# Fast, Parallel, GPU-based Space Filling Curves and Octrees

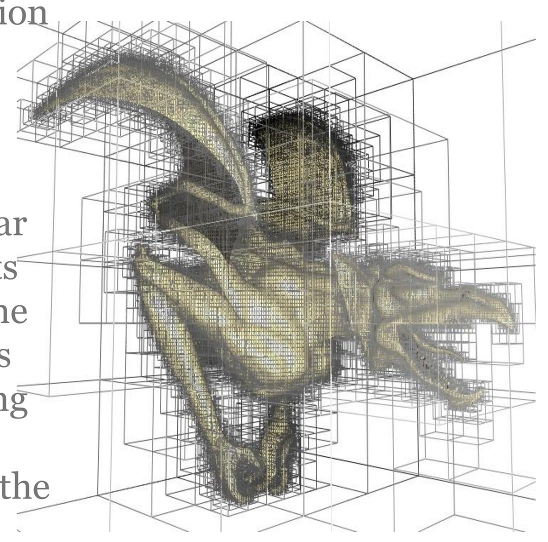
Prekshu Ajmera, Rhushabh Goradia, Sharat Chandran, Srinivas Aluru
Department of Computer Science & Engineering, IIT Bombay
{prekshu,rhushabh,sharat}@cse.iitb.ac.in
aluru@iastate.edu

## Problem Statement

- Given a cube bisected *k* times recursively along each dimension, and a set of points in the cube, generate a Space Filling Curve (SFC) to map each of the voxels to a *1-D* linear ordering, **in parallel on the GPU**
- Construct, **in parallel**, nodes of the octree representing the points. Also support parallel queries

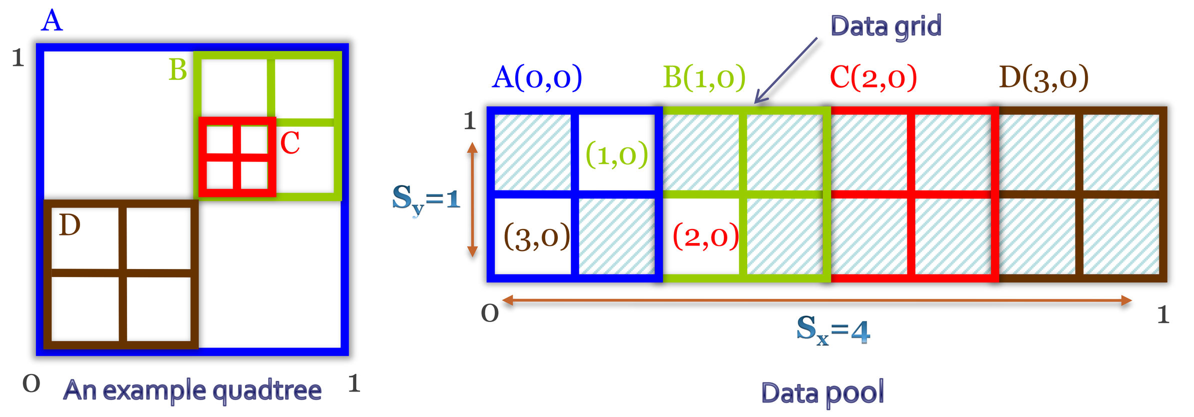
## Motivation

- **Spatial Domain Decomposition (SDD)** refers to the process of spatially partitioning the domain of the problem across processors in a manner that attempts to balance the work performed by each processor while minimizing the number and size of communication
- **SFC** is a key SDD method
- **Application** : SDD is a first step in many particle based methods. In graphics, a triangular element can be represented by its centroid. In the picture [2] on the right, the surface of the dragon is represented by points intersecting a cubic grid cell.
- Octrees are useful in organizing the resultant point set



## Prior Work

- Octrees are represented in the GPU as indexes in a texture[2]



