

Interactive Particle Visualization with Advanced Shading Models using Lazy Evaluation

Christiaan P. Gribble¹ and Steven G. Parker²

¹Visual Simulation Group, Department of Computer Science, Grove City College

²Scientific Computing & Imaging Institute, School of Computing, University of Utah

Abstract

Particle-based simulation methods are used to model a wide range of complex phenomena and to solve time-dependent problems of various scales. Effective visualizations of the resulting state will communicate subtle changes in the three-dimensional structure, spatial organization, and qualitative trends within a simulation as it evolves. We describe a visualization process targeting upcoming, highly parallel multicore desktop systems that enables interactive navigation and exploration of large particle datasets rendered with illumination effects from advanced shading models. These expensive illumination effects are evaluated lazily by decoupling interactive display and high quality rendering. We explore the performance characteristics of this approach and demonstrate its effectiveness using several large particle datasets.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Particle methods are commonly used to simulate complex phenomena in a wide variety of scientific domains. Using these techniques, computational scientists model such phenomena as a system of discrete particles that obey certain laws and possess certain properties. Particle-based simulation methods are particularly attractive because they can be used to solve time-dependent problems on scales from the atomic to the cosmological.

Frequently, millions of particles are required to capture the behavior of a system accurately. Such massive simulations lead to very large, very complex datasets, making interactive visualization a difficult task. Moreover, the need to simultaneously visualize both the large- and small-scale features within the data further exacerbate these issues. An effective particle visualization method will communicate subtle changes in the three-dimensional structure, spatial organization, and qualitative trends within the data as a simulation evolves, as well as enable easier navigation and exploration of the data through interactivity.

As particle-based simulations continue to grow in size and complexity, effective visualization of the resulting state becomes increasingly problematic. In particular, two issues arise. First, these datasets are difficult to visualize interactively because of their size. An effective visualization algorithm must be capable of rendering such a large number of particles efficiently. Second, the intricacies of complex data are difficult to convey sensibly. Particle methods often simulate complex objects with subtle features that interact in complex ways, and detecting the salient features within the data is a critical step in correctly interpreting the simulation

results. Unfortunately, the proper way to communicate this information is not well-understood by the visualization or perception communities.

Visualization systems typically employ local shading models when rendering large datasets. Local shading models are efficient, do not introduce shading artifacts that could be misinterpreted as artifacts in the underlying data, and provide cues about the orientation of a surface because the local properties at a given point are considered during shading. Other surfaces in the environment are not considered, however, so effects from the interaction of light with these surfaces are either ignored or crudely approximated.

A recent psychophysical study has demonstrated that advanced illumination effects can aid attempts to comprehend important features within complex particle datasets [GP06]. In contrast to purely local models, advanced shading models such as ambient occlusion and physically based diffuse interreflection provide more accurate approximations to the light transport equation [ICG86, Kaj86] and capture illumination effects that can enhance the perception of complex shapes with subtle features.

Unfortunately, advanced shading models that simulate global effects are computationally expensive, and current algorithms are not particularly well-suited to interactive use. We describe an interactive particle visualization process in which the computational limitations of expensive illumination effects are mitigated by evaluating these effects lazily. The results are stored in per-particle texture maps, called dynamic luminance textures (DLTs), which are then cached and reused throughout an interactive session. The algorithm targets upcoming, highly parallel multicore desktop com-

puter systems and enables interactive navigation and exploration of large particle datasets rendered with illumination effects from advanced shading models.

2. Background and Related Work

This research is motivated by the need to visualize data from a particle-based simulation technique called the material point method (MPM) [SZS94, SZS95]. MPM is a particle-in-cell [Har64] simulation technique that is particularly well-suited to problems with high deformations and complex geometries. While we demonstrate our approach using MPM data from simulations of structural mechanics problems, our approach is applicable to particle data from other simulation methods and other application domains as well.

2.1. Particle Visualization

Investigators typically use particle visualization to assist efforts in data analysis and feature detection, as well as in debugging ill-behaved solutions. One approach to particle visualization projects the particle values to a three-dimensional grid, and the transformed data is then visualized using standard techniques such as isosurface rendering [LC87] and direct volume rendering [Lev88].

Grid-based representations are suitable for some, but not all, particle visualization tasks. The limited resolution of the grid itself can be problematic: fine structural details within the data may be lost. To alleviate this issue, the grid can be refined, either uniformly or adaptively. However, investigators are often interested in simultaneously examining both the large- and small-scale structures within the data, so grid-based visualization techniques may not be appropriate. Additionally, interpolation may hide features or problems present in the original particle data. For example, small particles that have extremely high velocities may be invalid, but the influence of such particles can be masked by interpolation. Moreover, interpolation and isosurface extraction can be time-consuming tasks, particularly for large datasets.

