Global Illumination for Point Models

Rhushabh Goradia
Guide : **Prof. Sharat Chandran**

Fourth Annual Progress Seminar, 2008

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

27.08.2008

Outline

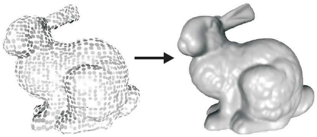
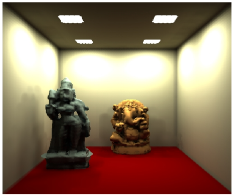
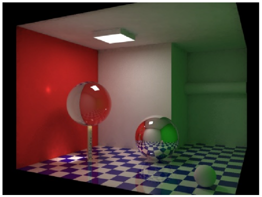
Outline

Problem Definition

Problem Statement

To compute a global illumination solution for complex scenes represented as point-models

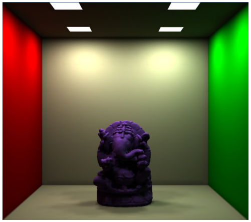
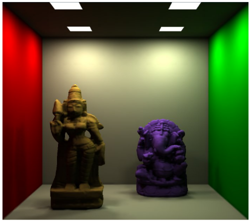
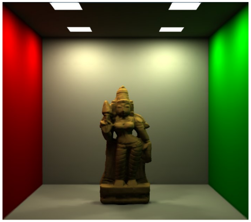
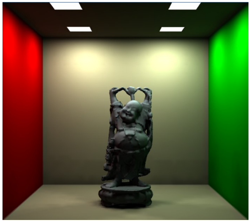
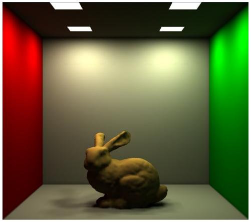
- Diffuse
- Specular

GI Effects: Color Bleeding, Soft Shadows, Reflections and Refractions, Specular Highlights and Caustics

Application Domains

Results: Global Illumination

The Fast Multipole Method (FMM) for Diffuse Global Illumination

- GI is a N-body problem
- FMM is concerned with evaluating the effect of a "set of sources" $\mathbb{Y}$, on a set of "evaluation points" $\mathbb{X}$.

More formally, given

$$\begin{aligned}\mathbb{X} &= \{x_1, x_2, \ldots, x_M\}, \quad x_i \in \mathbb{R}^3, \quad i = 1, \ldots, M, \\ \mathbb{Y} &= \{y_1, y_2, \ldots, y_N\}, \quad y_j \in \mathbb{R}^3, \quad j = 1, \ldots, N\end{aligned}$$

we wish to evaluate the sum

$$f(x_i) = \sum_{j=1}^{N} \phi(x_i, y_j), \quad i = 1, \ldots, M$$

- Total complexity : $O(NM)$

The Fast Multipole Method

$$f(x_i) = \sum_{j=1}^{N} \phi(x_i, y_j), \quad i = 1, \ldots, M$$

- The FMM attempts to reduce this seemingly irreducible complexity to $O(N + M)$.
- The three main insights that make this possible are
  - **Factorization** of the kernel into source and reciever terms
  - Many application domains do not require that the function f be calculated at very high accuracy.
  - FMM follows a **hierarchical structure** (*Octree*)
- Each node has an associated **Interaction Lists**

FMM and GPU

- Besides being very efficient (O(N) algorithm), the FMM is also highly parallel in structure.
- Highly parallel streaming processors on Graphics Processing Units (GPUs) can be used

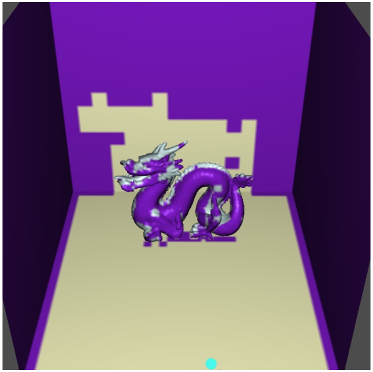
Octrees, FMM and GPU

- Underlying hierarchical structure of *Octree* helps FMM to evaluate solution to N-body problem in just $O(N)$ or $O(N \log N)$ operations
- As a part of parallel FMM framework, GPU-based parallel octree construction algorithm was also designed

Visibility Between Point Pairs

Visibility calculation between point pairs is **essential** to give *correct* GI results as a point recieves energy from other point only if it is **visible**

Why Visibility on GPU ?



Problem Statement and Work Done

Problem Statement

Capturing interreflection effects in a scene when the inputs are point samples of hard to segment entities

Work Done

- Mutual visibility between point pairs: A necessary step for achieving correct global illumination effects **(Done)**
- We compute diffuse inter-reflections using the FMM **(Done)**
- On current machines, time results of CPU-based implementations of visibility and FMM algorithms are not satisfactory
- Parallel implementation of the same performed on GPU (using CUDA) for multi-fold speedups **(Done)**
- Parallel octree construction on GPU which can be used with the GPU-based parallel FMM algorithm **(Done)**

Future Work

To Be Done

- Inter-reflection effects include both diffuse **(Done)** and specular effects like reflections, refractions, and caustics
- Capturing specular reflections is a part of work to be done in the coming year, which essentially, when combined with the diffuse inter-reflection implementation, will give a complete global illumination package for point models