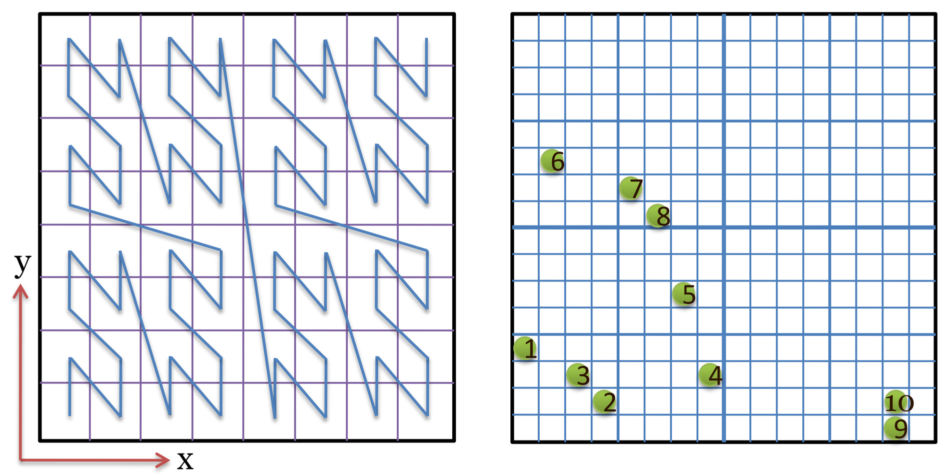

- However, the resulting top-down structure is intrinsically sequential. A bottom up representation (using SFC) can make use of large number of parallel GPU threads

## Contributions

- First parallel SFC construction algorithm on GPU
- Fast, parallel octree on GPU supporting
  - ✓ Parallel Post Order Traversal
  - ✓ Parallel Nearest Neighbor
  - ✓ Parallel Range Queries
  - ✓ Location of the cell containing the queried point
  - ✓ Least Common Ancestor of two cells

## Space Filling Curve (SFC)

A *d* dimensional hypercube bisected *k* times recursively along each dimension, results in $2^{dk}$ non-overlapping hypercells of equal size. The SFC is a mapping of these hypercells to a 1-D linear ordering. We use the z-SFC shown below




On the left we show a *2-D z*-SFC. On the right we show 10 points in a *2-D* space. The points are sequentially labeled in the z-SFC order.

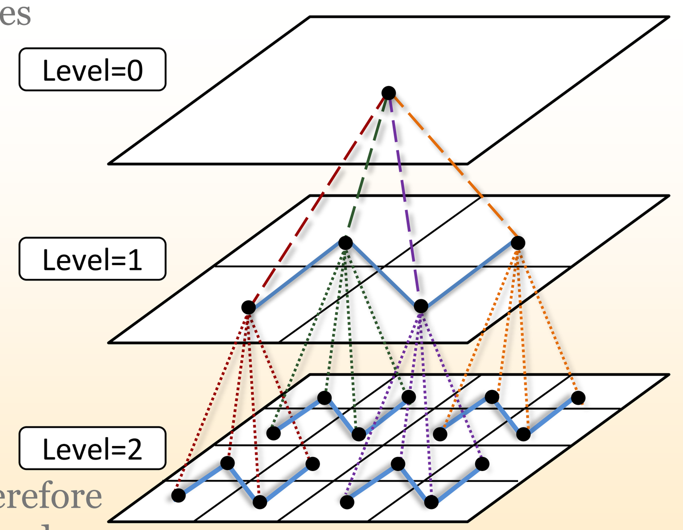
**Merit of SFC ordering:** Partitioning points as per SFC order ensures load balancing. Also, as important we have data ownership, i.e., implicit knowledge of where each point lives

**GPU-based Parallel SFC Construction Algorithm**

1. Consider a 3 dimensional particle space of side length *D* and let its bottom left corner be at the origin
2. **In parallel** do,
   For resolution *k*, integer coordinates of a cell having a point $P(P_x, P_y, P_z)$ is $(\lfloor 2^k P_x / D \rfloor, \lfloor 2^k P_y / D \rfloor, \lfloor 2^k P_z / D \rfloor)$
3. Allocate $8^k$ threads . **In parallel** do
   Interleave each of the k bits of a cell coordinate starting from the first dimension to form a *3k* bit value. For example, SFC value of a cell with coordinates *(3, 1, 2) = (11, 01, 10)* is *101110= 46*

