Fast, GPU-based Illumination Maps For Point Models using FMM

Rhushabh Goradia, Prekshu Ajmera and Sharat Chandran

Indian Institute of Technology Bombay

Abstract

Point-based methods have gained significant interest due to their simplicity. The lack of connectivity touted as a plus, however, creates difficulties in generating global illumination effects. We are interested in looking at inter-reflections in complex scenes consisting of several models, the data for which are available as hard to segment aggregated point-based models.
In this paper we use the Fast Multipole Method (FMM) which has a natural point based basis, and the light transport kernel for inter-reflection to compute a description – illumination maps – of the diffuse illumination. These illumination maps may be subsequently rendered using methods in the literature such as the one in [WS05]. We use Graphics Processing Units with CUDA programming enivronment to achieve multi-fold speed-ups.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

This paper is about capturing interreflection effects of a set of objects when the input is available as point samples. We use the technique of the Fast Multipole Method which also starts with points as primitives.

1.1. Point Samples

Points as primitives have come to increasingly challenge polygons for complex models; as soon as triangles get smaller than individual pixels, the raison d'etre of traditional rendering can be questioned. Simultaneously, modern 3D digital photography and 3D scanning systems [LPC*00] acquire both geometry and appearance of complex, real-world objects in terms of (humongous) points. More important, however, is the considerable freedom points enjoy. The independence of connectivity and topology enable filtering operations, for instance, without having to worry about preserving topology or connectivity [PKKG03, OBA*03, PZvBG00].

1.2. Global Illumination

Global illumination – the simulation of the physical process of light transport – is a memory intensive and compute intensive operation. This problem has been considered for several years with interesting methods like statistical photon tracing, directional radiance maps, and wavelets based hierarchical radiosity. Traditionally all these methods *assume a surface* representation for the propagation of indirect lighting. Surfaces are either explicitly given as triangles, or implicitly computable. The lack of any sort of connectivity information in point-based modeling (PBM) systems now *hurt* photo-realistic rendering. This becomes especially true when it is not possible to correctly segment points obtained from an aggregation of objects (see Figure 1) to stitch together a surface.

Recent work [WS05] suggests one way to handle this problem — ray tracing. However, as both rays and points are singular primitives, this requires one to trace thick rays [Ama84, SJ00]. Alternatively, points are seen as covering a finite area by expanding them to ellipses [RL00], or filtering them with an implicit function [AA03, OBA*03].

Our view is that these methods would work *even better* if fast pre-computation of diffuse illumination could be per-

Figure 1:
Grottoes, such as the ones from China and India form a treasure for mankind. If data from the ceiling and the statues are available as point samples, can we capture the interreflections?

formed, much the way photon tracing is done for triangulated models before rendering.

1.3. Fast computation with FMM

Computational science and engineering is replete with problems which require the evaluation of pairwise interactions in a large collection of particles. Direct evaluation of such interactions results in $O(N^2)$ complexity which places practical limits on the size of problems which can be considered. The first numerically-defensible algorithm [DS00] that succeeded in reducing the N-body complexity to $O(N)$ was the Greengard-Rokhlin Fast Multipole Method (FMM) [GR87]. The FMM, in a broad sense, enables the product of restricted dense matrices with a vector to be evaluated in $O(N)$ or $O(N \log N)$ operations, when direct multiplication requires $O(N^2)$ operations.

Global illumination problem requires the computation of pairwise interactions among each of the surface elements (points) in the given data (usually of order $> 10^6$) and thus naturally fits in the FMM framework.

Besides being very efficient ($O(N)$ algorithm) and applicable to a wide range of problem domains, the FMM is also highly parallel in structure. Thus implementing it on a parallel, high performance multi-processor cluster will further speedup the computation of diffuse illumination for our input point sampled scene. Our interest lies in a design of a parallel FMM algorithm that uses static decomposition, does not require any explicit dynamic load balancing and is rigorously analyzable. We use latest Nvidia's G80/G92 architechture GPUs with CUDA [**?**] as the programming environment.