Particles can also be represented directly by simple, iconic shapes called glyphs. For many applications, a sphere or an ellipsoid is a natural representation of an individual particle. Glyph-based representations are able to preserve the fine details within the data while maintaining the large-scale three-dimensional structure of the entire domain. The resulting visualizations are particularly useful for the data analysis and code development tasks that investigators often perform.

Several efforts have explored techniques to render large numbers of spheres efficiently, from rasterization on massively parallel processors [KPH97], visualization clusters [LMC04], custom hardware [ZTH03, ZHC*04], and programmable graphics hardware [GGSP06] to interactive ray tracing on tightly coupled supercomputers [BGG*06] and highly parallel multicore systems [GIK*].

Our approach targets upcoming, highly parallel multicore desktop systems. The current implementation leverages the Manta interactive ray tracing system [BSP06], which has

been designed for both interactivity and flexibility. This system has been shown to scale efficiently to large numbers of processors for complex visualization tasks [SBB*06].

2.2. Interactive Global Illumination

Interactively rendering images of complex environments with full global illumination computed in every frame currently remains out of reach. However, several rendering algorithms offer alternatives for efficient computation of global illumination effects. These techniques generally query data structures that store illumination samples of the environment, or maintain interactivity by limiting the number and cost of paths traced during each frame.

For example, systems based on ray tracing or rasterization can include global illumination by precomputing and storing the effects, and later using the results during interactive rendering. Such an approach requires a representation of the precomputed solution that is appropriate for use in an interactive renderer. Several such representations have appeared in the literature, ranging from illumination maps [Arv86] and grid-based structures [GSHG98] to representations in complex bases like spherical harmonics [RH01, SKS02] and non-linear wavelets [NRH03, NRH04].

Advanced illumination effects can also be sampled lazily during interactive rendering, and the samples can then be cached and reused when appropriate. As with precomputation, this method requires a representation of the computed results that is suitable for caching, as well as the ability to reuse the cached samples. Additionally, these techniques require the ability to detect out-of-date samples that must be recomputed. Methods such as the irradiance volume [GSHG98], the render cache [WDG02, WDP99], and several others utilize this approach.

Other algorithms decouple parts of the rendering process to facilitate better user interaction and faster image generation [BDT99, WDP99, WDG02]. Typically these methods decompose the interactive pipeline into two distinct, asynchronous components: an interaction loop that responds to user input, and a high quality rendering engine that continually updates the image as quickly as possible. These approaches provide a responsive environment with which the user interacts, but maintain image quality by employing an expensive rendering engine.

3. Dynamic Luminance Textures

We seek practical methods that include effects from advanced shading models in an interactive particle visualization process. One possible approach involves precomputed luminance textures (PLTs) [GP06], in which per-particle texture maps are used to capture illumination effects from advanced shading models. Unfortunately, this approach imposes lengthy preprocessing phases in which the luminance textures are generated for every particle in a particular dataset. Then, the results are typically compressed to improve manageability of the resulting data.

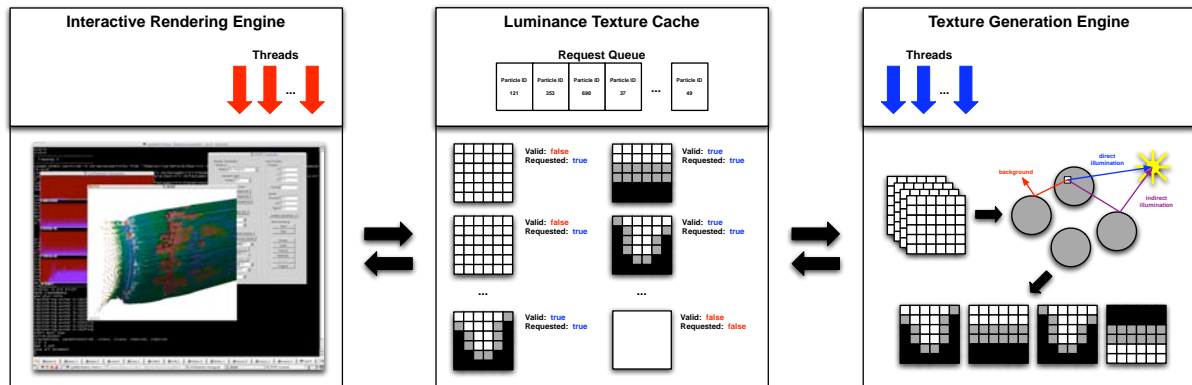


Figure 1: Major components of the DLT particle visualization pipeline. The interactive rendering engine responds to user input, displays the currently visible particles, and generates texture requests (left). The luminance texture cache manages outstanding requests and completed textures (middle). The texture generation engine asynchronously satisfies outstanding texture requests using Monte Carlo path tracing (right).