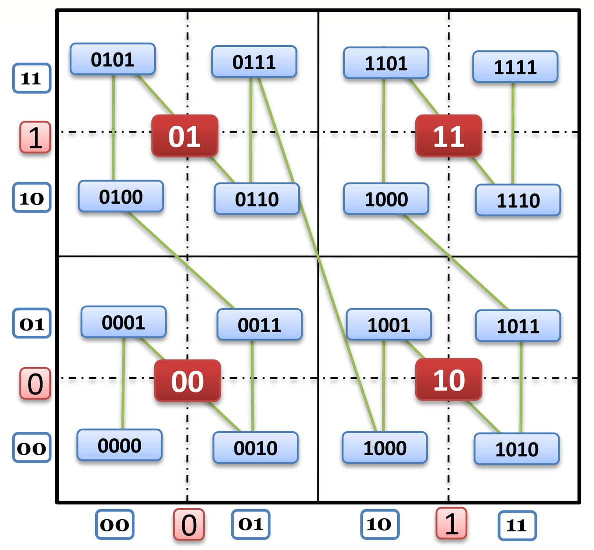
## SFC& Octrees

- If the computed SFC values (at any fixed resolution) are sorted, then we have the correct order to consider nodes in a bottom up traversal of an octree
- Octrees can be viewed as multiple SFCs at varying resolutions
- A linear bottom up octree construction is therefore easy if we follow the SFC order




## Construction of Parallel Octree

- Removing the least *d* bits from the value of a cell gives the value of its parent
- Value of parent cell can be computed independently in parallel

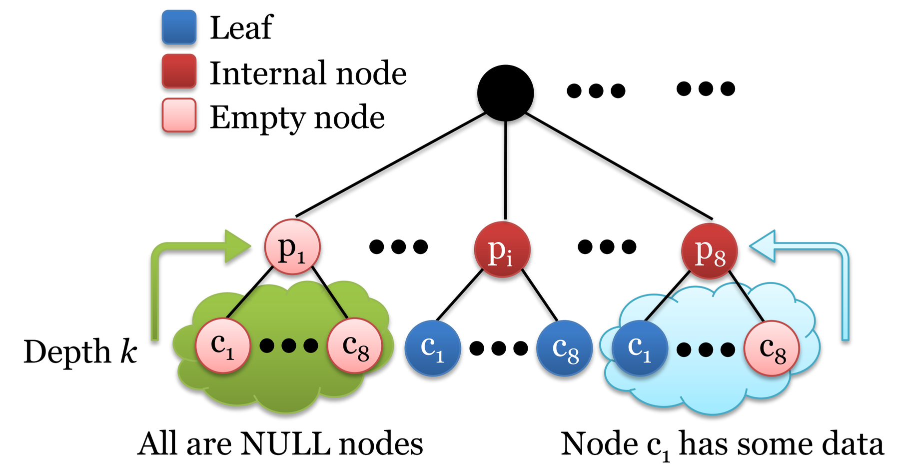



**GPU-based Parallel Octree Construction Algorithm**

**Input :** SFC based sorted ordering of cells at resolution *k*
**Output :** An Adaptive Octree (Leaves present at different levels)

1. Allocate $L_0, L_1, \ldots, L_k$ arrays of sizes $8^0, 8^1, \ldots, 8^k$ respectively
2. Loop for *i=k to i=1*
   a. Allocate $8^{i-1}$ threads
   b. Each thread checks *8* elements in $L_i$ from SFC ids $(8{*}Thread_{id})$ to $(8{*}Thread_{id}+8)$
   c. If all 8 elements are empty then make all the elements NULL and their PARENT at level $L_{i-1}$ as leaf (The *3-D* position of the parent of a node in the upper layer can directly be calculated from the *3-D* position of the child )