1.4. Contributions

Can the point-based framework of the FMM (albeit without visibility) be coupled with the input point models to store the diffuse illumination? This paper answers this question in the affirmative. We store the precomputation in a data structure called Illumination Maps which are conceptually like photon maps except that we do not employ statistical photon tracing. The challenges we solve in the process are

- Earlier [ala04], we presented the mathematical apparatus required to apply the *linear-time adaptive* FMM algorithm to diffuse objects given as triangles. Five mathematical results with respect to the core interreflection kernel under full visibility are now available. We extend this to blend the point based nature of FMM with input available as PBMs instead of triangles. For storing illumination maps, this is sufficient. For more complete rendering (purely based on the FMM technique) we require the BRDF to be available as a low rank matrix. This coupled with a directional discretization of radiance [Wal05] should be employed for pure FMM-based rendering of non-diffuse objects.
- We exploit the inherent parallelism of this method to implement it on the data parallel architecture of the GPU to achieve multifold speedups. Further, the same parallel implementation on the GPU, designed for point models, can also be used for triangular models.

The rest of the paper is organized as follows. First, we gave the basic background of FMM followed by our mathematical results for factorizing the interreflection kernel. Section 2.4 provides the crucial extension needed when points are given as input. Our visibility requires us to provide a stripped-down FMM algorithm which we give in Section 2.5. We follow this with our GPU-based parallel FMM algorithm using CUDA in section 3. Pictures of our illumination maps appear in Section 4 followed by concluding remarks.

2. FMM for Global Illumination

The Fast Multipole Method [GR87] is concerned with evaluating the effect of a "set of sources" $\mathbb{Y}$, on a set of "evaluation points" $\mathbb{X}$. More formally, given

$$\mathbb{X} = \{x_1, x_2, \ldots, x_M\}, \quad x_i \in \mathbb{R}^3, \quad i = 1, \ldots, M, \quad (1)$$

$$\mathbb{Y} = \{y_1, y_2, \ldots, y_N\}, \quad y_j \in \mathbb{R}^3, \quad j = 1, \ldots, N \quad (2)$$

we wish to evaluate the sum

$$f(x_i) = \sum_{j=1}^{N} \phi(x_i, y_j), \quad i = 1, \ldots, M \quad (3)$$

The function ϕ which describes the interaction between two particles is called the "kernel" of the system. The function f essentially sums up the contribution from each of the sources y_j. Assuming that the evaluation of the kernel ϕ can be done in constant time, evaluation of f at each of the N evaluation points requires N operations. The total complexity of this operation will therefore be $O(NM)$. The FMM attempts to reduce this seemingly irreducible complexity to $O(N\log N+M)$ or even $O(N+M)$. The three main insights that make this possible are (a) factorization of the kernel, (b) the observation that many application domains do not require that the function f be calculated at very high accuracy, (c) FMM follows a **hierarchical structure** (*Octrees*).

2.1. Interreflection in the FMM context

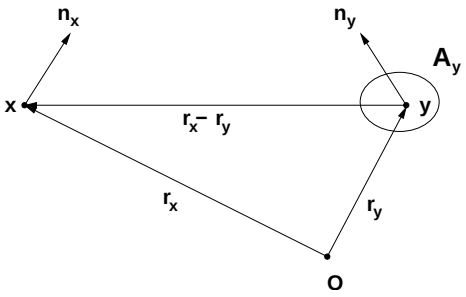


Figure 2: Geometry and notations used in this paper.

Figure 2 shows how a point x receives irradiance from a small area around y. The nature of this interaction is quadratic for all points as in Equation 3. Further, the kernel of the geometric interaction (assuming full visibility) can be written as:

$$I(x)=\int_{A_y}\frac{[\vec{ny}.(\vec{rx}-\vec{ry})][\vec{nx}.(\vec{ry}-\vec{rx})]}{\pi|\vec{ry}-\vec{rx}|^4}\,dA_y \tag{4}$$

Notice that the interaction written in this form is coupled in nature. The theory of the FMM, in general, requires factorization and translation theorems for the type of kernel under consideration. Simply stating, these results are based on the position and orientation of the source and receivers. These results are given in brief in Section 2.2 and the proofs appear in [ala04, ala03]. The nature of light transport is even more complicated than this, but Equation 4 is sufficient to capture the diffuse illumination maps.