The approach described here is motivated by lazy evaluation of illumination effects. Rather than generate the per-particle luminance textures during a preprocessing phase, the DLT approach evaluates the user-specified shading model on-the-fly during interactive visualization.

The DLT algorithm consists of three components. The first is an interactive rendering engine that responds to user input, displays the currently visible particles, and generates luminance texture requests for these particles on-the-fly. The second is a texture generation engine based on Monte Carlo path tracing [Kaj86]; this engine asynchronously satisfies outstanding texture generation requests. The third component is a luminance texture cache that manages the requests and makes completed textures available for use in subsequent frames. These components are illustrated in Figure 1.

3.1. Interactive Rendering Engine

The interactive front-end responds to user input and displays the currently visible particles. During rendering, the status of the texture corresponding to a visible particle is determined by first querying the luminance texture cache. If the texture is valid, the values in the appropriate texels are used during shading. Specifically, the appropriate (u, v) texture space coordinate of a visible point is computed, the four values required to bi-linearly interpolate the luminance value at that point are queried, and the resulting value is multiplied by the color of the particle. However, if the texture is invalid, a request is submitted to the texture cache and the particle is temporarily shaded using the Lambertian shading model. The results of this process are depicted in Figure 2.

In our current implementation, the Manta interactive ray tracing system [BSP06] serves as the interactive front-end. Two important features were added to this system to support the DLT algorithm. First, Manta was extended to support efficient rendering of large, time-varying particle datasets using multilevel grids [PMS*99]. Second, a material shader

that either interpolates textured luminance values or generates texture requests and temporarily shades particles using Lambertian shading and shadows was added to the system. Although we currently employ the Manta framework for interactive display, a front-end based on programmable graphics hardware could be modified to support DLTs as well.

3.2. Texture Generation Engine

The texture generation engine satisfies outstanding texture generation requests. When work is available, the engine dequeues the request and allocates memory for the texture. Then, samples in the (u, v) parameter space of the texture are generated and mapped to the corresponding particle. Rays originating at these points are traced through the scene according to the user-specified shading model, and the resulting luminance values are stored in the corresponding texel. When the texture is complete, the status of the appropriate entry in the cache is updated.

In the DLT approach, textures are generated on-the-fly, so the texture generation engine must communicate with the texture cache. We have implemented two types of texture generation engines within the Manta framework: internal and external. Experimentation shows that the internal engine typically provides the desired performance characteristics [Gri06], and we describe the details here.

With internal texture generation, Manta rendering threads process outstanding requests. These threads (or some subset of them) dedicate a user-specified portion of the per-frame rendering time to texture generation via Manta's transaction processing framework [BSP06]. A parallel callback is first registered during initialization. The threads invoke the callback at the beginning of every frame and dynamically switch to texture generation, processing outstanding requests for the user-specified time interval. When the control timer expires, texture generation ceases (even if outstanding requests remain) and the threads proceed to interactive rendering. Texture generation within Manta is illustrated in Figure 3.

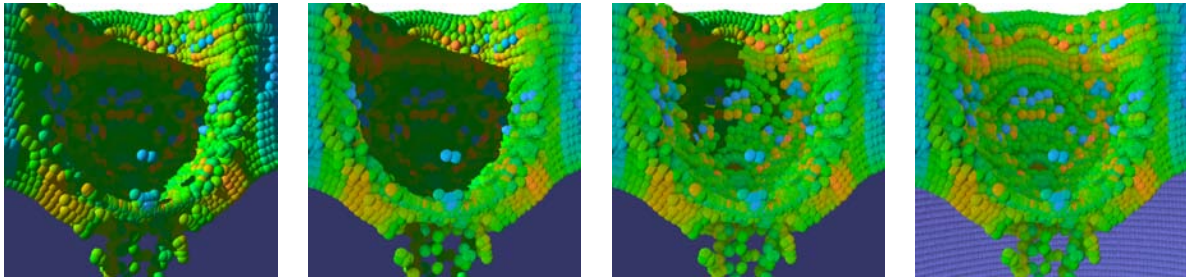


Figure 2: Interactive rendering and asynchronous luminance texture generation. The interactive rendering engine responds to user input, displays the currently visible particles, and generates luminance texture requests for these particles on-the-fly. The texture generation engine asynchronously satisfies these requests using Monte Carlo path tracing. (In these images, completed textures have been artificially brightened to draw attention; in practice, the results of using luminance textures are more subtle.)

