Procedural generated trees using L-systems, and further optimizations

Alessandro Dal Corso

DTU Compute · Technical University of Denmark
Kgs. Lyngby, Denmark

Introduction

- We present a way to procedurally generate tree models, using the formalism of L-systems.
- We explain some optimizations that can be performed in order to have the possibility to draw multiple trees without losing real-time interactivity.

L-systems

Grammars

L-systems[1] are a particular kind of system that model the evolution of a *multicellular organism* over time. In addition, L-systems can be employed in modeling realistic representation of more complex objects, like trees.
As we can see in the Figure 1, we can make a system evolve by specifying a set of rules, or productions. Each production is composed of two elements, a predecessor, on the left-hand side of the production, and a successor, on the right-hand side. The production transforms the predecessor symbol into the successor.

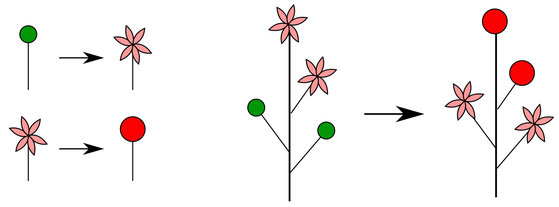

FIGURE 1: Example of grammar-generated state transition. The green circles represent buds, the red ones the young fruits. The other symbols are flowers.

Given an initial state of a system, the transition to another state happens by applying all the valid productions available in parallel.

Parametric L-systems

Formally, a parametric 0L-system is defined as an ordered quadruple of elements $G = (V, \Sigma, \omega, P)$, where V is an alphabet, Σ is a set of formal parameters, ω is a non empty sequence of parameters called the *axiom* (i.e. the initial state) and P is a set of productions.
A production is defined as follows:

$$pred : cond \rightarrow succ$$

Where *cond* is a boolean condition for the transition to trigger. This in practice is used to limit the amount of iterations of the system. As an example, we can see the following production:

$$A(t) : t > 1 \rightarrow S(t+1)BS(t-1)$$

In this case, the production has predecessor $A(t)$, condition $t > 1$ and successor $S(t+1)BS(t-1)$.

The turtle interpretation

When it comes to displaying a 0L-system, the sequence of symbols representing a production is scanned from left to right, with some symbols interpreted as commands to move a LOGO-style turtle in three dimensions. The basic commands include draw a line with width s, $F(s)$, turn left or right $(+, -)$, pitch down or up (&,^) and roll left or right (\,/). The turtle has a position $\boldsymbol{P}$ and an orientation defined by three rotation vectors $\boldsymbol{H}$, $\boldsymbol{U}$, $\boldsymbol{L}$ (head, up and left of the turtle).

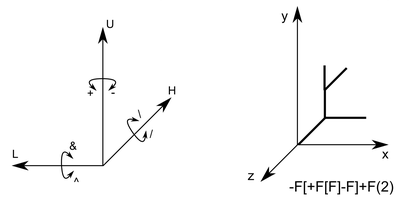

FIGURE 2: The turtle model, with an example of path generated from a string of symbols.

In addition, the current state of the turtle can be pushed or popped in a stack (with [or]), to generate branching structures (see example in Figure 2), as the ones used in the exercise:

$$\begin{aligned} &w : A(100, w_0) \\ &p : A(s, w) : s \geq \min \rightarrow F(s)[+(\alpha_1)/(\phi_1)A(s \cdot r_1, wq^e)][+(\alpha_2)/(\phi_2)A(s \cdot r_2, w(1-q)^e)] \end{aligned} \quad (1)$$