2.2. Multipole Expansion

If we denote the spherical coordinates of $\vec{rx}$ by (r_x,θ_x,ϕ_x), then our first result makes use of [Hau97] to write (for $r_y<r_x$),

$$\frac{1}{|\vec{ry}-\vec{rx}|^4}=\sum_{n=0}^{\infty}\sum_{j=0}^{[n/2]}\sum_{m=-n+2j}^{n-2j}\pi_n^j\left\{\frac{1}{r_x^{n+4}}Y_{n-2j}^m(\theta_x,\phi_x)\right\}\left\{r_y^n\overline{Y_{n-2j}^m(\theta_y,\phi_y)}\right\} \tag{5}$$

where

$$e_n^j=4\frac{(n-j+1)!(j+1/2)!}{(n-j+1/2)!j!}$$

and Y_n^m are the normalized spherical harmonics. Substituting (5) in (4) and rearranging terms, we get the *multipole expansion* in Equation 8 as

$$I(x) = \sum_{n=0}^{\infty}\sum_{j=0}^{[n/2]}\sum_{m=-n+2j}^{n-2j}e_n^jR_{nj}^m(x)\otimes M_{nj}^m(A_y) \tag{6}$$

$$R_{nj}^m(x) = \frac{\rho(x)}{r_x^{n+4}}Y_{n-2j}^m(\theta_x,\phi_x)\mathbf{R}M(x) \tag{7}$$

$$M_{nj}^m(A_y) = \int_{A_y}r_y^n\overline{Y_{n-2j}^m(\theta_y,\phi_y)}\mathbf{S}M(y)\,dA_y \tag{8}$$

Here, **R**M(x) and **S**M(y) stands for the receiver and the source matrices respectively. The intuition for this step is shown in Figure 3. For practical implementation, the summation to infinity is truncated to (in our case) two terms. We have theoretically and experimentally verified [ala04] that the error incurred is very small.

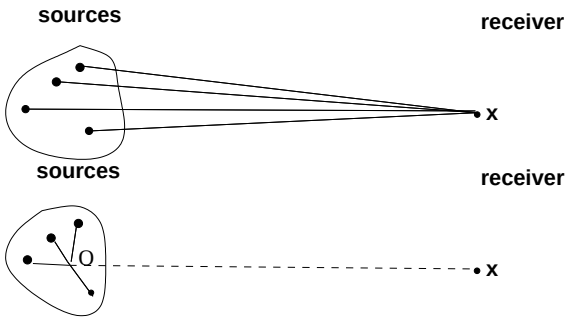


Figure 3: By associating a constant number of coefficients at center O, we can calculate the irradiance received by x from a number of differential emitters. The value of the coefficients depends upon the location of these emitters, and the recipient has to be sufficiently far.

Since the FMM algorithm is hierarchical, we need a way to collect irradiance, as shown in Figure 4.

2.3. Local Expansion

Equation 6 may be viewed as an irradiance gather process "outside" the sources. We need a similar expression on how irradiance collected at a center is distributed to receivers. For $r_x<r_y$, we derive [ala03] the so-called *local expansions* in terms of the coefficients L_{nj}^m. Our intuition behind this formulation is explained in Figure 5.

Similar to the multipole coefficients, the *local coefficients* L_{nj}^m are also additive, and can be translated to a different coordinate system. We illustrate this in Figure 6.

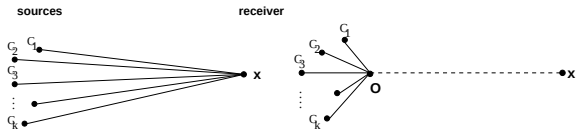


Figure 4: Multipole coefficients are additive and can be translated to a different coordinate system. This enables a hierarchical approach by considering the effect of several clusters. For each cluster $C_1, C_2, C_3, \ldots C_k$, the *multipole coefficients* $M_{nj}^m(A_y)$ are first accumulated and then "translated" to get the cumulative effect of the entire set of clusters.

