Procedural Tree Generation

Torsten Gustafsson

January 2016

1 Introduction

This report aims to describe the development-process of a procedural tree generation application made for the course TNM084 in Linköping University. The generation is made using an algorithm called the "Space Colonisation Algorithm".

2 Method

2.1 Development Tools

The application was developed in JavaScript with WebGL and THREE.js [1] and written in Notepad++ [2]. The work have been largely based on Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz article "Modeling Trees with a Space Colonization Algorithm" [3].

2.2 The Space Colonisation Alogithm

When a tree is created, several parameters are set. A base position (a "tree node") from where the tree will start to grow is set as well as a number of "attraction points" in a spherical space above this start point. Once those points have been created, the algorithm starts working iteratively by creating new tree nodes in the direction of the closest attraction points from each current tree node. See figure 1.

For each iteration every current tree node searches for attraction points in an area around itself defined by the *radius of influence (di)*. If one or more points are found a new tree node will be created in the direction defined by equation 1. Here S(v) represent the points found around tree node v. Basically the direction is the average of all the points found.

$$\hat{n} = \frac{\vec{n}}{||\vec{n}||} \;\; where \;\; \vec{n} = \sum_{s=1}^{S(v)} \frac{s - v}{||s - v||} \tag{1}$$

When the direction have been found, the new node is created at a distance D from the current node in that direction. Attraction points that come within the *kill distance (dk)* from a tree node are removed. Then the iteration process continues until no more nodes are in range of any attraction points.

After every iteration, new branches are created. These branches are represented as cylinders in the scene and will be added as children of their parent branch, with the start node as the root branch. The cylinders are created as "cylinderInstance" objects, which are

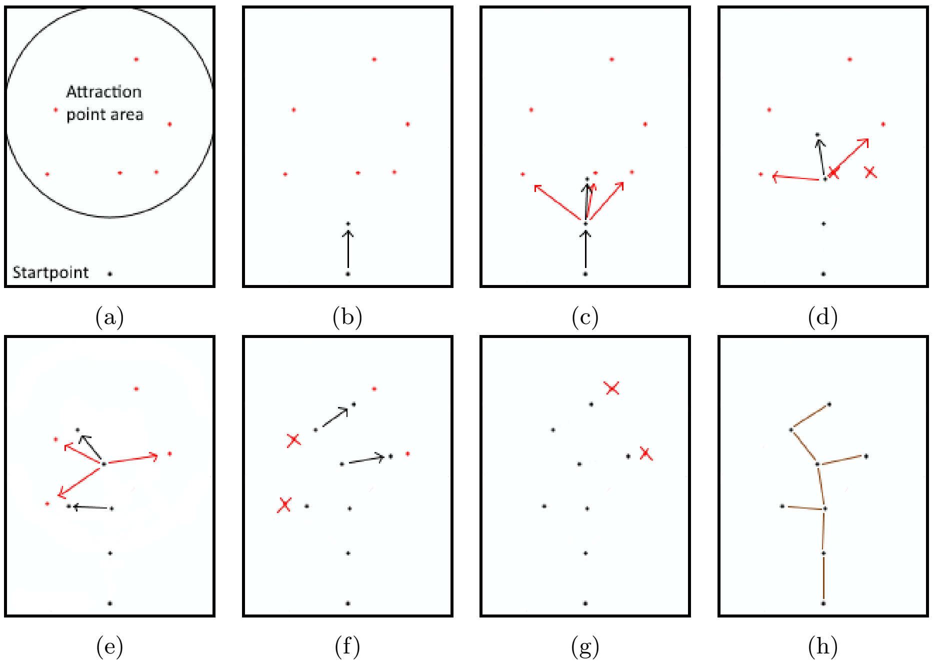


Figure 1: Iteration process of a tree in two dimensions. Attraction points are shown as red dots, and tree nodes as black dots. If no attraction points are close enough to the start point, a new node is created directly over it as seen in figure 1b. Figure 1h shows the resulting tree.

basically THREE.js cylinders with extended functionality. The new branches are rotated in the direction of the newly created tree nodes.

Because every branch is a direct child of its parent, which means it shares its parent's transformation, wind generation can be simulated by adding a small sin-shaped rotation to each branch. When the root node rotates, its rotation will be added to its children which also get their own rotation, adding up to a nice periodic shaking of the tree.

3 Discussion

3.1 Development Process

At first the tree was developed in global coordinates without a child-based system, to get a simple static tree. This went pretty smoothly but when local coordinates were implemented a few problems occurred, mostly due to inexperience, specifically with quaternions. Figures 2, 3 and 4 shows this process. I was stuck a long time with the local transformation problem which led to that other ideas for the project had to be skipped. Ideally I wanted to work more on a realistic wind simulation, and even implement a system where the user use sound as an input to generate trees.

