Parallel Generation of L-Systems

Markus Lipp[1] Peter Wonka[2] Michael Wimmer[1]

(a) (b) (c) (d)

This figure shows L-systems generated in real-time utilizing our algorithms. (a) Hilbert 3D space-filling curve, (b) 2D plant, (c) 3D tree, (d) closeup of a Koch curve. Our system can work with parametric, context-sensitive, bracketed and stochastic L-systems and we are able to generate up to 198,000 L-system modules per millisecond. Our algorithm is efficient in the sense that it requires no explicit inter-thread communication or atomic operations, and is thus completely lock free.

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.

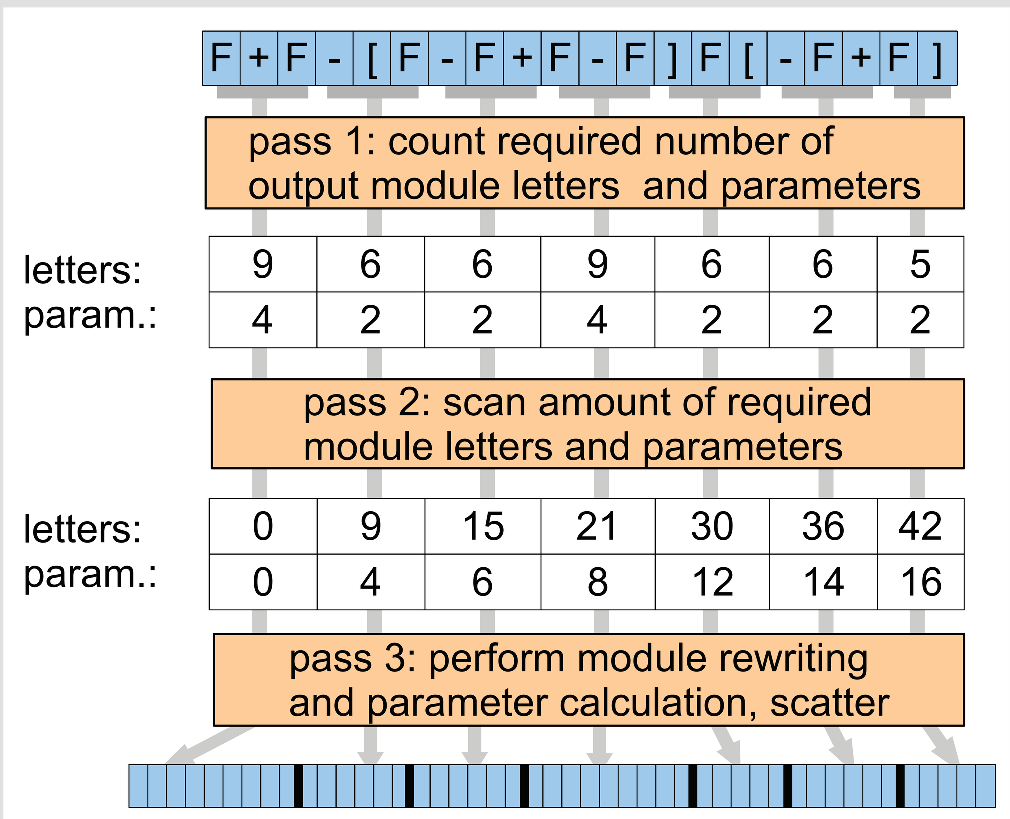
Generating geometry using L-systems consists of two passes: Iterative ***derivation*** of an axiom string to generate the output string, and ***interpretation*** of this string using turtle commands to generate geometry. Although L-systems are parallel rewriting systems, derivation leads to very uneven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

Previous Work

Lacz and Hart showed how to use manually written vertex and pixel shaders combined with a render-to-texture loop 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 step of every production's successor to a set of successors is needed to allow independent parallel execution in a shader. This is only valid if the successor does not have any effect on the traversal state, which is not generally the case.