Work-Flow

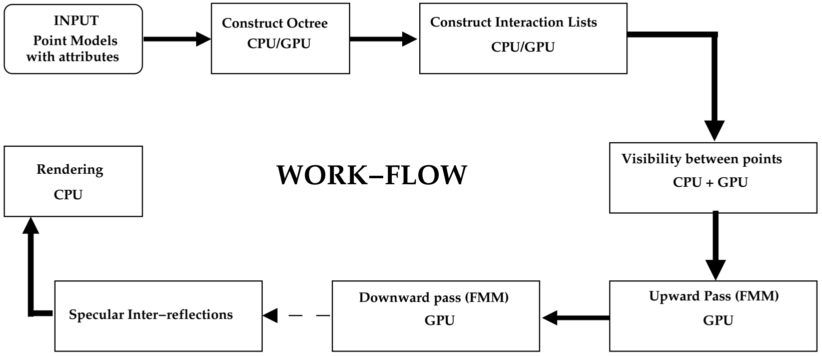


Figure: Work-flow of my thesis

Publications

1. *Visibility Map for Global Illumination in Point Clouds* by R. Goradia, A. Kanakanti, S. Chandran and A. Datta was accepted as an **oral paper** at Proceedings of **ACM SIGGRAPH GRAPHITE**, 5th International Conference on Computer Graphics and Interactive Techniques, 2007. It presents our V-map construction algorithm on the CPU.
2. *GPU-based Hierarchical Computation for View Independent Visibility* by R. Goradia, P. Ajmera and S. Chandran was accepted as an **oral paper at ICVGIP**, Indian Conference on Vision, Graphics and Image Processing, 2008. This paper details our fast, GPU-based V-map construction algorithm.
3. *Fast, parallel, GPU-based construction of space filling curves and octrees* by P. Ajmera, R. Goradia, S. Chandran and S. Aluru was accepted as a **poster at ACM SIGGRAPH SI3D '08**: Proceedings of the 2008 symposium on Interactive 3D graphics and games, 2008. It presents a GPU-based, parallel octree, and first-ever parallel SFC construction algorithm.

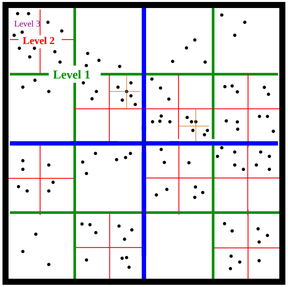
Technical Reports

- *Fast, GPU-based Illumination Maps for Point Models using FMM* by R. Goradia, P. Ajmera and S. Chandran. This work details FMM algorithm for point models to achieve a global illumination solution and the enhanced, fast version of the same on the GPU.
- *GPU-based, fast adaptive octree construction algorithm* by R. Goradia, P. Ajmera, S.Chandran and S. Aluru. It presents two, different, memory-efficient parallel octree construction algorithms on the GPU, which can be combined with the current GPU-based FMM framework.

Outline

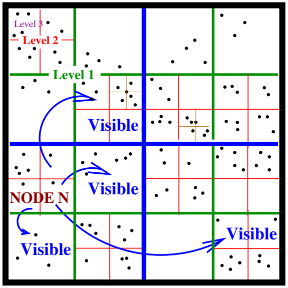
What is a Visibility Map (V-map)?

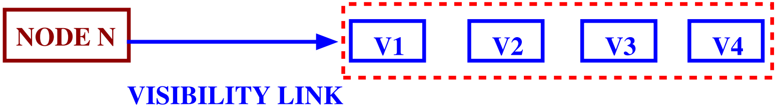
- The *visibility map* for a tree is a collection of visibility links for every node in the tree
- The *visibility link* for any node N is a set L of nodes
- Every point in any node in L is guaranteed to be visible from every point in N



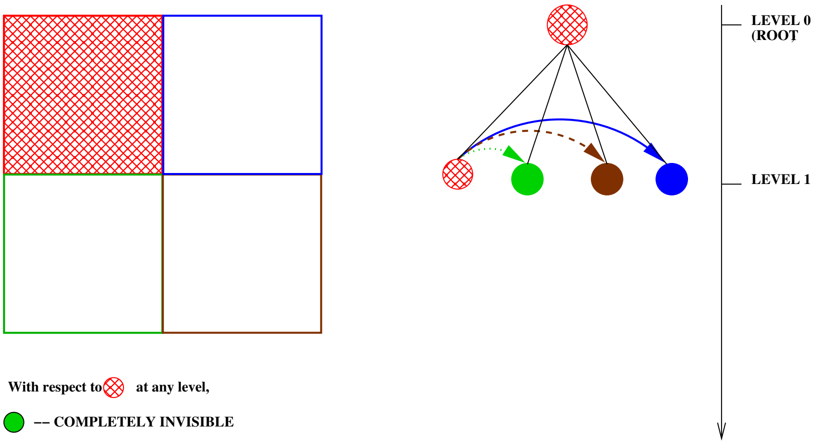
What is a Visibility Map (V-map)?

- The *visibility map* for a tree is a collection of visibility links for every node in the tree
- The *visibility link* for any node N is a set L of nodes
- Every point in any node in L is guaranteed to be visible from every point in N





What is a Visibility Map (V-Map)?



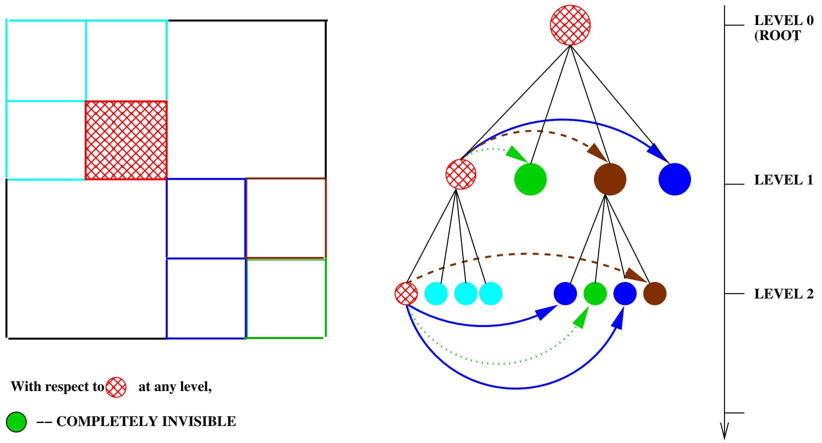
With respect to ▨ at any level,

● –– COMPLETELY INVISIBLE

● –– COMPLETELY VISIBLE

● –– PARTIALLY VISIBLE

What is a Visibility Map (V-Map)?