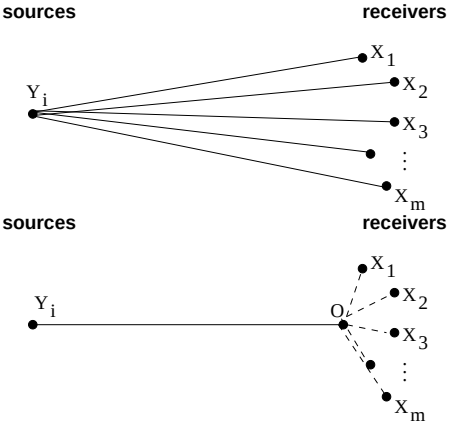


Figure 5: The irradiance stored at a virtual point O in the form of a constant number of coefficients can be dissemenated to different receivers. This is valid only if the receiver points are "close by."

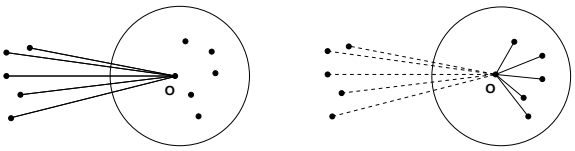


Figure 6: Local coefficients are additive. On the left, we first collect the cumulative local coefficient of several clusters from the local coefficients of each cluster and accumulate it in the center O. We then disseminate it to the recipients.

Finally, a very important result is ilustrated in Figure 7.

2.4. Assigning Weights to Points

The equations in the previous section assume that we are in a position to integrate over a surface area. In our earlier work, we had assumed triangles as input and we performed Gaussian quadrature to calculate the integral exactly. For PBMs, we do not have any surface information; we therefore approximate this integration. Weights are assigned to each point and signify the contribution of the point to the reconstruction of the surface. This is a local property based on the normal available at points. As the number of points increase, the integration is computed more accurately.

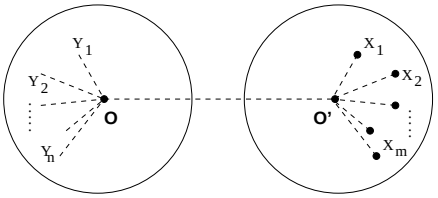


Figure 7: A crucial part of FMM is the conversion of multipole coefficients at a given center into local coefficients at another center. The multipole to local translation converts the multipole coefficients of a set of N source points into local coefficients for a set of M receiver points.

In summary, we can define the multipole coefficients (and similarly local coefficients) for a point y as

$$M_{nj}^m(y) = w(y) r_y^n \overline{Y_{n-2j}^m(\theta_y, \phi_y)} \mathbf{S} M(y) \qquad (9)$$

We thus replace the interaction between surfaces and points (in Equation 8) as between points only. This interaction is termed as a *particle interaction*.

2.5. The FMM Algorithm

A brief version of the algorithm is given here for the sake of completeness.

1. **Setup**: We start with the given input point model. All points are arranged in an adaptive octree such that no leaf node contains more than $s = O(1)$ points. With each node, we associate two set of disjoint nodes:
   - *near neighbors* of a node b are nodes that share a common boundary point of the node. Points in these nodes do not satisfy the distance constraint in (Equation 6).
   - *interaction list* of a node b are the children of the near neighbors of the parent of b — children who are *not* near neighbors of b itself. When occlusion is present in the scene, the interaction list is modified as in [GAC08].
2. **Upward Pass**: For each leaf node in the octree, we calculate the multipole coefficients of all points contained in the node about its center. Then, for each level (starting from the penultimate level) we calculate the multipole coefficients of each node at that level by translating and accumulating the multipole coefficients of its children.
3. **Downward Pass**: For each level (starting from the second), the local coefficients at each node b are calculated by converting the multipole coefficients of boxes in the

interaction list of b into local coefficients about b's center using the multipole to local translation algorithm (Figure 7). Additionally, the local expansion coefficients obtained from the individual points contained in the local interaction list are aggregated.