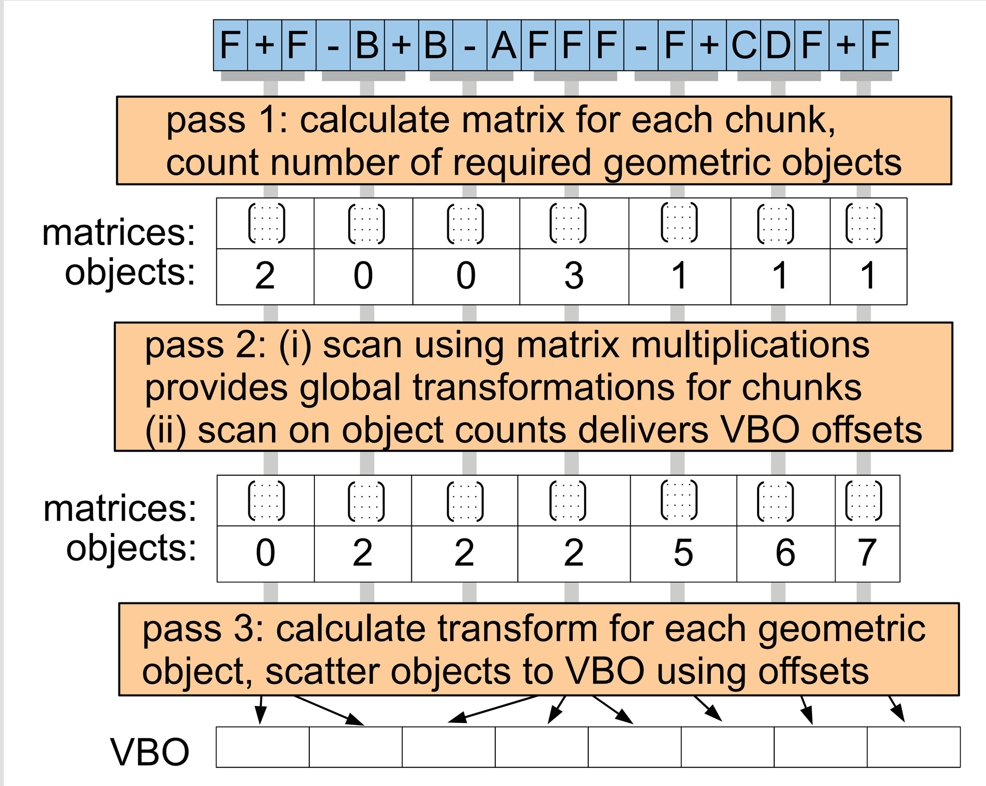
Derivation

Three passes are performed for each iteration: First, the chunks of the input string are evenly assigned to threads. Every thread calculates the output length of its assigned chunk. Second, a parallel scan operation yields offsets for each thread to access the output string. Finally, the replacement of string characters according to the L-system rules is performed to generate the output string.