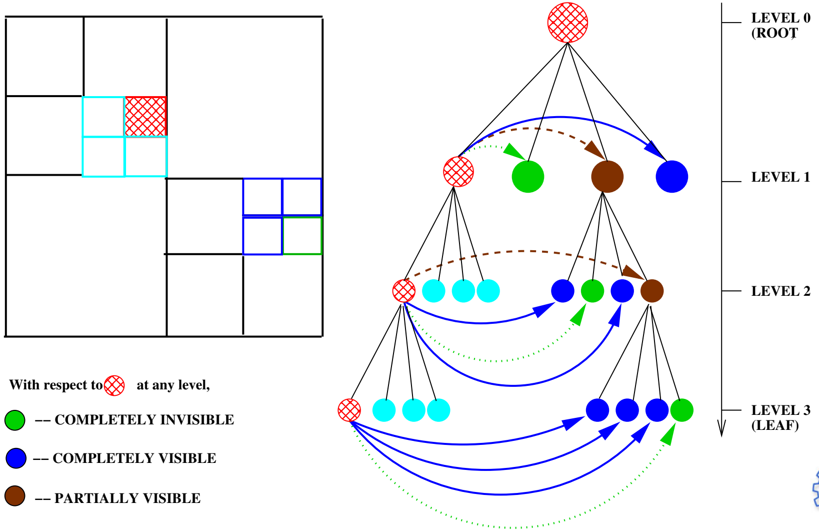
With respect to ▨ at any level,

🟢 -- COMPLETELY INVISIBLE

🔵 -- COMPLETELY VISIBLE

🟤 -- PARTIALLY VISIBLE

What is a Visibility Map (V-Map)?



With respect to ▨ at any level,

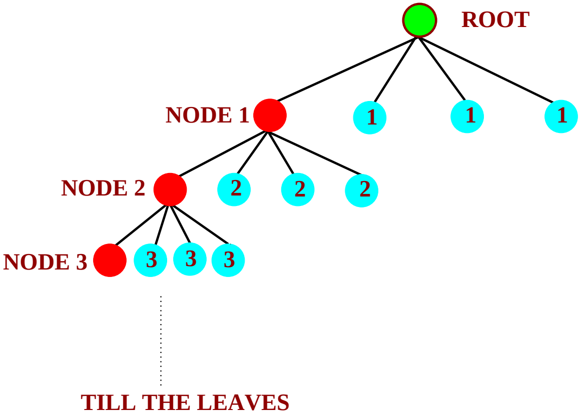
● -- COMPLETELY INVISIBLE

● -- COMPLETELY VISIBLE

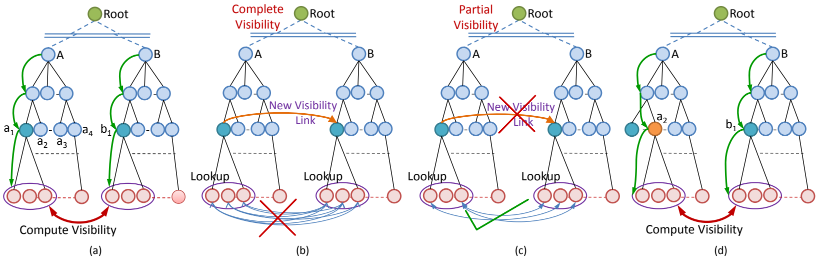
● -- PARTIALLY VISIBLE

Vmap Construction Algorithm

- Initialize the o-IL of every node to be its seven siblings



Vmap Construction Algorithm on CPU



Vmap Construction Algorithm on GPU: Strategy 1

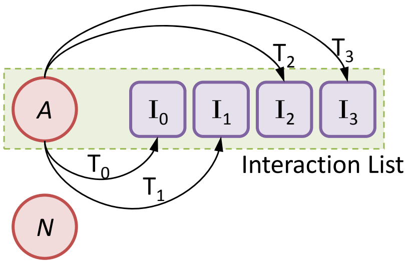


Figure: Multiple Threads Per Node Strategy

- Degree of parallelism here is limited by the size of the interaction list of a node
- **Serious limitation**: Recursion and Dynamic memory allocation

Vmap Construction Algorithm on GPU: Strategy 2

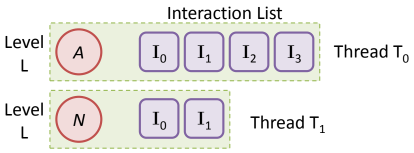


Figure: One Thread Per Node Strategy

- Performance of the algorithm increases as we go down the octree
- Does not utilize the commutative nature of visibility
- **Serious limitation**: Recursion and Dynamic memory allocation

Vmap Construction Algorithm on GPU

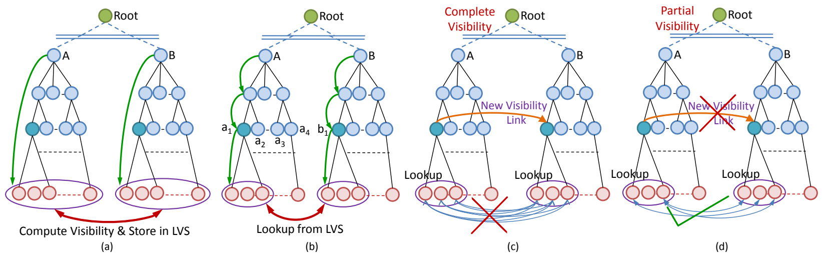
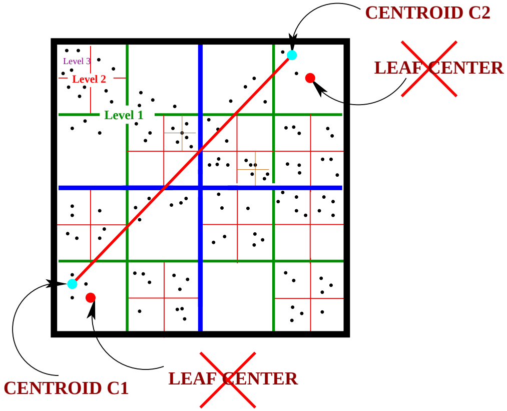