3.2 Memory Deallocation Problem

Javascript does not allow manual memory management, but instead forces the developer to adhere to the garbage collector rules. These rules are not always simple to follow and I didn't manage to deallocate the memory my application used up (which is a lot). If I had more time maybe I would have solved it, but after realising it would take more time than I wanted to, I decided to skip it due to time constraints.

3.3 Final thoughts

I wanted to make this application in Javascript and WebGL because it would be easy to present a live demo of the final product. Everyone with a working browser that can handle WebGL is able to run my application, which is a big plus. Another reason was that I had never worked in WebGL before and I wanted to try it out. There were, however several problems with making an application such as this in WebGL. This application is very demanding on memory and processing power, and without good deallocation the computer might run out of memory very quickly, causing a crash. Had I instead developed this application in C++ for example, this would not have been such a big problem because I am more familiar with how memory management works there.

4 Conclusion

The project resulted in an application that generates trees of varying size and complexity. Figure 5 shows how one such tree may look like. Figure 6 shows the final application, run on a web-browser, in this case Mozilla Firefox.

References

[1] THREE.js is a javascript library that is used to simplify WebGL 3D graphics.
Available: http://threejs.org/ (06/01/2016)

[2] THREE.js is a javascript library that is used to simplify WebGL 3D graphics.
Available: https://notepad-plus-plus.org/ (07/01/2016)

[3] Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz (2007) *Modeling Trees with a Space Colonization Algorithm*, University of Calgary.
Available: http://algorithmicbotany.org/papers/colonization.egwnp2007.html (06/01/2016)

Appendices

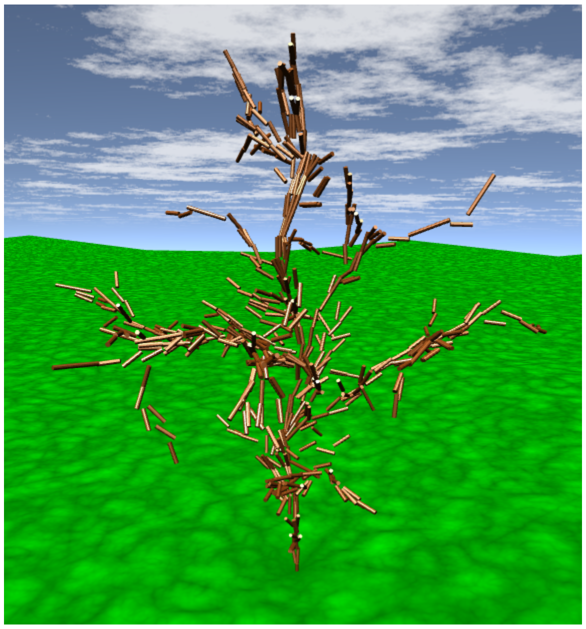


Figure 2: The first working "tree". The branches are not connected correctly and no optimisation was performed, like branch reduction.

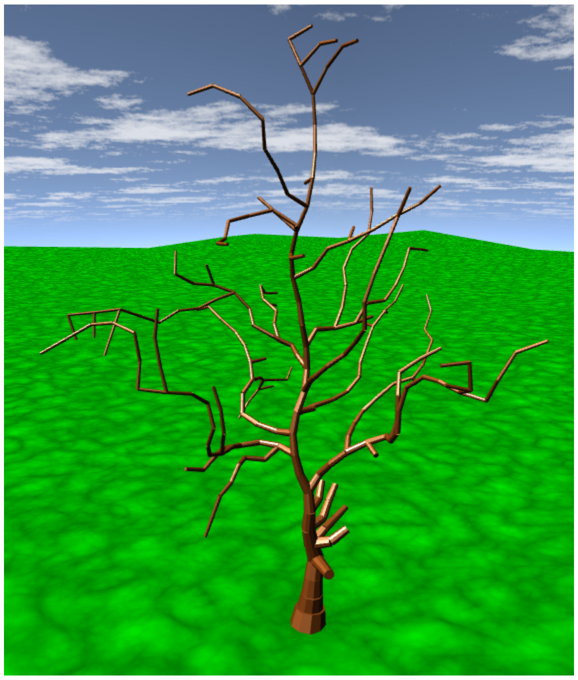


Figure 3: This tree update its branches' radius according to how close to the root they are. Every branch is still added in global coordinates which means no transformations can be performed on the tree as a whole.

