Fast, Parallel, GPU-based Construction of Space Filling Curves and Octrees

Prekshu Ajmera Rhushabh Goradia Sharat Chandran
Indian Institute of Technology Bombay
Web: www.cse.iitb.ac.in/∼{prekshu,rhushabh,sharat}

Srinivas Aluru
Iowa State University
www.ee.iastate.edu/∼aluru/home.html

Abstract

Space Filling Curves (SFC) are particularly useful in linearization of data living in two and three dimensional spaces and have been used in a number of applications in scientific computing, and visualization. Interestingly, octrees, another versatile data structure in computer graphics, can be viewed as multiple SFCs at varying resolutions, albeit with parent-child relationship.

In this paper we provide a parallel implementation of SFCs and octrees on GPUs that rely on algorithms designed to minimize or eliminate communications.

INTRODUCTION: Construction and traversal of the ubiquitous octree on a single processor CPU is well understood; the natural top down hierarchical structure provides a partitioning of space, eliminating the need to deal with unnecessary portions of large volumes of data. Recently, with rapid improvements in the performance and programmability, Graphic Processing Units (GPUs) containing parallel computational units have fostered considerable interest.

PRIOR WORK: A straightforward implementation of the top-down octree for the GPUs appears in [Lefebvre et al. 2005]. Here pointers simply become indices within a texture, and are encoded as RGB values. The content of the leaves is directly stored as an RGB value within the parent node's array of pointers. The alpha channel is used to distinguish between a pointer to a child and the content of a leaf. When computations are required in parallel, the top down structure creates a bottleneck at the root, and at higher levels especially since the data resides at the leaves. There is no communication free knowledge of where a particular data node is going to land in the texture.

CONTRIBUTIONS: Maximizing performance on a GPU requires an algorithm that is relatively communication free. The large number of threads available on current GPUs should be used efficiently to work at the leaf level of the octree, and 'somehow' encode the octree structure in a data parallel manner. In this paper, we use SFCs as a data representation of octrees and show how to support typical queries. Specifically

1. Multiple GPU threads compute the SFC code of the domain at the highest resolution in parallel. This is repeated across multiple scales to produce (in parallel) all nodes of the octree.

2. Once a multilevel structure is in place, we support parallel queries. For example, a seemingly sequential operation, the post order traversal of the octree is obtained in parallel. Other parallel queries supported include nearest neighbor queries, and the least common ancestor of two nodes.

PROBLEM SETTING: For brevity we assume that the data of interest is available as points in a domain. For example, these could be the intersection of the surface of the Stanford bunny with cells that are obtained by bisecting the bounding cube recursively along each dimension $k \geq 1$ times (this will normally eventually result in an octree with height k, and a grid with 8^k cells). We make no assumption on the number of points in the model. However, memory limitations of the GPU will possibly result in multiple points within a cell, when an octree of height $h < k$ is constructed.

SFC CONSTRUCTION: Consider a d dimensional particle space of side length D with the bottom left corner as origin. Threads in parallel compute the relation of points to cells at resolution k. (Integer coordinates of a cell having a point at (P_x, P_y, P_z) will be $(\lfloor 2^k P_x/D \rfloor, \lfloor 2^k P_y/D \rfloor, \lfloor 2^k P_z/D \rfloor)$). The SFC value is then computed by representing the coordinates of the cell using k bits for each dimension and interleaving these bits. (The SFC index of, e.g. a cell with coordinates (3, 1, 2) = (11, 01, 10) is 101110)

OCTREE CONSTRUCTION: We first observe that if the computed SFC values (say at resolution k) are sorted, then we have the correct order to consider nodes in a bottom up traversal of an octree. This leads us to the intuition that an octree can be viewed as (sorted) SFC values at varying resolution. Further, discarding the least significant d bits from an SFC value of a cell gives us the SFC value of the parent cell. It is therefore possible, in a communication free manner, for parallel threads to consider the data values of eight SFC cells to make a decision about a parent cell. For example, if all eight cells are empty, the leaf is promoted to a higher level in a bottom up manner.

QUERIES: Given a node (and thus the SFC value), it is straightforward to compute its parent. A more complicated query is to find the least common ancestor of two nodes c_1 and c_2. This can be done in parallel, and in a communication free manner for multiple pairs, by finding the longest common prefix of the SFC values of each pair that is a multiple of d. The postorder traversal is another query. For all nodes in the octree, we first find the post order number (PONA) in a notional (non-existing) complete octree. During the bottom up octree construction algorithm, we also store, for each node, the number of empty nodes (NE) in its subtree. These are nodes that should have existed, but do not exist, because the octree is adaptive, and leaves are at various levels. The position of the given node in the final postorder traversal is then simply the difference PONA-ΣNE of nodes appearing before the current node.

RESULTS: The octree of height 11 for 5 million points can be done in 0.605ms (the comparable number for a CPU is 5244 ms). Other results (for example, for varying octree heights) and source code are available at http://www.cse.iitb.ac.in/∼rhushabh/i3d. Our implementation uses [CUDA] on an nVidia GeForce 8800.

References

CUDA. Nvidia Compute Unified Device Architechture (CUDA) Programming Guide. `http://developer.nvidia.com/cuda`.

LEFEBVRE, S., HORNUS, S., AND NEYRET, F. 2005. *GPU Gems 2*. Addison Wesley, ch. Octree Textures on the GPU, 595–614.

I3D 2008, Redwood City, California, February 15–17, 2008.
ACM 978-1-59593-983-8/08/0002