4. **Evaluation**: For each leaf b in the octree, for each evaluation point $x \in b$, the local expansion about the center of b is evaluated at x.

We iterate over these steps till sufficient convergence is reached. The evaluation points are the same points that represent the input point model.

3. FMM on GPU

Besides being very efficient ($O(N)$ algorithm) and applicable to a wide range of problem domains, the FMM is also highly parallel in structure. Thus implementing it on a parallel, high performance multi-processor cluster will further speedup the computation of diffuse illumination for our input point sampled scene.

FMM algorithm used for GI consists of the following five phases:

1. Octree Construction
2. Generating interaction lists
3. Determine visibility between octree nodes
4. Upward Pass
5. Downward Pass and Final Summation

Our parallel FMM algorithm specifically solves the last two phases (Upward pass, Downward pass and Final summation stage) on the GPU. These phases are the ones which take more than 97% of the run time (not taking visibility phase into account). Hence we *first* implemented these two stages on the GPU while the Octree Construction and Interaction List Construction stages were performed on the CPU. We assume, as a part of pre-processing step, that we have been given an octree constructed for the input 3D model along with the interactions lists for each of the octree nodes (containing only visible nodes). The octree can be constructed on the CPU or on the GPU , while the interaction lists construction happens on the CPU. These two phases will eventually be implemented on GPU and combined with the rest of the algorithm. Visibility between octree nodes is determined by using a CPU-GPU combo algorithm presented in [GAC08]. Our FMM algorithm bears some similartiy to the one presented in [GD07]. But they solve for Laplace kernel, whereas our radiosity kernel involves much more complications.

INPUT: A 3D model with its defined octree and visible interaction lists.
OUTPUT: A Diffuse Global Illumination solution for the given model.

Our input octree is a long one dimensional array with each level of octree stored one after the other(starting from the *root*). The parent-child relationship is established using the array indices. We also define four one dimensional arrays, each corresponding to one of the interaction list's type (far, near, multipole, local). The size of each of these arrays is the sum total of the number of nodes in the interaction lists of every node. The relationship between each node and each of its interaction lists is defined by storing in it the start and the end indices of each of its interaction list in the four global interaction list arrays. A 3D input point model is stored as a single point array with its necessary attributes (co-ordinates, normal, diffuse surface color, emmissivity, gaussian weights). Incase of triangular models, they are converted to points using gaussian quadrature weights theory [ala03].

3.0.1. Step 1: Generating Multipole Expansion Co-efficients for the Leaves (Upward Pass)

We need to calculate, in parallel, for each leaf in the octree, the multipole, or S expansion of all particles (sources) contained in the node about the center of the node. The expansions from all particles (sources) in the node are consolidated in a single expansion by summing the coefficients corresponding to each particle (source). We adopt a *one thread per node* strategy (and the same strategy for other passes of FMM). In this case one thread performs expansion for each of the sources in the leaf and consolidates these expansions. So one thread produces full multipole expansion for the entire leaf. The advantage of this approach is that the work of each thread is completely independent and so there is no need for shared memory. This perfectly fits the situation when each leaf may have different number of sources, as the thread that finishes work for a given leaf simply takes care of another leaf, without waiting or need for synchronization with other threads.

1. For every level of octree, starting from the last level of octree, upto the root do
   a. Allocate threads equal to number of nodes at the current level
   b. For every thread, *in parallel*, Do
   c. If current node is a leaf Then
      i. Calculate the multipole expansion of all particles (sources) contained in the current leaf about the center of that leaf.
      ii. Consolidate each of these expansions in a single expansion at the current leaf's center.

3.0.2. Step 2: Generating Multipole Expansion Coefficients for the Internal Nodes (Upward pass)

We need to calculate, in parallel, for each level $l = l_{max} - 1, \ldots 2$, for each node b at that level, the multipole, or S expansion coefficients $\mathbf{M}(b)$ due to all particles in that node by translating and aggregating the multipole expansion coefficients of all its children.