**Note**: Implementation is highly data parallel with *zero communication* between the GPU threads




## Typical Queries

We use the bit representation of SFC values

- **Is node $C_1$ contained in node $C_2$ ?**

$C_1$ is contained in $C_2$ if and only if the SFC value of $C_2$ is a prefix of the SFC value of $C_1$

- **Given $C_2$ as a descendant of $C_1$, return child of $C_1$ containing $C_2$**

For dimension *d* and level *l*, *dl* is the number of bits representing $C_1$. The required child is given by the first *d(l + 1)* bits of $C_2$

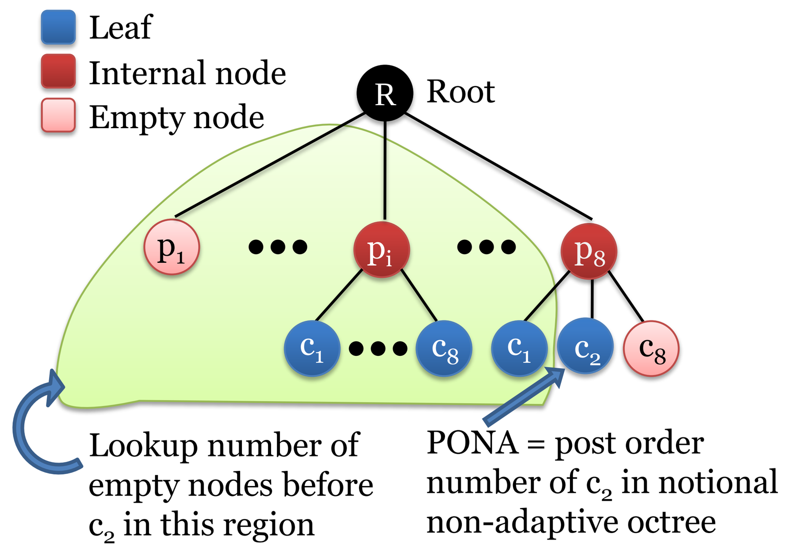
- **What is the Least Common Ancestor of nodes $C_1$ & $C_2$ ?**

The longest common prefix of the SFC values of $C_1$ and $C_2$ which is a multiple of dimension *d* gives us the least common ancestor

**Note:** Computation is directly done on SFC values. Therefore performance loss due to many threads accessing the same node will not occur even if there are multiple queries

- **Post Order Traversal**

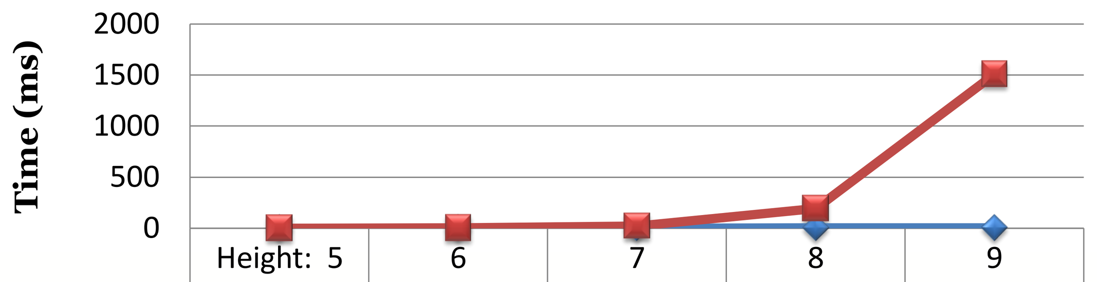
For each node in parallel do
1. Compute post order number *(PONA)* in a notional non-adaptive tree (this is an O(1) computable formula)
2. Lookup previously computed number of empty nodes *(NE)* from a set of nodes that occur before the node in question
3. *PONA* $- \sum NE$ *is* the final post order number of the node in question




