* (1995), pp. 119-128.

[WZF*03] Wei, X., Zhao, Y., Fan, Z., Li, W. Yoakum-Stover, S. Kaufman, A.: Blowing in the Wind. Proc. of Symposium on Computer Animation (2003). pp. 75-85.

Figure 7. Multiple simulation LODs generated for a tree. The original tree had 77,141 branches and simulated at 2 fps. The SLODs shown here (left to right): 25,434 branches, 6 fps; 12,761 branches, 10 fps; 5654 branches, 21 fps; 1709 branches, 62 fps; 714 branches, 81 fps, 110 branches, 85 fps.

Figure 8. A sequence of tree motion (left to right). The last image shows the simulation structure. Note that at this distance, the difference in simulated appearance between the two trees is minor, though the lower LOD is much faster.