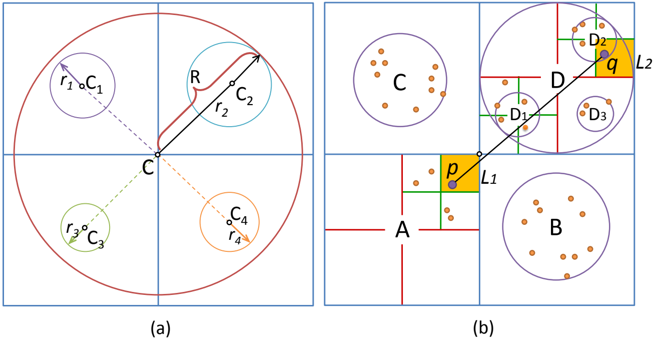
Figure: Multiple Threads per Node-Pair

Leaf-Leaf Visibility Algorithm (CPU)

- Consider centroid and **NOT** leaf center

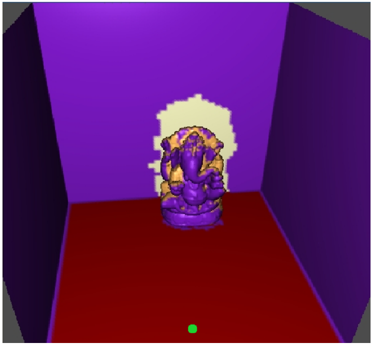


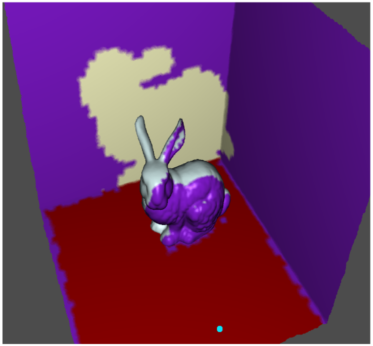
Leaf-Leaf Visibility Algorithm (GPU)



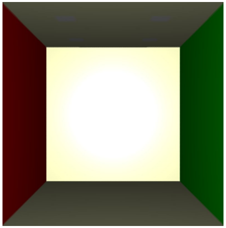
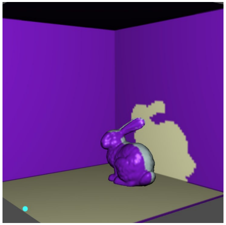
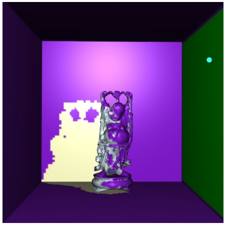
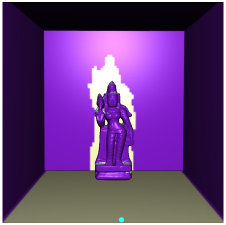
- Recursive Line-Sphere Intersection Test replaces 3D Bresenham's Line algorithm
- Recursion performed using own stack

Results: Visibility Validation Videos (CPU)

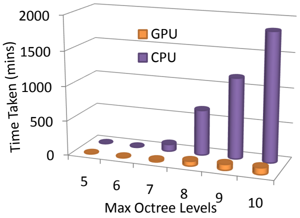




Results: Visibility Validation (GPU)

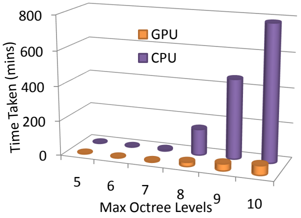





Results: Visibility Timings (GPU)



Bunny in Cornell Room

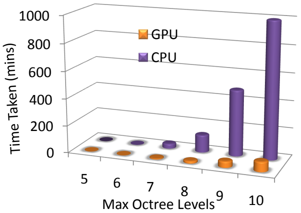
Max Levels	Leaves	GPU (mins)	CPU (mins)	*Speedup*
5	1675	1.87	0.34	*0.18*
6	2181	6.24	2.51	*0.40*
7	7837	21.54	99.31	*4.61*
8	20971	68.25	652.51	*9.56*
9	27514	83.47	1159.04	*13.88*
10	39111	95.98	1839.96	*19.17*



Buddha in Cornell Room

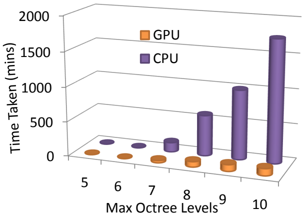
Max Levels	Leaves	GPU (mins)	CPU (mins)	*Speedup*
5	1407	1.35	1.01	*0.75*
6	4612	1.87	1.56	*0.83*
7	9215	4.62	6.79	*1.47*
8	16891	23.89	150.03	*6.28*
9	22117	39.11	456.68	*11.68*
10	28981	54.19	780.23	*14.39*

Results: Visibility Timings (GPU) ..... Continued



Dragon in Cornell Room

Max Levels	Leaves	GPU (mins)	CPU (mins)	*Speedup*
5	965	0.85	0.42	*0.49*
6	1358	1.08	1.05	*0.97*
7	5380	6.74	35.96	*5.34*
8	12587	18.25	128.82	*7.05*
9	21452	45.91	483.84	*10.54*
10	29523	73.83	995.16	*13.48*