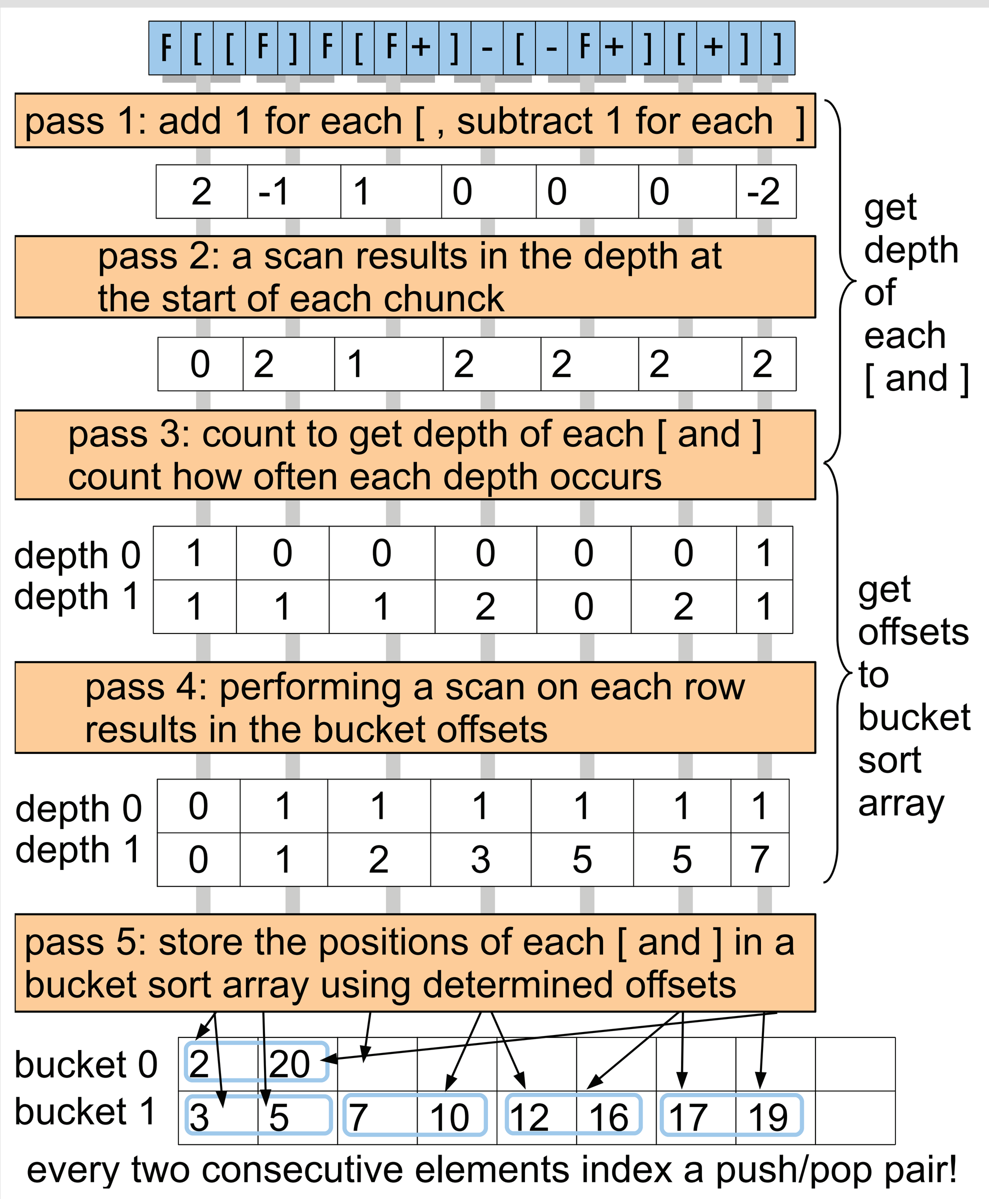


Interpretation

We need to differentiate between unbracketed and bracketed L-systems. Bracketed L-systems additionally utilize stack push "[" and pop "]" commands.
For unbracketed L-systems we represent the turtle state as a 4 by 4 matrix and every turtle command as a matrix multiplication, enabling the following algorithm: Every thread calculates a local combined matrix for the assigned chunk. Then a parallel scan operation using matrix multiplication as operator is performed to get the global position of each chunk. Finally, the geometry is generated for each chunk.



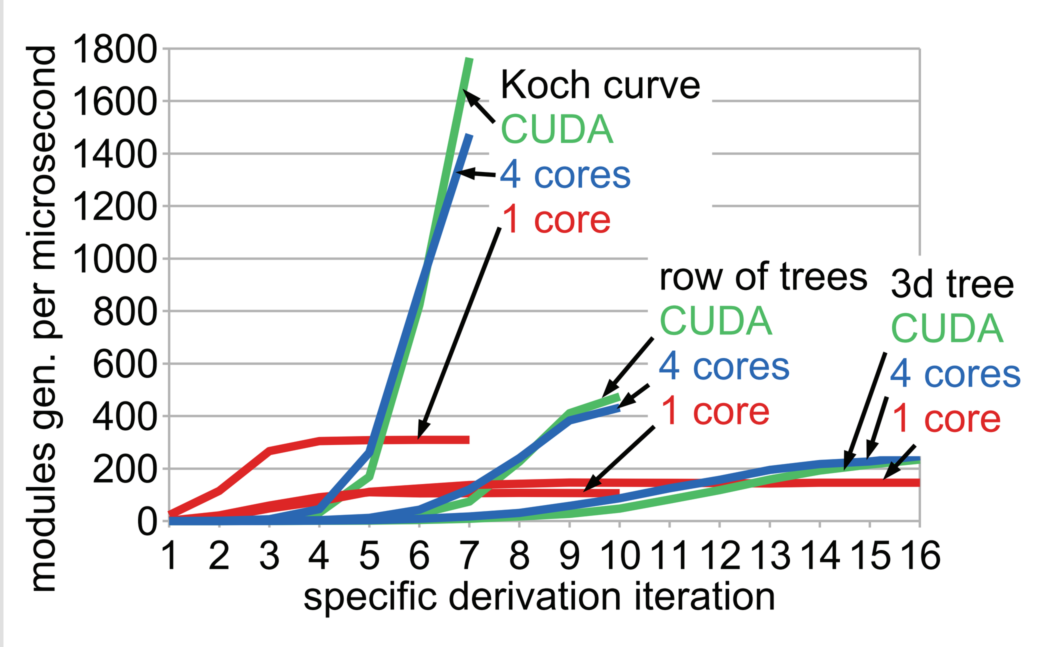
Push and pop commands can not be represented as matrix operations, therefore we parallelize bracketed L-systems the following way: We create two work items when a push command is encountered (one for the push and one for the corresponding pop), and use a parallel work-queue approach to distribute work. 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.