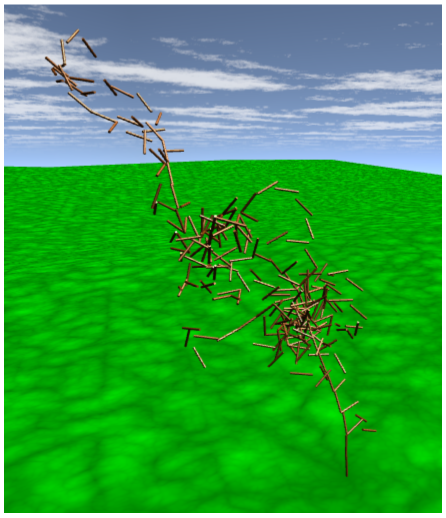


Figure 4: First attempt at creating a child-based tree. Both the local translation and rotation was wrong here, messing the whole structure up.

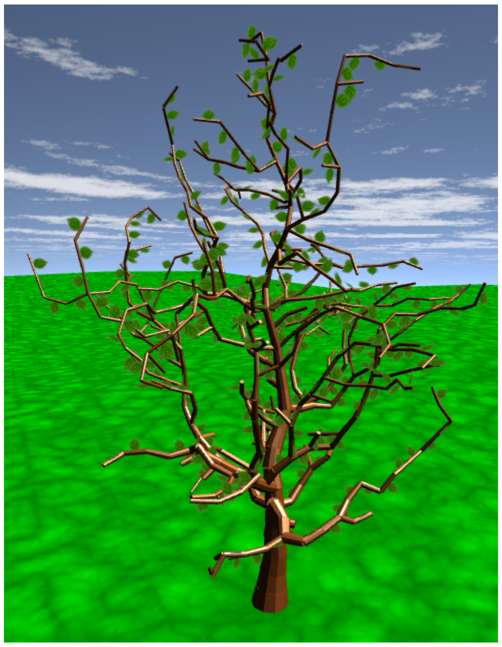
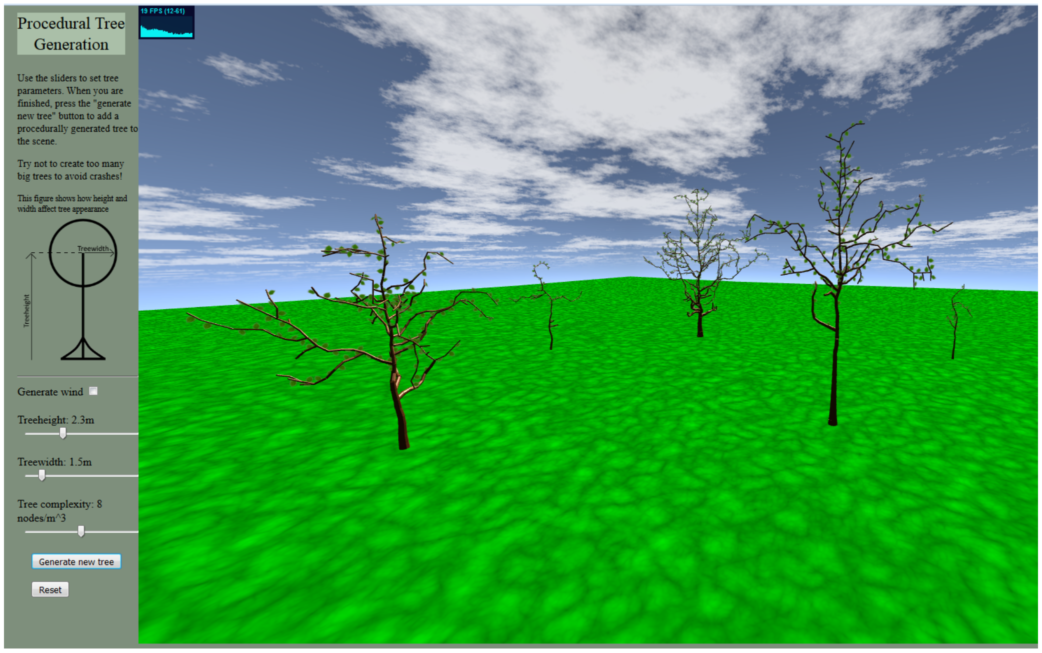


Figure 5: A complete tree with added leaves. Every branch get a leaf with a random rotation around its y-axis. Here every branch is a child of its parent branch.



Procedural Tree Generation

Use the sliders to set tree parameters. When you are finished, press the "generate new tree" button to add a procedurally generated tree to the scene.

Try not to create too many big trees to avoid crashes!

This figure shows how height and width affect tree appearance

Figure 6: The final application. The users may change parameters with the sliders and create their own trees.