Ganesha & Satyavathi in Cornell Room

Max Levels	Leaves	GPU (mins)	CPU (mins)	*Speedup*
5	1232	0.76	0.52	*0.69*
6	2012	1.4	1.05	*0.75*
7	7902	32.43	135.45	*4.18*
8	14500	76.53	587.02	*7.67*
9	28936	94.37	998.26	*10.58*
10	45191	101.12	1747.85	*17.28*

Outline

Phases of FMM

5 phases of FMM algorithm for diffuse GI

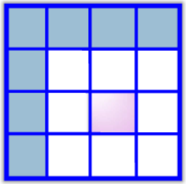
1. Octree Construction **(CPU/GPU)**
2. Generating interaction lists **(CPU/GPU)**
3. Determine visibility between octree nodes **(CPU + GPU)**
4. Upward Pass **(GPU)**
5. Downward Pass and Final Summation **(GPU)**

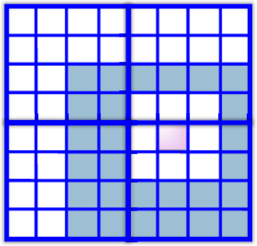
FMM Algorithm: Building Interaction Lists

Building Interaction Lists

Each node has two kind of interaction lists

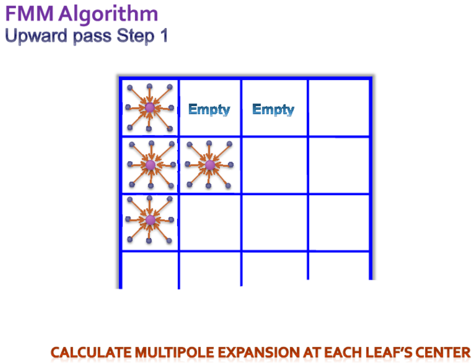
- Far Cell List
- Near Cell List





- No far cell list at level 1 and level 0 since everything is near neighbor of other
- Transfer of energy from near neighbors happens only for leaves

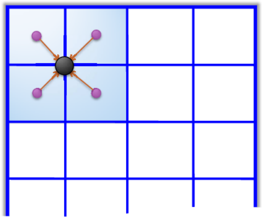
Parallel FMM Algorithm: Upward Pass 1



- One Thread per leaf
- Work of each thread is independent of other
- No need for synchronization
- We need atleast 8192 threads for full-load on GPU

Parallel FMM Algorithm: Upward Pass 2

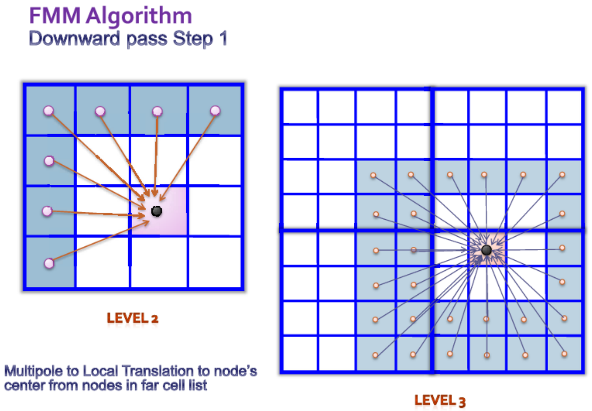
FMM Algorithm
Upward pass Step 2



CALCULATE MULTIPOLE EXPANSION AT EACH PARENT'S CENTER

- Iterate from last level onto the root
- One thread per parent node
- GPU load is low at levels near to root
- Upward Pass takes less than 1% of total FMM time

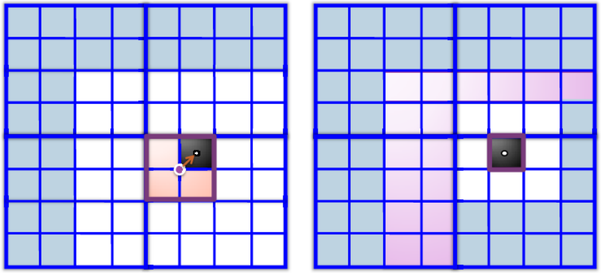
Parallel FMM Algorithm: Downward Pass 1


FMM Algorithm
Downward pass Step 1
LEVEL 2
Multipole to Local Translation to node's center from nodes in far cell list
LEVEL 3

- Iterate from level 2 to last
- Multipole-to-Local translation for every node at current level, **in parallel**
- Most Expensive step of the FMM algorithm

Parallel FMM Algorithm: Downward Pass 2

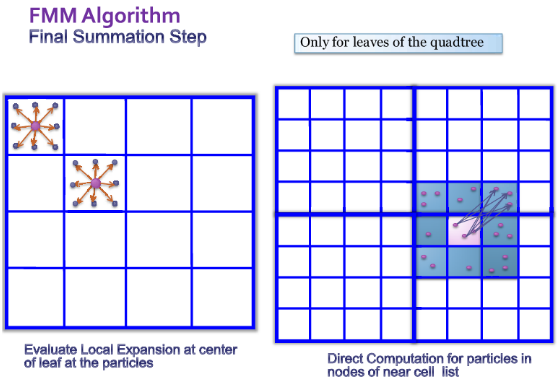
FMM Algorithm
Downward pass Step 2



LOCAL TO LOCAL TRANSLATION FROM PARENT'S CENTER TO CHILDREN'S CENTER

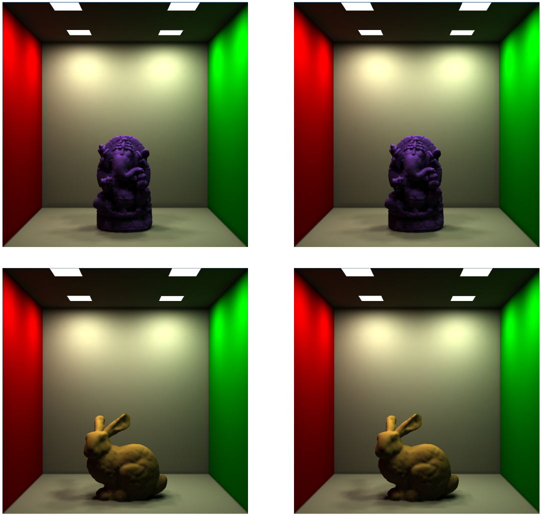
- Iterate from level 2 to last
- Local-to-Local translation for every node at current level, **in parallel**