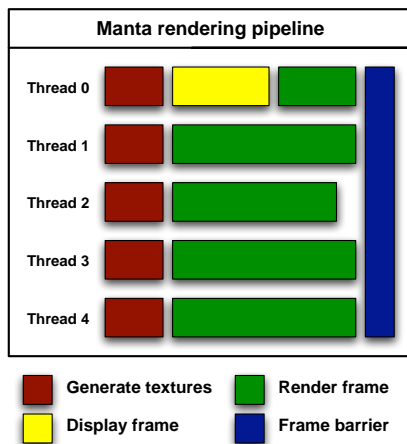


Figure 3: Texture generation in Manta. Texture generation requests are processed by a parallel callback executed within the Manta pipeline. When the rendering threads invoke the callback, some user-specified portion of the total frame time is dedicated to texture generation before rendering the current frame.

This process imposes an upper bound on the achievable frame rate. For example, if the rendering threads dedicate 0.03 seconds to texture generation, the highest achievable frame rate will be 30 frames per second. Similarly, this approach places a lower bound on the texture generation latency. If 0.03 seconds are dedicated to texture generation, then the minimum latency will be 0.03 seconds.

3.3. Luminance Texture Cache

The luminance texture cache is a globally accessible data structure that manages the state related to dynamically generated textures. In particular, the cache stores completed textures and makes the results available for use by the interactive front-end. Requests for new luminance textures are also managed by the cache and communicated to the texture generation engine via the request queue.

When a request is processed by the texture generation engine, memory for the texture is allocated, if necessary. The memory corresponding to a particular texture is reused

whenever possible, as in the case of invalidating the currently valid entries within the cache in response to changes in the underlying geometry or lighting configurations. Once memory for the texture is available, the texture generation engine satisfies the request as described above. Upon completion, the result is cached and the status of the appropriate entry is updated, indicating that the texture is available for use in subsequent frames.

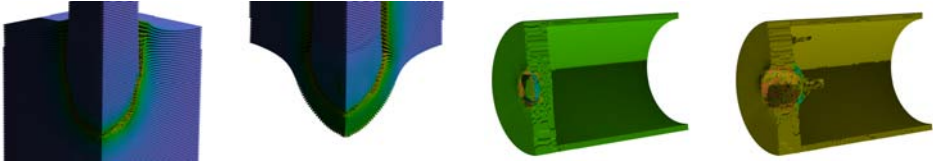
New texture requests are handled by the request queue. When the interactive rendering engine encounters a visible particle, the cache is queried concerning the state of the corresponding texture. If the texture has not yet been requested, either a unique time stamp or a priority measure is generated for the current particle, the request is stored at the appropriate position within the queue, and the status of the corresponding particle is updated.

We have implemented three request scheduling strategies: first-in, first-out (FIFO); last-in, first-out (LIFO); and priority-based scheduling. Experimentation shows that priority-based scheduling provides the desired performance characteristics [Gri06], and we describe the details here.

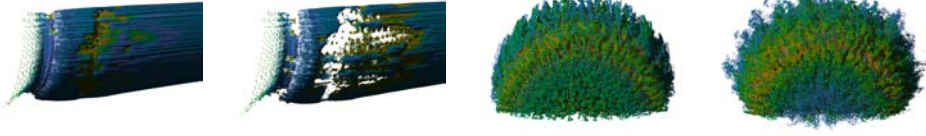
With priority-based texture request scheduling, each request is assigned a priority, and the request queue sorts outstanding requests such that the request with the highest priority always resides at the front of the queue.

Some issues with this scheme deserve attention. For example, this approach requires an appropriate priority metric. Both the FIFO and LIFO strategies are implicitly priority-based schemes. The priority of each request is determined by a unique time stamp, and the requests are ordered accordingly: ascending order (oldest first) in the FIFO queue, and descending order (newest first) in the LIFO queue. The requests are implicitly sorted because the time stamp is a monotonically increasing function.