1. For every level of octree, starting from the second last level, upto the root do

a. Allocate threads equal to number of nodes at the current level
b. For every thread, *in parallel*, Do
   i. If current node is a non-leaf node Then
   ii. For all childeren of current node, Do
   iii. Translate S coefficient corresponding to the child to the center of the current node
c. Synchronize the threads
d. Sum up all the coefficients at center of current node

Note that the upward pass is a very cheap step of the FMM and normally takes not more than 1% of the total time. This also diminishes the value of putting substantial resources and effort in achieving high speedups for this step.

3.1. Downward Pass

We repeat the following steps for each level of the octree, starting from level 2 to the maximum level l_{max}. Downward pass and the final summation phases are combined into a single phase.

3.1.1. Step 1: Multipole to Local Translations

For each node, in parallel, translate and aggregate the multipole, or S expansion coefficients of every node in the far cell interaction list of the current node into local, or R expansion coefficients about the current node's center.

1. Allocate threads equal to number of nodes at the current level
2. For every thread, *in parallel*, Do
   a. For each node A in the far cell list of current node Do
      i. Translate the multipole expansion coefficients of A into local, or R expansion coefficients about the center of current node.
      ii. Aggregate each of these expansions in a single expansion at the current node's center.

3.1.2. Step 2: Local List Translations

For every node, in parallel, in addition to converting the multipole expansion coefficients of all nodes in the interaction list into local expansion coefficients at the node's center, the local expansion coefficients obtained from the individual particles contained in the local interaction list are also aggregated.

1. Allocate threads equal to number of nodes at the current level
2. For every thread, *in parallel*, Do
   a. For each node n in the local list of current node Do
      i. Obtain the local expansion coefficients obtained from the individual particles contained in n about the center of current node.
      ii. Aggregate each of these expansions in a single expansion at the current node's center and add up to its existing local expansion coefficients.

3.1.3. Step 3: Local to Local Translations

In addition to multipole-to-local and local-list translations, we further need to calculate, in parallel, for each node b at current level, the local, or R expansion coefficients about its center by translating and aggregating the local expansion coefficients from its parent.

1. Allocate threads equal to number of nodes at the current level
2. For every thread, *in parallel*, Do
   a. Obtain the local expansion coefficients from its parent node about the center of current node.
   b. Add up to the existing local expansion coefficients about current node's center.

This step is very similar to the step 2 of upward pass. For parallelization of this step, the *one thread per node* strategy is used.

3.1.4. Step 4: Evaluate Local Expansion at Points

Evaluate, in parallel, the local expansions at individual points in each of the leaves, from the corresponding leaf's center.

1. Allocate threads equal to number of nodes at the current level
2. For every thread, *in parallel*, Do
   a. If current node is a leaf Then
      i. Obtain the radiosity values at individual points in the current leaf, from the local expansion coefficients at current leaf's center.

This step is very similar to the multipole expansion generator discussed above in step 1 of the upward pass. For parallelization of this step, the *one thread per node* strategy is used. The performance of this step is approximately the same as of the multipole expansion generator.

3.1.5. Step 5: Near Cell List Translations

For every node in parallel, evaluate the near neighbor interactions (if current node is a leaf) between the points in the current node and every point in each of the nodes in its near cell interaction list. This, and the remaining steps, are a part of the final summation phase.

1. Allocate threads equal to number of nodes at the current level
2. For every thread, *in parallel*, Do
   a. If current node is a leaf Then
      i. For each node n in the near cell list of current node Do

A. For all points in current leaf
B. For all points in n
C. Evaluate the radiosity interaction directly
D. Add up the evaluated value to the exisiting radiosity values of the points

3.1.6. Step 6: Multipole List Translations

For every node, in parallel, in addition to evaluating the near neighbors and local expansion coefficients at each particle, we also evaluate the multipole expansion coefficients of all nodes in the multipole interaction list.