Parallel FMM Algorithm: Final Summation

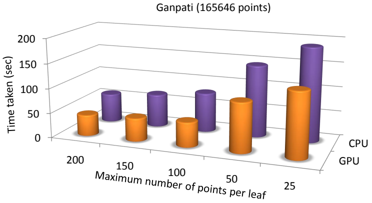

FMM Algorithm
Final Summation Step
Only for leaves of the quadtree
Evaluate Local Expansion at center of leaf at the particles
Direct Computation for particles in nodes of near cell list

- For Local Expansion, iterate from level 2 to last
- If Leaf, compute Local Expansion for leaves at current level, **in parallel**
- For near Neighbor Calculations, iterate from level 2 to last
- If Leaf, each thread performs all near neighbor computations for a particular leaf

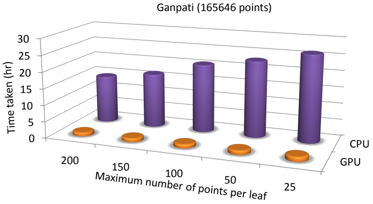
Results: Quality Comparision



Results: Timing Comparision



Number of points per leaf	GPU (sec)	CPU (sec)	GPU Speedup
200	42.3485	58.9931	1.39
150	46.5512	67.2873	1.44
100	49.6921	79.7653	1.61
50	99.4292	145.2349	1.46
25	130.5751	189.4829	1.45



Number of points per leaf	GPU (hr)	CPU (hr)	GPU Speedup
200	1.11	14.54	13.1
150	1.16	16.58	14.3
100	1.21	20.81	17.2
50	1.28	23.15	18.1
25	1.41	26.37	18.7

Outline

Parallel GPU-based Octree Construction

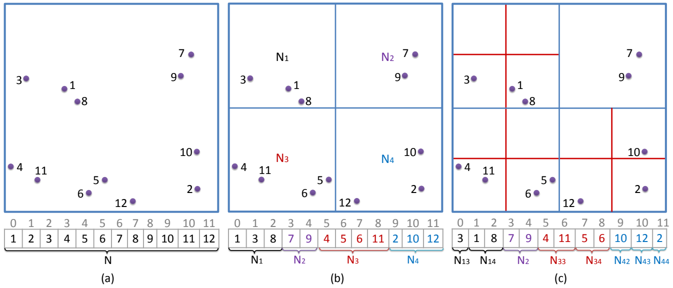


Figure: Spatial Clustering of Points

- Read points in an array P of size n

Parallel GPU-based Octree Construction

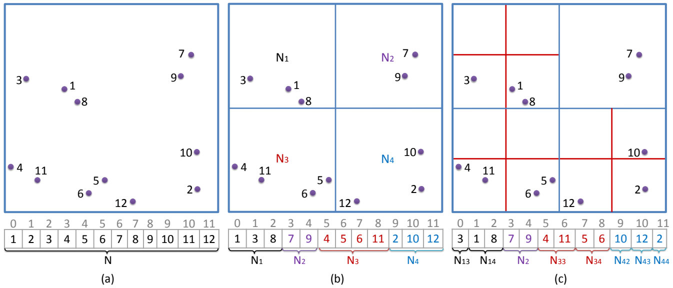


Figure: Spatial Clustering of Points

- Initialize the $root$ node of the octree as containing all points of P
- Set the bounds defining cluster of points belonging to the root as 0 and $n - 1$

Parallel GPU-based Octree Construction

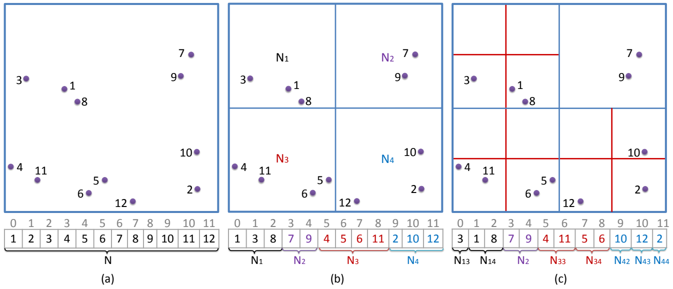


Figure: Spatial Clustering of Points

- Now loop on current step
- Allocate threads equal to the number of partitions. ($Num\_Threads = 1$ initially for the root and then increases as we iterate)

Parallel GPU-based Octree Construction

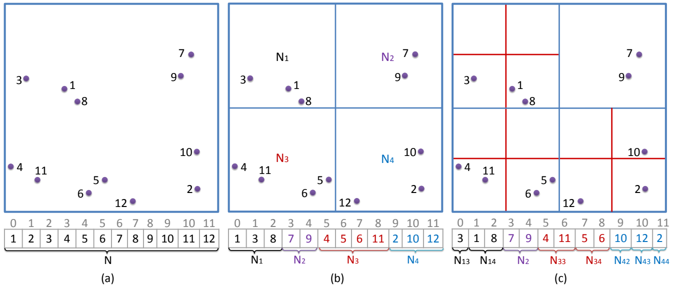


Figure: Spatial Clustering of Points

- For every thread, *in parallel,* do
- STOP the thread if the current partition is a leaf
- ELSE, create 8 new partitions and 8 new octree nodes
- Record the respective partition bounds in the nodes created