With explicit priority-based scheduling, a more complicated priority metric is necessary to overcome the implicit ordering resulting from the time stamp function. In general, requests corresponding to the currently visible particles should be given preference over those generated earlier,



	Bullet-2	Bullet-7	Cylinder-6	Cylinder-22
# particles	569523	549128	214036	212980
Data size	13.04 MB	12.57 MB	6.53 MB	6.50 MB
Frame rate	5.20 fps	3.59 fps	2.74 fps	2.46 fps



	Fireball-10	Fireball-12	JP8-128	JP8-173
# particles	954903	951449	834271	809533
Data size	14.57 MB	14.52 MB	22.28 MB	21.62 MB
Frame rate	3.54 fps	3.11 fps	3.40 fps	2.85 fps

Table 1: Particle datasets used to evaluate the DLT approach. These datasets exhibit a wide variety of sizes and geometric complexity, and each represents a single time step of the full simulation. We evaluate a working implementation of the DLT approach using the viewpoints and time steps shown above. (Frame rates reflect performance while servicing outstanding texture generation requests.)

which argues for an ordering based on timestamps. However, interactions between a strict time-based ordering and the Manta rendering engine result in distracting artifacts because, in this case, requests are always generated in a very specific order. Thus, the following heuristic is used to compute the priority of each request: $P(x) = T(x) - D(x)$, where the priority $P(x)$ of a request for particle x is the difference between the time $T(x)$ that request is generated and the distance $D(x)$ from the particle to the current viewpoint. This simple metric biases request ordering towards particles that have been encountered recently and that are close to the current viewpoint. While several other heuristics can be used, user experience shows that this heuristic does not lead to visually distracting artifacts and ensures that particles encountered more recently are given preference. In addition, this metric can be evaluated almost as quickly as the time stamp alone, requiring just one additional operation.

A data structure capable of dynamically ordering the requests such that the highest priority request is always at the front of the queue is also required with this strategy. We currently implement the priority queue using a tree-based heap structure. With this design, pushing and popping operations can be implemented in a straightforward manner, and the ordering constraint does not impose measurable overhead.

4. Results

To quantify the performance of the DLT approach, we use the datasets depicted in Table 1. The results reported in this section were gathered by rendering several frames at 512×512 resolution using a 16 core Opteron machine with 2.6 GHz processors and 64 GB of physical memory.

We first quantify the performance of priority-based scheduling. Textures are generated using physically based diffuse interreflection, 16×16 texels, and 49 samples per texel. Each rendering thread dedicates at least 0.04 seconds to texture generation. The results are reported in Table 2.

We observe that the number of textures generated in each frame is greater than the number of threads, indicating that the average texture requires less than the maximum 0.04 seconds that threads dedicated to texture generation in each frame. Also, the priority queue minimizes the number of new requests generated in each frame. Currently, the size of the queue is, in theory, unbounded as low-priority requests are not dropped; instead, the priorities of previously requested textures are periodically updated if the corresponding particles remain visible for a user-specified time interval. In practice, however, the size of the request queue immediately grows to the number of particles visible in the first frame, and never grows larger than the maximum number of particles visible during an interactive session.

The DLT approach generates and caches textures for only the particles that are visible throughout an interactive session. As a result, the memory required to store dynamically generated textures is considerably less than that required by PLTs. Table 3 compares the memory consumed by textures in the cache with the size of uncompressed and PCA compressed PLTs. The number of visible particles is significantly less than the total number of particles in each dataset, ranging from 2.38% (Bullet-7) to roughly 12% (Cylinder-22). These data indicate a dramatic reduction in the number of textures that must be resident in memory. The memory con-

Dataset	# generated	Latency
Bullet-2	21.40	0.04
Bullet-7	22.88	0.06
Cylinder-6	27.36	0.20
Cylinder-22	21.04	0.15
Fireball-10	37.72	0.15
Fireball-12	33.60	0.14
JP8-128	23.92	0.09
JP8-173	19.20	0.08

Table 2: Priority-based request scheduling. The average number of textures generated in each frame and the average latencies (in seconds) for requests in the queue using 16 threads on the test machine. In general, priority-based scheduling provides desirable performance characteristics.

Dataset	% visible	DLT	PLT	PLT-PCA
Bullet-2	4.5%	6.3	139.0	4.4
Bullet-7	2.4%	3.2	134.1	4.2
Cylinder-6	11.6%	6.3	52.3	1.6
Cylinder-22	11.9%	6.2	52.0	1.6
Fireball-10	8.1%	18.8	233.1	7.3
Fireball-12	7.3%	17.0	232.4	7.3
JP8-128	7.7%	15.6	203.7	6.4
JP8-173	8.9%	17.6	197.6	6.2

Table 3: Memory requirements for dynamically generated luminance textures. The memory (in megabytes) consumed by DLTs is a factor of 8.24–42.03 less than that consumed by uncompressed PLTs, and is comparable to that required by PCA compressed PLTs storing eight basis textures. (PLT/PLT-PCA data after Gribble [Gri06].)

summed by uncompressed PLTs can be reduced by factor of 8.24–42.03 using uncompressed, dynamically generated luminance textures. The requirements are thus comparable to those of PLTs that have been compressed using PCA; the use of DLTs does not impose any texture compression phases, however.