## Results

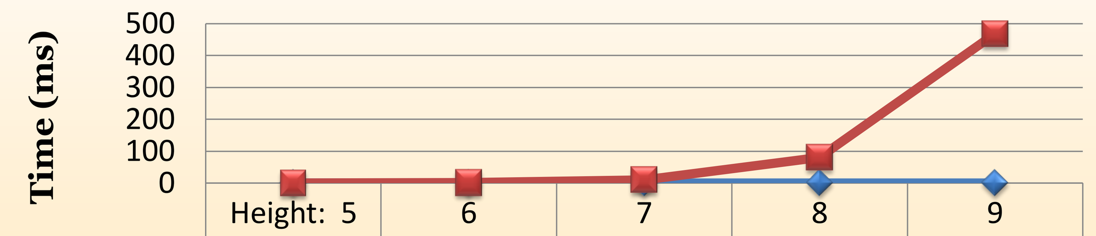
Results were generated on an AMD Opteron 2210, 64-bit dual core CPU & nVidia 8800 GTS using **CUDA** [3]. GPU timings in charts does not include data copy time from CPU to GPU.

**SFC CONSTRUCTION (2 Million Points)**



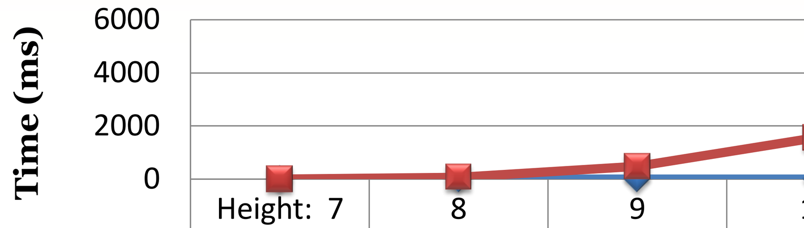
| Height: | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|
| GPU (ms) | 0.068 | 0.087 | 1.181 | 2.106 | 3.123 |
| CPU (ms) | 0.353 | 2.687 | 22.622 | 192.063 | 1510.12 |

**OCTREE CONSTRUCTION (2 Million Points)**



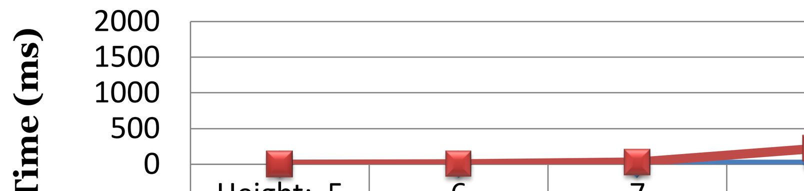
| Height: | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|
| GPU (ms) | 0.27 | 0.31 | 0.316 | 0.321 | 0.354 |
| CPU (ms) | 0.152 | 0.891 | 9.59 | 80.05 | 467.4 |

**OCTREE CONSTRUCTION (5 Million Points)**



| Height: | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|
| GPU (ms) | 0.319 | 0.331 | 0.369 | 0.481 | 0.605 |
| CPU (ms) | 10.01 | 83.1 | 489.42 | 1560.9 | 5244.17 |

**POST ORDER OCTREE TRAVERSAL (2 Million Points)**



| Height: | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|
| GPU (ms) | 0.223 | 0.273 | 1.339 | 2.17 | 3.921 |
| CPU (ms) | 1.27 | 4.57 | 30.23 | 220.13 | 1613.9 |

Similar results were obtained for parallel computation of finding

- Near neighbors for *n* points
- Locations in the octree of *n* query points

We observe that if the problem size is large, GPU vastly outperforms the CPU

## Future Work

Applying the SFC-based constructed parallel octree to an N-body problem for the Global Illumination solution in point models [4] using the Fast Multipole Method on GPU

## References

1. Sagan H. *Space Filling Curves.* Springer-Verilag, '94
2. Lefebvre S., Hornus S., and Neyret F. *GPU Gems 2, chapter* Octree Texture on GPU, pages 595-614. Addison Wesley, '05
3. nVidia CUDA Programming Guide,developer.nvidia.com/cuda
4. Goradia R., Kanakanti A., Chandran S., and Datta A. *Visibility Map for Global Illumination in Point Clouds.* In *Procs. of ACM SIGGRAPH GRAPHITE*, pages 39-46. '07