Parallel GPU-based Octree Construction

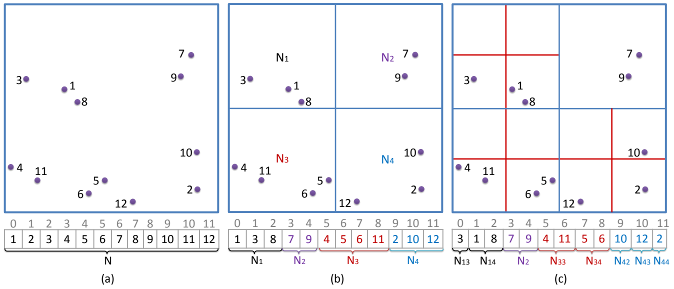
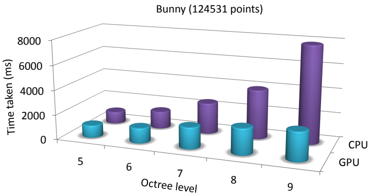


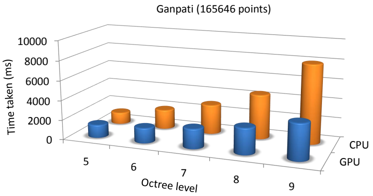
Figure: Spatial Clustering of Points

- STOP looping when every thread encounters a leaf and hence no new partitions are generated

Results: Timing Comparision



Tree level	GPU (ms)	CPU (ms)
5	1001	993
6	1231	1421
7	1742	2521
8	2117	3981
9	2323	7851



Tree level	GPU (ms)	CPU (ms)
5	1321	1200
6	1536	1981
7	2009	2997
8	2654	4521
9	3658	8001

Outline

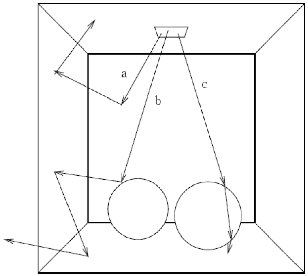
Problem Definition

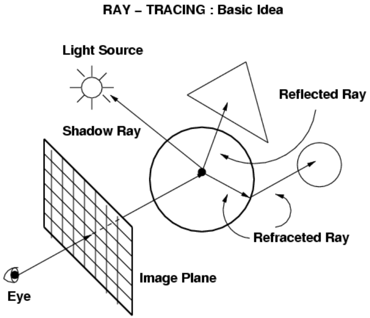
- Computing specular-interreflections, including caustics for point models
- Combine with diffuse inter-reflections
- **Approach**
  - Photon Mapping accounts for both diffuse and specular inter-reflections
  - Analyze it from the perspective of applying it to point models
  - Interactive requirement

Photon Mapping

- A two-pass method
    - First pass builds the photon map
        - Emit photons from light sources into the scene
        - Store them in a photon map on hitting diffuse objects
    - Second pass, the rendering pass
        - Make kNN queries on the photon map
        - Extract information about the radiance values

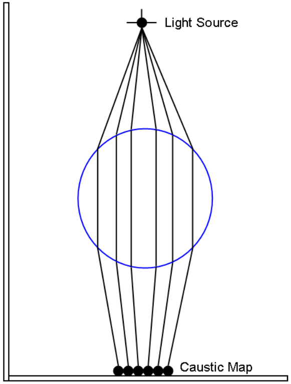
Photon Mapping

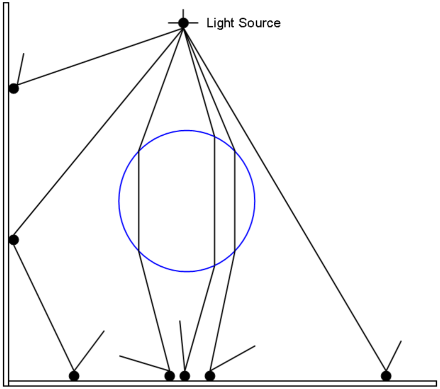




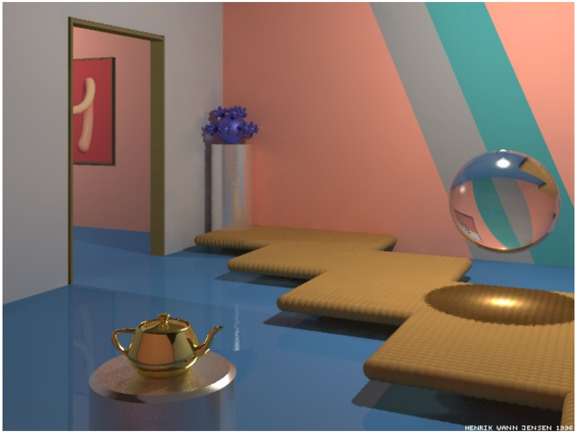
Photon Maps

- *Caustic Photon Maps*: $LS + D$ photons
- *Diffuse Photon Maps*: $LS|D * D$ photons





Example output of Photon Mapping



Splat-Based Ray Tracing(SBRT)

- Deals with intersection of rays and *splats* (disks around points)
- Smoothly varying normal field defined over each splat
- Optimal radii computed for minimum overlap and no holes left between them
- Ray-Splat intersection performed at the time of rendering using Octree Traversal
- Generates reflections, refractions and shadows
- **No Caustics** !

Merge with traditional Photon Mapping to get caustics

Splat-Based Ray Tracing(SBRT)

- Deals with intersection of rays and *splats* (disks around points)
- Smoothly varying normal field defined over each splat
- Optimal radii computed for minimum overlap and no holes left between them
- Ray-Splat intersection performed at the time of rendering using Octree Traversal
- Generates reflections, refractions and shadows
- **No Caustics** !

