Parallel Generation of L-Systems
Markus Lipp1  Peter Wonka2  Michael Wimmer1

Problem Statement
In this work we investigate whether it is possible to efficiently evaluate one of the most classical procedural modeling primitives, L-systems, directly on parallel architectures, exemplified by current GPUs and multi-core CPUs. The main motivation is to enable interactive editing of large L-systems by designers, therefore it is important to speed up the computation of L-systems in order to achieve low response times.

Generating geometry using L-systems consists of two passes: iterative derivation of an axiom string to generate the output string, and interpretation of this string using turtle commands to generate geometry. Although L-systems are parallel rewriting systems, derivation leads to very uneven workloads. Furthermore, the interpretation of an L-system is an inherently serial process. Thus, L-systems are not straightforward amenable to parallel implementation.

Previous Work
Lacz and Hart showed how to use manually written vertex and pixel shaders combined with a render-to-scene technique to compute L-systems [1]. This concept was later extended using automatically generated geometry shaders [2]. Both methods require a shader compilation step for the productions. Further a transformation of bracketed L-systems the following way: We create two work items when a push command is encountered (one for the push and one for the corresponding pop). We use a parallel work-queue approach to distribute work. The difficult part is to quickly find the pop command corresponding to a push command in parallel. The main idea is to extract the push and pop commands from the module string and sort their positions into buckets according to their depths.

Interpretation performance:
- Push and pop commands can not be represented as matrix operations, therefore we parallelize bracketed L-systems the following way: We create two work items when a push command is encountered (one for the push and one for the corresponding pop), and use a parallel work-queue approach to distribute work. The difficult part is to quickly find the pop command corresponding to a push command in parallel. The main idea is to extract the push and pop commands from the module string and sort their positions into buckets according to their depths.

Derivation performance:
- We implemented our parallel algorithms for GPUs utilizing CUDA and for multi-core CPUs using POSIX threads, and compared them against a highly optimized single-core CPU version. The GPU version performs approximately as fast as the multi-core version running on a quad-core. Both versions are about 4 times faster on most L-systems compared to the single-core version. In context-sensitive and stochastic L-systems the single-core version is the fastest.

Results

<table>
<thead>
<tr>
<th>Interpretation performance:</th>
<th>Derivation performance:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push and pop commands can not be represented as matrix operations, therefore we parallelize bracketed L-systems the following way: We create two work items when a push command is encountered (one for the push and one for the corresponding pop), and use a parallel work-queue approach to distribute work. The difficult part is to quickly find the pop command corresponding to a push command in parallel. The main idea is to extract the push and pop commands from the module string and sort their positions into buckets according to their depths.</td>
<td>We implemented our parallel algorithms for GPUs utilizing CUDA and for multi-core CPUs using POSIX threads, and compared them against a highly optimized single-core CPU version. The GPU version performs approximately as fast as the multi-core version running on a quad-core. Both versions are about 4 times faster on most L-systems compared to the single-core version. In context-sensitive and stochastic L-systems the single-core version is the fastest.</td>
</tr>
</tbody>
</table>

| Literature |
|-----------------------------|-------------------------|

Contact
1 (lippwimmer)@cg.tuwien.ac.at
2 pwonka@gmail.com

Supported by:
The Institute of Computer Graphics and Algorithms
Computer Graphics Group