We also examine the impact of DLTs on interactive visualization performance. Lambertian shading with shadows serves as the baseline. To isolate the overhead imposed by cache query operations, the texture generation control timer is set to expire immediately. As the data in Table 4 show, these operations reduce performance by roughly a factor of three. Performance drops by an additional factor of 2.81–6.06 while outstanding requests are processed, but still permits fluid interactions with the data. However, when all outstanding requests have been satisfied, performance increases dramatically, achieving frame rates that are a factor of 1.10–1.58 higher than Lambertian shading with shadows.

Finally, the graph in Figure 4 summarizes the scaling characteristics of our current implementation. While the number of texture requests satisfied in a given frame scales roughly linearly with the number of threads, interactive per-

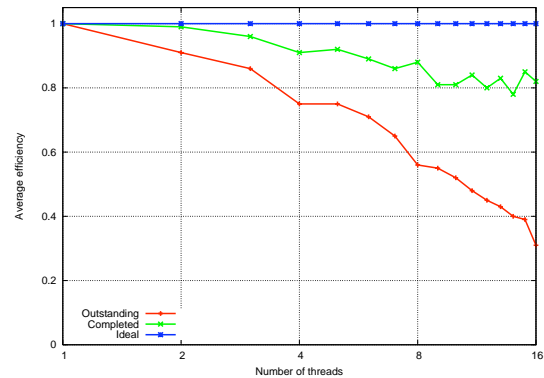


Figure 4: Average efficiency achieved by the DLT algorithm. Interactive performance does not improve when using a large number of threads to service outstanding texture generation requests (red), but the algorithm scales very efficiently once these requests have been satisfied (green).

formance does not improve substantially when using a large number of threads to service these requests. In this case, the operations imposed by texture generation and texture cache updates constitute a bottleneck and thwart efficient scaling. Once these requests have been satisfied, however, the algorithm scales to 16 threads with approximately 80% efficiency.

5. Conclusions and Future Work

We have introduced a particle visualization algorithm that lazily evaluates advanced shading models, permitting interactive navigation and exploration of large particle datasets rendered with expensive illumination effects. The algorithm targets upcoming, highly parallel multicore desktop systems and exhibits efficient scaling to 16 threads on the test machine.

The algorithm presented here brings perceptually beneficial effects from advanced shading models to the interactive visualization of particle-based simulation data. Moreover, because an investigator can interact with the whole dataset, a clear understanding of the state of each particle and its relationship to the full computational domain can be achieved.

Discussion. The DLT approach is designed to address some of the limitations inherent to an approach based on precomputation [GP06]. For example, the memory consumed by dynamic luminance textures is much less than that required by precomputing textures for every particle. As a result, texture compression is not required. Lazy evaluation also removes restrictions regarding the geometry and lighting configurations used during interactive visualization. Luminance textures can simply be recomputed by invalidating cached entries if these configurations change during interactive visualization. The DLT approach requires neither lengthy preprocessing phases nor texture compression, has lower memory requirements, and enables a wider range of interactions with the data than approaches based on precomputation.

Dataset	Lambertian	Query only	Texture requests	
			Outstanding	Completed
Bullet-2	46.13	17.44	5.20	72.95
Bullet-7	26.22	13.63	3.59	37.75
Cylinder-6	29.84	9.23	2.74	39.59
Cylinder-22	26.56	8.50	2.46	37.90
Fireball-10	59.17	21.44	3.54	65.37
Fireball-12	44.92	16.75	3.11	55.24
JP8-128	25.71	9.55	3.40	38.79
JP8-173	24.78	8.85	2.85	35.81

Table 4: Impact of DLTs on interactive visualization performance. Frame rates achieved using 16 threads on the test machine. Cache query operations reduce performance by roughly a factor of three, and texture generation has an additional impact. However, once outstanding texture generation requests have been satisfied, frame rates improve significantly.

We have evaluated the performance of our approach using a system with eight 2.6 GHz dual-core processors (16 processing cores total). The evaluation of our implementation shows good texture generation latencies and interactive frame rates on this reasonably priced machine. However, the theoretical peak available on the test machine is substantially less than the raw FLOPS provided by a single 3.2 GHz Cell processor [BWSF06]. With the advent of massively multi-core processors, compute power exceeding that of the test machine will be available on commodity desktop systems in the near future. The DLT algorithm is designed for such highly parallel architectures, and the scaling characteristics indicate that it can be modified to accommodate future architectures in a straightforward manner. The anticipated platforms thus suggest that our algorithm may facilitate interactive particle visualization with effects from advanced shading models on a single, very inexpensive processor.