Results

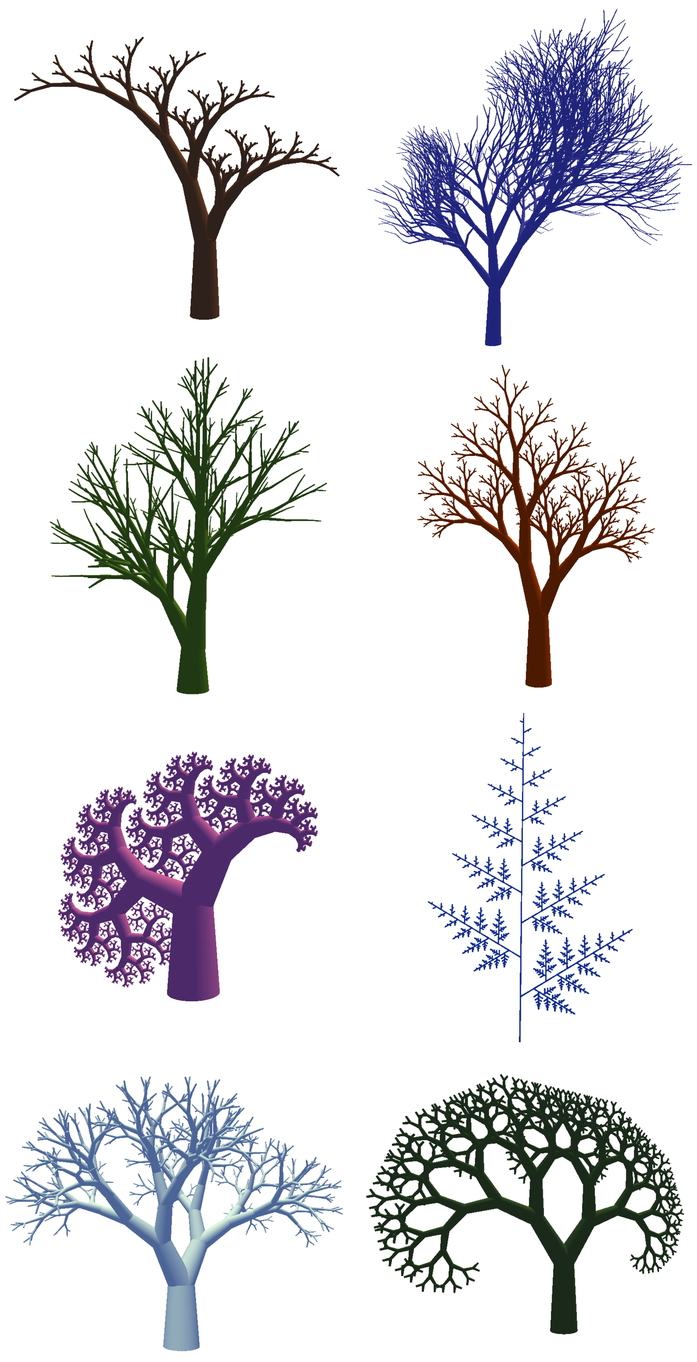

FIGURE 3: Some of the trees generated in our exercise, with different rules and parameters.

Performance

When we are generating the trees in a computer graphics application, we would like to give some thickness to the branches of the tree. That is why every time the L-system draws a line, in our implementation we instead display a truncated cone. This raises some performance problem once we want to display a full scene with many trees maintaining an acceptable frame rate. Most of the techniques in this section are dealt with in [2, Ch. 18] in mode detail.

Triangle strips : Instead of loading into the GPU all the triangle data directly, with an average of 3 vertices per triangle, we employ triangle strips to send, on the long run, only one vertex per triangle on average.

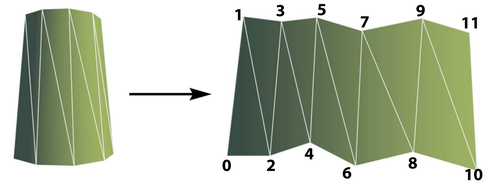

FIGURE 4: Unwrap of a truncated cone, with the vertices' indices shown.

In this mode (`GL_TRIANGLE_STRIP`) each new vertex we tell the graphic card will form a triangle with the two previous ones in the sequence. So, apart from the first triangle, we send one vertex per new triangle. As an example, the cone in Figure 4 will need the following data in memory with `GL_TRIANGLES` enabled:

```
[0 1 2 1 2 3 2 3 4 3 4 5 4 5 6 5 6 7 6 7 8 7 8 9 8 9 10 9 10 11]
```

With triangle strips, instead, the sent data are:

```
[0 1 2 3 4 5 6 7 8 9 10 11]
```

With a reduction of the 60% of the data sent to the GPU.

Interleaving: By packing the data for the same vertex (normals, texture uvs, positions) into structures and then interleaving them, the GPU will find them closer to each other once they are loaded into the GPU's memory. This improves the cache hit average of the GPU, thus improving performance.

Instancing: this technique is to improve efficiency when we want to render a great number of instances of the same model. We use for them the same model all the time, and then we send into the GPU only the data that depend on the single instance, like the model-view matrix, the material, etc. In OpenGL, we can use instancing by calling a special function, `glDrawElementsInstanced`, instead of the usual `glDrawElements`.

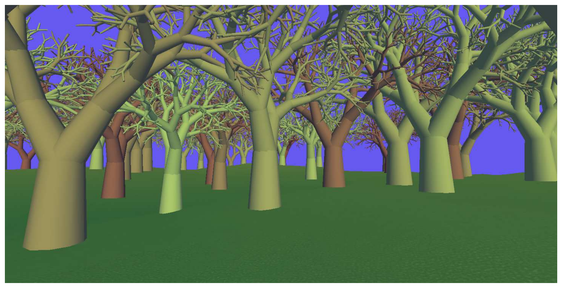

FIGURE 5: Final scene with instancing and element-based interleaving.

References

[1] Przemyslaw Prusinkiewicz, Mark Hammel, Jim Hanan, and Radomir Mech. L-systems: from the theory to visual models of plants. *Plants to ecosystems. Advances in Computational Life Sciences*, 1:1–27, 1997.

[2] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008. ISBN 987-1-56881-424-7. 1045 pp.