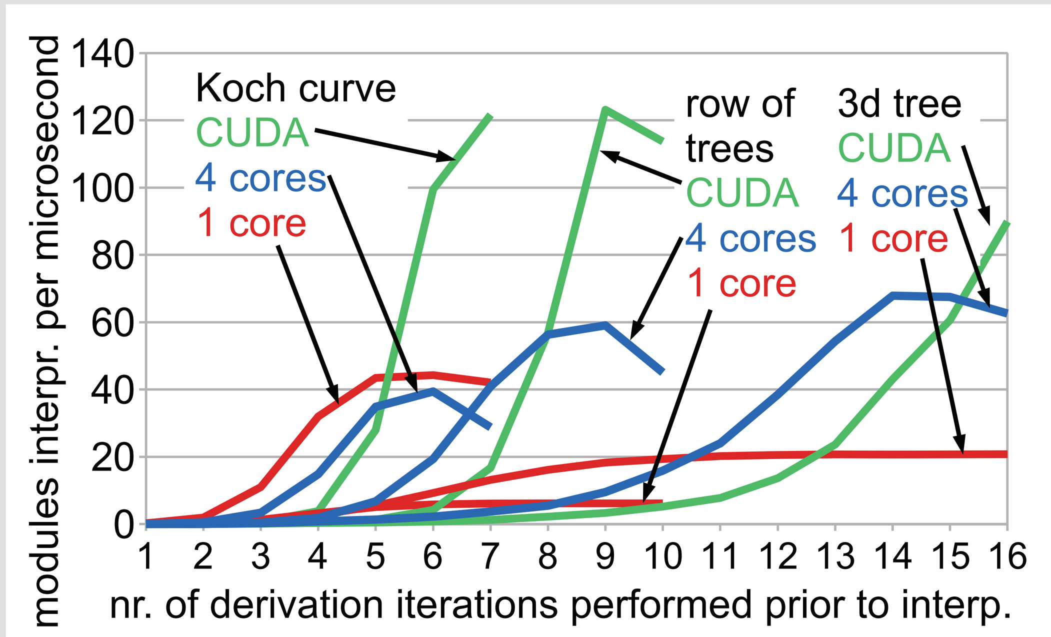
Results

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.

Derivation performance:



Interpretation performance:



Literature

[1] LACZ, P., AND HART, J. 2004. Procedural geometric synthesison the gpu. In Workshop on General Purpose Computing on Graphics Processors, ACM, NY, USA, 23–23.

[2] MAGDICS, M. 2009. Real-time generation of l-system scene models for rendering and interaction. In Spring Conf. on Computer Graphics, Comenius Univ., 77–84.

Contact

[1] {lipp|wimmer}@cg.tuwien.ac.at
[2] pwonka@gmail.com

TU VIENNA Technische Universität Wien Vienna University of Technology

1 The Institute of Computer Graphics and Algorithms Computer Graphics Group

2 Arizona State University

Supported by: FFG