Merge with traditional Photon Mapping to get caustics

Optimizing Photon Generation

- **Only** Caustic photon generation will take place
- Some of the cost factors can not be improved on. For example,
  - Incoherent Rays
  - Several surface intersections to generate caustics
- However, the number of paths that actually yield caustic photons can be influenced and maximized

Selective Photon Tracing (SPT)

Optimizing Photon Generation

- **Only** Caustic photon generation will take place
- Some of the cost factors can not be improved on. For example,
  - Incoherent Rays
  - Several surface intersections to generate caustics
- However, the number of paths that actually yield caustic photons can be influenced and maximized

Selective Photon Tracing (SPT)

Selective Photon Tracing

- SPT was originally used to generate fast photon maps for dynamically changing object
- Dont consider the temporal domain but adaptively sample path space
    - Send "pilot photons"
    - Detect caustic paths
    - Use periodicity of the Halton sequence to generate similar photons around those paths

Optimizing Photon Tracing and Intersections

- USE SBRT's intersection
- For Photon Traversal,
  - Re-use the Octree generated while performing diffuse GI
  - Use same octree traversal algorithm of SBRT

Fast Photon Retrieval using Optimized kNN-Query Algorithm

- kd-trees are slow for interactive settings
- Serial dependencies are involved while traversing Octree downwards

Low Latency Photon Retrieval Using Block Hashing

Fast Photon Retrieval using Optimized kNN-Query Algorithm

- kd-trees are slow for interactive settings
- Serial dependencies are involved while traversing Octree downwards

Low Latency Photon Retrieval Using Block Hashing

Block Hashing

- Uses the **Locality-Sensitive Hashing (LSH)** function to categorise photons by their positions
- Photon data is broken into fixed-sized blocks
- Accesses memory using burst transfer of cached data blocks
- A kNN query matches the hash bucket to the query point and retrieves neighboring photons

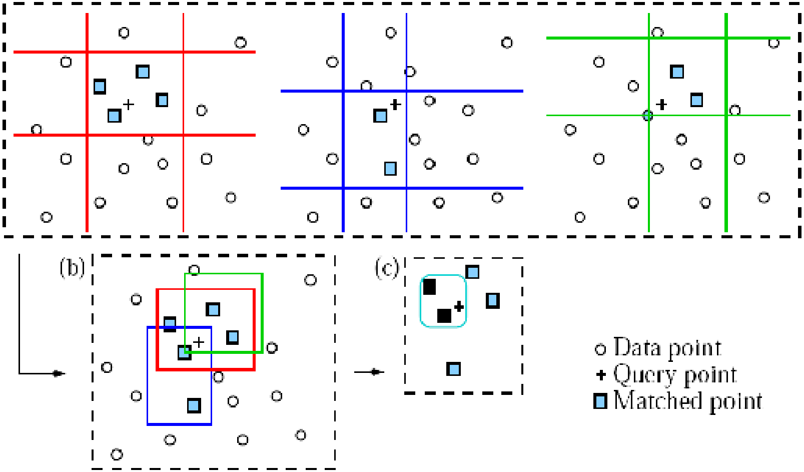
Advantages

- Takes constant time. No serial dependency
- Accessing data in the hash table takes only a single access (using hash-value)

Block Hashing: Phases

- Its, thus, a two-pass algorithm
- A preprocessing phase
  - Organizing the photons into fixed-sized memory blocks
  - Creation of a set of hash tables
  - Inserting photon blocks into the hash tables
- A query phase
  - Hash tables are queried for a set of candidate photons
  - $knearest$ photons are retrived for rendering

Block Hashing : Querying



Merger into a Single System

- Fast caustic photon generation code using *Selective Photon Tracing*
- Ray-splat intersection code from *SBRT* for caustic photon traversing and intersection
- Use *SBRT* for ray tracing and getting the reflections and refractions
- Usage of caustic photon maps and fast kNN query algorithm using *Block Hashing* provides us with efficient caustics while rendering

Combine all these algorithms in a single system

Merger into a Single System

- Fast caustic photon generation code using *Selective Photon Tracing*
- Ray-splat intersection code from *SBRT* for caustic photon traversing and intersection
- Use *SBRT* for ray tracing and getting the reflections and refractions
- Usage of caustic photon maps and fast kNN query algorithm using *Block Hashing* provides us with efficient caustics while rendering

Combine all these algorithms in a single system

Outline

Conclusion and Future Work

- The lack of surface information in point models creates difficulties in operations like generating global illumination effects and computing point-pair visibility
- Point-to-Point Visibility is arguably one of the most difficult problems in rendering since the interaction between two primitives depends on the rest of the scene
- One way to reduce the difficulty is to consider clustering of regions such that their mutual visibility is resolved at a group level (V-Map)
- Visibility Map data structure we propose enables efficient answer to common rendering queries
- By viewing this visibility map as a 'preprocessing' step, photo-realistic global illumination rendering of complex point-based models have been shown

Conclusion and Future Work

- We exploited the parallel computing power of GPUs for implementation of the FMM as well as the point-pair visibility determination algorithm using V-Maps
- GPU-based Parallel octree construction algorithm was presented which can be combined with the parallel FMM framework
- Further, we saw how various algorithms from the literature were combined under a single domain to get us a time-efficient system designed to generate the desired specular effects for point models
- **We now aim to implement these algorithms, merge them together and get the specular effects solution for point models**

Conclusion and Future Work

- We, thus, will have a two-pass global illumination solver for point-based scenes consisting of both diffuse and specular models
    - First pass will calculate the diffuse illumination maps
    - Second pass for specular effects
- Finally, the scene will be rendered using *splat-based ray-tracing* technique
- **However, a question remains that since we are parting the diffuse and specular effect calculations for the scene, how would we handle specular objects (and their effects on diffuse objects) while calculating only diffuse global illumination in the first pass of the global illumination solver**
- *This important issue needs to be investigated thoroughly*

Thank you for your time !

Questions ?