Continuous Climbing Vine Growth

CS275 - Artificial Life

Alan Litteneker

Abstract

This report explores simple interactions between climbing vines and their environment, with a specific emphasis on simulating continuous growth and structural support. It further explores the efficacy of the real time modeling of these interactions with a particle plant system and environment voxelization.

Introduction

There has been a tremendous amount of research in mathematics, computer science, and botany into the simulation of plants over the years, ranging from the simplest of algaes to the tallest of trees to the most beautiful of flowers. This research is largely driven by the incredible complexity of plant life seen in nature, for which a number of different models exist including L-systems and particle systems.

Lindenmayer systems, commonly called L-systems, are a famous set of models proposed in the 1960s which attempt to use a set of formal grammatical production rules to discretely model the growth of plants. This abstract formality makes L-systems capable of modeling almost any real plant. However, this power comes at the price of ease of use: L-systems are notoriously difficult to program, complicated to allow interaction with the environment (eg. open L-systems), and the inherently large time steps make them difficult to use in animation.

Particle systems are a type of alternate model first proposed for plant growth in the 1980s. They model plants as a simple set of particle positions, for which the growth rules can be easily engineered by a human programmer. While this allows a significant simplicity advantage over L-systems, it requires that each system be plant type specific, reducing the allowable complexity in the model.

This paper will discuss an implementation effort to attempt to use a particle system to simulate climbing plants with a voxelized environment. Unlike many plant growth systems (including [1] which inspired this work), this implementation focuses on continuous growth, rather than taking large time steps which add entire new plant segments each frame.

Implementation & Project Description

All implementation for this project was done in C++ 11 with openGL. Apart from a small amount of code pulled from some of my own earlier research projects, all of the code for this project was written from scratch by me alone.

The first step was to implement a library to load the environment from an OBJ file and construct a voxel map from it. Below on the left is a render of one of the loaded environment files, containing approximately 1000 polygons. On the right is a render of the same environment voxelized.

Note that the resolution of the voxel map was high enough that even relatively fine details of the environment, such as the barrels and inset of the door are captured. In addition to updating the positions of voxels in the environment, the implementation also contains an on-demand 3D DDA iterator. This algorithm allows for fast and efficient determination of whether a particular ray or segment crosses any occupied voxels.

One last algorithm of note was the closest point bucket fill algorithm. An idea very similar to this was proposed by [2], but not for a real time application. As described below, each growing node must know how close it is to the underlying environmental structure. To accomplish this without performing a massive search each frame, the voxel map preprocesses this query. Each voxel is informed of the location of its closest occupied sibling using a 3D bucket fill algorithm. When the growing bud queries for this property, all that is required is to find the voxel which contains the end of the bud, and find its closest occupied sibling.

The climbing vines were modeled using a hierarchical oriented particle system. This means that the position of each vector is relative to the position and coordinate space of its parent, which allows for easy minimization of curvature. Leaves and branches are also easily modeled in this hierarchical system.

The growth of the vine in this system was modeled using the following rules:

1. No bud growth can intersect the building (ie. any occupied voxel) or any other part of the plant.
2. Reception of light is maximized and proximity to underlying structure is minimized by growing buds.
3. As a bud grows it also creates additional lateral dormant buds along its path.
4. Traumatic reiteration: If no direction of growth is possible for a particular bud (ie. it has grown into a corner), the closest dormant lateral bud is activated. This mechanism also powers branching, as there is a small probability that a lower bud will be activated on creation.

These rules provide a set of mathematical terms in the fitness function of the vine. As each node grows, it randomly considers a variety of small orientation changes. If any of them have a higher fitness score than the score of the current orientation, it will shift.

Results

Despite extensive optimization attempts, the algorithm was never able to obtain frame rates higher than 10 FPS after branching more than 6 times. Moreover, these performance issues only presented themselves when the vine was growing; simply drawing the vine produced upwards of 45 FPS. The reasons for this are still somewhat unclear, but there are several key candidates:

1. Too many random orientation samples were being taken. In order to achieve somewhat reasonable appearance, each growing node required 64 samples per frame.
2. Missing optimizations in the voxel map or DDA iterator. Preliminary tests of these algorithms did not show any significant performance issues, but
3. The number of rendered polygons simply rose too high.

However, despite experimenting with tweaks to all of these, the frame rate was still dismally low. It may be that there is something intrinsically wrong with the entire approach, perhaps that its unbiased stochasticity or lack of stronger particle system algorithmic optimizations. Either way, the performance results of the growth methods proposed here are inconclusive at best, and temperamentally dismal at worst.

Future Work

A more critical problem was with the overall appearance of directionality of growth in some situations. For example, as this is an optimization system which queries the environment directly, the plant had an eery ability to pointedly seek out far away environment structures without any searching, creating a large lapse in realism. However, this is far from an unsolvable problem. A modification towards being more stochastic or heuristic to the growth algorithm ought to at least alleviate these visual incongruities.

References

1. Benes, B.; Millan, E.U., "Virtual climbing plants competing for space," *Computer Animation, 2002. Proceedings of* , vol., no., pp.33,42, 2002.
2. N. Greene. “Voxel space automata: modeling with stochastic growth processes in voxel space,” *Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '89). Pages 175 - 184, 1989.