Future work. Currently, computing an accurate solution to the light transport equation at highly interactive rates remains out-of-reach, even on multicore platforms like the test machine used to benchmark our implementation. As a result, some aspects of the DLT algorithm may warrant further investigation. For example, improving the performance of luminance texture generation by aggressive optimization of the path tracing engine would be valuable. Specifically, leveraging the coherent grid traversal algorithm for particle data [GIK*] during texture generation is one promising direction for improving performance. In addition, texture level-of-detail could be used to reduce texture generation time for perceptually unimportant particles. Such an approach would require a human behavior model incorporating factors from both space navigation [FTM97] and visual attention [SCCD04]. Different representations for storing illumination effects may also provide more accurate or more compact results. Spherical harmonics is just one of several alternatives that could be explored.

As discussed above, the memory required by the DLT approach is significantly less than that of uncompressed PLTs, so we have not explored texture compression in this context. However, there is nothing inherent to the DLT approach that precludes the use of texture compression with dynam-

cally generated textures. Incremental compression schemes, for example, those based on principal component analysis [OPK*04, Wen03, ZYK06], can be used to further reduce the memory required by dynamically generated textures.

Although the number of texture requests satisfied in a given frame scales roughly linearly with the number of texture generation threads, interactive performance does not improve significantly when using a large number of threads. This analysis indicates that the luminance texture cache represents the current bottleneck in our implementation. A decentralized caching scheme in which each thread manages some part of the cache may help to alleviate the bottleneck, particularly as more and more processing cores become available. More sophisticated request scheduling heuristics may also be helpful in this context, and could be combined with texture level-of-detail to further improve performance.

It would also be interesting to explore new global illumination algorithms that trade fidelity for computational complexity, and vice versa. For example, developing computationally simple, perceptually-equivalent approximations to indirect illumination is one promising direction. Likewise, it would be valuable to investigate specialized acceleration structures for on-demand computation of global illumination effects. Exploring these problems will potentially lead to more general solutions that can then be tailored to interactive particle visualization.

Parts of this work were funded by the DOE ASC program.

References

- [Arv86] ARVO J.: Backward ray tracing. *Developments in Ray Tracing (Siggraph '86 Course Notes) 12* (August 1986).
- [BDT99] BALA K., DORSEY J., TELLER S.: Radiance interpolants for accelerated bounded-error ray tracing. *ACM Transactions on Graphics 18*, 3 (1999), 213–256.
- [BGG*06] BIGLER J., GUILKEY J., GRIBBLE C., PARKER S., HANSEN C.: A case study: Visualizing material point method data. In *Proceedings of the Eurographics/IEEE Symposium on Visualization* (May 2006), pp. 299–306.
- [BSP06] BIGLER J., STEPHENS A., PARKER S. G.: Design for parallel interactive ray tracing systems. In *Proceedings of the*