1. Allocate threads equal to number of nodes at the current level
2. For every thread, *in parallel*, Do
   a. If current node is a leaf Then
      i. For each node n in the multipole list of current node Do
         A. Translate the multipole expansion coefficients of n from its center to individual points of current leaf
         B. Add up the evaluated value to the exisiting radiosity values of the points

Performance improves with the size of the problem and, respectively, as the maximum level of the octree increases (for $level_{max} = 8$ the time ratio reached 20). Each point has 3 primary colors associated with it viz. Red, Green and Blue. Hence, we run all the above steps thrice, corresponding to each color. Further, we converge to the final Global Illumination solution by iterating all the above steps (for all colors) three times (Empirical evidences prove that the solution converges to a good extent in 3 iterations).

4. Results

The CUDA based parallel FMM algorithm, implemented on G80 NVIDIA GPU, was tested on several point models. In this section we provide qualitative validation and quantitative results. Note that all input such as the models in the room, the light source, and the walls of the Cornell room are given as points.

Fig. 8 shows results of FMM based global illumination algorithm applied to point models. Effects of color bleeding and soft shadows are clearly visible. It also works well even in the case of aggregated input models (e.g., point models of both Ganesha and Satyavathi placed in a point model of a Cornell room). Note that the input is a single, large, *mixed* point data set consisting of Ganesha, Satyavathi, and the Cornell room. These models were not taken as separate entities nor were they segmented into different objects during the whole process.

Quality Comparisons

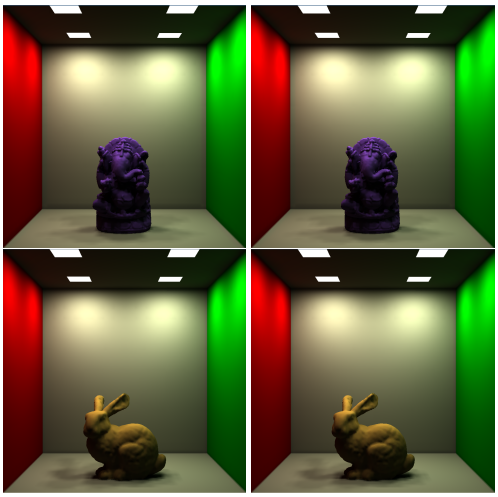


Figure 9: Left (2 pics): A Cornell Room with the Ganesha's point model on CPU and corresponding GPU result. Right (2 pics): A Cornell Room with the Bunny's point model on CPU and corresponding GPU result. Both the results assume 50 points per leaf.

To compare CPU and GPU implementations we use 3-d point models of bunny and Ganesha in a Cornell room each having four light sources on the ceiling. As we can see in Fig. 9 the *CPU-GPU results look identical*. Color bleeding and soft shadows are also clearly visible. We converge after 3 iterations.

Timing Comparisons

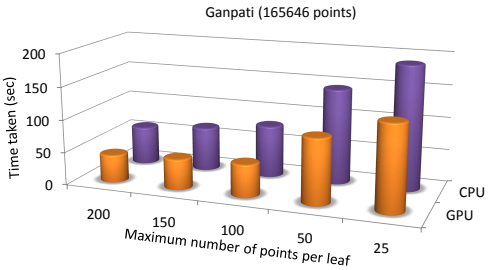
The timing calculations are done on a machine having a dual core AMD Opteron 2210 processor with 2 Gbs of RAM, NVIDIA GeForce 8800 GTS with 320 Mbs of memory and Fedora Core 7 (x86_64) installed on it. The total time taken by the upward and downward passes of the FMM algorithm for all 3 iterations and all 3 colors RGB is shown in the results below. The time taken by each iteration is approximately same.

Thus, we see that the GPU outperforms the CPU by factors of $13-20$ in the downward pass of the FMM algorithm. We also see that the upward pass of the FMM algorithm consumes less than 1% of the time taken by the downward pass. Thus, the speedup achieved in the upward pass does not play an important role in the overall FMM speedup. *The overall speedup achieved is the speedup achieved in the downward pass.*

5. Conclusion

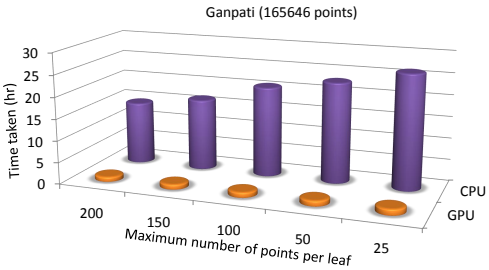
The FMM method is elegant because it trades off error with quality in a disciplined quantitative way. In this paper, the kernel of the energy balance in the rendering equation has

Figure 8: Point models rendered with diffuse global illumination effects of color bleeding and soft shadows. Pair-wise visibility information is essential in such cases. Note that the Cornell room as well as the models in it are input as point models.



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

Figure 10: FMM Upward Pass: Ganpati with 165646 points



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

Figure 11: FMM Downward Pass: Ganpati with 165646 points

been made conformant to the FMM by deriving the near and far field expansions. The illumination problem over surfaces is reduced to a solution over points enabling point based rendering. We have also extended this to latest NVidia's GPU and achieved multi-fold speed-ups. Photo-realistic global illumination for point models have been shown.

References

[AA03] ADAMSON A., ALEXA M.: Ray tracing point set surfaces. In *Proceedings of Shape Modeling International 2003* (2003).

[ala03] *Removed for purposes of Review*. Master's thesis, 2003.

[ala04] Removed for purposes of review.

[Ama84] AMANATIDES J.: Ray tracing with cones. In *Computer Graphics (SIGGRAPH '84 Proceedings)* (1984), Christiansen H., (Ed.), vol. 18, pp. 129–135.

[DS00] DONGARRA J., SULLIVAN F.: The top ten algorithms. *Computing in Science and Engineering 2* (2000), 22–23.

[GAC08] GORADIA R., AJMERA P., CHANDRAN S.: Gpu-based hierarchical computation for view independent visibility. *Accepted at ICVGIP, Indian Conference on Vision, Graphics and Image Processing* (2008).

[GD07] GUMEROV N. A., DURAISWAMI R.: Fast multipole methods on graphics processors. *Astro GPU* (2007).

[GR87] GREENGARD L., ROKHLIN V.: A fast algorithm for particle simulations. *Journal of Computational Physics 73* (1987), 325–348.

[Hau97] HAUNSNER A.: Multipole expansion of the light vector. *IEEE Transactions on Visualization and Computer Graphics 3*, 1 (Jan-Mar 1997), 12–22.

[LPC*00] LEVOY M., PULLI K., CURLESS B., RUSINKIEWICZ S., KOLLER D., PEREIRA L., GINZTON M., ANDERSON S., DAVIS J., GINSBERG J., SHADE J., FULK D.: The digital michelangelo project: 3D scanning of large statues. In *Siggraph 2000, Computer Graphics Proceedings* (2000), Akeley K., (Ed.), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 131–144.

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK

G., SEIDEL H.-P.: Multi-level partition of unity implicits. *ACM Trans. Graph. 22*, 3 (2003), 463–470.

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. *ACM Trans. Graph. 22*, 3 (2003), 641–650.

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Siggraph 2000, Computer Graphics Proceedings* (2000), Akeley K., (Ed.), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 335–342.

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: A multiresolution point rendering system for large meshes. In *Siggraph 2000, Computer Graphics Proceedings* (2000), Akeley K., (Ed.), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 343–352.

[SJ00] SCHAUFLER G., JENSEN H.: Ray tracing point sampled geometry. In *Eurographics Rendering Workshop Proceedings* (2000), pp. 319–328.

[Wal05] WALD I.: High-Quality Global Illumination Walkthroughs using Discretized Incident Radiance Maps. *Technical Report, SCI Institute, University of Utah, No UUSCI-2005-010 (submitted for publication)* (2005).

[WS05] WALD I., SEIDEL H.-P.: Interactive Ray Tracing of Point Based Models. In *Proceedings of 2005 Symposium on Point Based Graphics* (2005).