- 2006 *IEEE Symposium on Interactive Ray Tracing 2006* (2006), pp. 187–196.
- [BWSF06] BENTHIN C., WALD I., SCHERBAUM M., FRIEDRICH H.: Ray tracing on the CELL processor. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing* (September 2006), pp. 15–23.
- [FTM97] FUJISHIRO I., TANAKA R., MARUYAMA T.: Human behavior-oriented adaptive texture mapping: A time-critical approach for image-based virtual showrooms. In *Proceedings of the 1997 Virtual Reality Annual International Symposium (VRAIS '97)* (March 1997), pp. 4–11.
- [GGSP06] GRIBBLE C. P., GUILKEY J. E., STEPHENS A. J., PARKER S. G.: Visualizing particle-based simulation data on the desktop. In *British HCI 2006 Workshop on Combining Visualization and Interaction to Facilitate Scientific Exploration and Discovery* (September 2006), pp. 1–8.
- [GIK*] GRIBBLE C. P., IZE T., KENSLER A., WALD I., PARKER S. G.: A coherent grid traversal approach to visualizing particle-based simulation data. *IEEE Transactions on Visualization and Computer Graphics*. To appear.
- [GP06] GRIBBLE C. P., PARKER S. G.: Enhancing interactive particle visualization with advanced shading models. In *ACM Siggraph Third Symposium on Applied Perception in Graphics and Visualization* (July 2006), pp. 111–118.
- [Gri06] GRIBBLE C. P.: *Interactive Methods for Effective Particle Visualization*. PhD thesis, University of Utah, 2006.
- [GSHG98] GREGER G., SHIRLEY P., HUBBARD P., GREENBERG D.: The irradiance volume. *IEEE Computer Graphics and Applications* 18, 2 (1998), 32–43.
- [Har64] HARLOW F. H.: The particle-in-cell method for fluid dynamics. *Methods for Computational Physics* 3 (1964), 319–343.
- [ICG86] IMMEL D. S., COHEN M. F., GREENBURG D. P.: A radiosity method for non-diffuse environments. In *Proceedings of Siggraph 1986* (1986), pp. 133–142.
- [Kaj86] KAJIYA J. T.: The rendering equation. In *Proceedings of Siggraph 1986* (1986), pp. 143–150.
- [KPH97] KROGH M., PAINTER J., HANSEN C.: Parallel sphere rendering. *Parallel Computing* 23, 7 (1997), 961–974.
- [LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: a high resolution 3D surface construction algorithm. In *International Conference on Computer Graphics and Interactive Techniques* (1987), pp. 163–169.
- [Lev88] LEVOY M.: Display of surfaces from volume data. *IEEE Computer Graphics and Applications* 8, 3 (1988), 29–37.
- [LMC04] LIANG K., MONGER P., COUCHMAN H.: Interactive parallel visualization of large particle datasets. In *Eurographics Symposium on Parallel Graphics and Visualization* (2004), pp. 111–118.
- [NRH03] NG R., RAMAMOORTHI R., HANRAHAN P.: All-frequency shadows using non-linear wavelet lighting approximations. *ACM Transactions on Graphics* 22, 3 (2003), 376–381.
- [NRH04] NG R., RAMAMOORTHI R., HANRAHAN P.: Triple product wavelet integrals for all-frequency relighting. *ACM Transactions on Graphics* 23, 3 (2004), 477–487.
- [OPK*04] OZAWA S., PANG S., KASABOV N., ZHANG C., GUESGEN H. W.: A modified incremental principal component analysis for on-line learning of feature space and classifier. In *Pacific Rim International Conference on Artificial Intelligence* (August 2004), pp. 231–240.
- [PMS*99] PARKER S., MARTIN W., SLOAN P.-P. J., SHIRLEY P., SMITS B., HANSEN C.: Interactive ray tracing. In *Symposium on Interactive 3D Graphics* (1999), pp. 119–126.
- [RH01] RAMAMOORTHI R., HANRAHAN P.: An efficient representation for irradiance environment maps. In *Proceedings of Siggraph '01* (August 2001), pp. 497–500.
- [SBB*06] STEPHENS A., BOULOS S., BIGLER J., WALD I., PARKER S. G.: An application of scalable massive model interaction using shared memory systems. In *Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization* (May 2006), pp. 19–26.
- [SCCD04] SUNDSTEDT V., CHALMERS A., CATER K., DEBATTISTA K.: Top-down visual attention for efficient rendering of task related scenes. In *Vision, Modelling and Visualization* (November 2004).
- [SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics* 21, 3 (2002), 527–536.
- [SZS94] SULSKY D., ZHOU S., SCHREYER H. L.: A particle method for history dependent materials. *Computer Methods in Applied Mechanical Engineering* 118 (1994), 179–196.
- [SZS95] SULSKY D., ZHOU S., SCHREYER H. L.: Application of a particle-in-cell method to solid mechanics. *Computer Physics Communications* 87 (1995), 236–252.
- [WDG02] WALTER B., DRETTAKIS G., GREENBERG D. P.: Enhancing and optimizing the render cache. In *Proceedings of Eurographics Workshop on Rendering* (2002), pp. 37–42.
- [WDP99] WALTER B., DRETTAKIS G., PARKER S.: Interactive rendering using the render cache. In *Eurographics Workshop on Rendering* (1999), Eurographics Association, pp. 19–30.
- [Wen03] WENG J.: Candid covariance-free incremental principal component analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25, 8 (2003), 1034–1040.
- [ZHC*04] ZEMCIK P., HEROUT A., CRHA L., FUCIK O., TUPPEC P.: Particle rendering engine in DSP and FPGA. In *11th International Conference and Workshop on the Engineering of Computer-based Systems (ECBS '04)* (2004), p. 361.
- [ZTH03] ZEMCIK P., TISNOVSKY P., HEROUT A.: Particle rendering pipeline. In *19th Spring Conference on Computer Graphics* (2003), pp. 165–170.
- [ZYK06] ZHAO H., YUEN P. C., KWOK J. T.: A novel incremental principal component analysis and its application for face recognition. *IEEE Transactions on Systems, Man, and Cybernetics (Part B)* 36, 4 (2006), 873–886.