Practical Global Illumination for Interactive Particle Visualization

Christiaan P. Gribble [a], Carson Brownlee [b], and Steven G. Parker [b]

[a]*Laboratory for Interactive Visualization, Entertainment, & Mobility Department of Computer Science, Grove City College, 100 Campus Drive, Grove City, PA 16217, USA*

[b]*Scientific Computing & Imaging Institute University of Utah, 50 S Central Campus Drive, MEB 3490, Salt Lake City, UT 84112, USA*

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 present two algorithms targeting upcoming, highly parallel multicore desktop systems to enable interactive navigation and exploration of large particle datasets with global illumination effects. Monte Carlo path tracing and texture mapping are used to capture computationally expensive illumination effects such as soft shadows and diffuse interreflection. The first approach is based on precomputation of luminance textures and removes expensive illumination calculations from the interactive rendering pipeline. The second approach is based on dynamic luminance texture generation and decouples interactive rendering from the computation of global illumination effects. These algorithms provide visual cues that enhance the ability to perform analysis and feature detection tasks while interrogating the data at interactive rates. We explore the performance of these algorithms and demonstrate their effectiveness using several large datasets.

Key words: interactive particle visualization, global illumination, ray tracing

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.

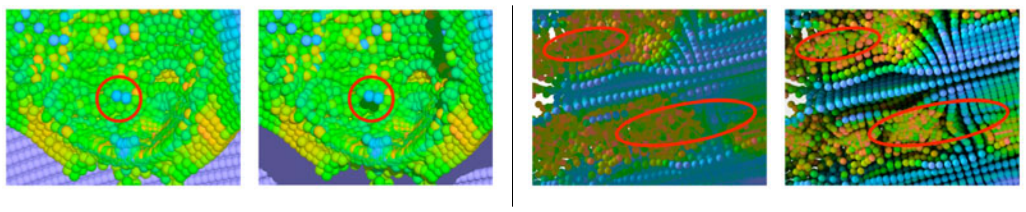


Fig. 1. *Global illumination in particle visualization.* Shadows help to disambiguate the relative position of objects in complex particle datasets (left). Although the crude approximation to indirect illumination used by local shading models tends to obscure geometric detail in shadowed regions, these details can be restored by using more advanced shading models (right).

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 interactive navigation and exploration of the data through interactivity. However, as particle-based simulations continue to grow in size and complexity, effective visualization of the resulting state becomes increasingly problematic. 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.

A recent psychophysical study has demonstrated that advanced illumination effects can aid attempts to comprehend important features within complex particle datasets [8]. 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 [11,12] and capture illumination effects that can enhance the perception of complex shapes with subtle features (see Fig. 1).

Unfortunately, advance shading models that simulate global effects are computationally expensive, and current algorithms are not particularly well-suited to interactive use. We introduce two algorithms that alleviate these issues and make global illumination effects practical for interactive particle visualization.

The first approach, which we call *precomputed luminance textures* (PLTs), overcomes the computational limitations of advanced shading models by removing the illumination calculation from the interactive rendering pipeline. In a preprocessing phase, the illumination across each particle is sampled using Monte Carlo path tracing and the results are stored in a luminance texture. To reduce the memory requirements imposed by this approach, either vector quantization (VQ) or principal component analysis (PCA) is used to compress the textures into a more manageable representation. During interactive rendering, the compressed textures are reconstructed (if necessary) and mapped to the particles, providing an approximation to the illumination in the scene. This simple, effective process enables the interactive navigation and exploration of large particle datasets with effects from advanced shading models.

The second approach, which we call *dynamic luminance textures* (DLTs), lazily evaluates advanced illumination effects by decoupling high quality rendering from interactive display. During rendering, requests for computationally expensive illumination effects are generated for the currently visible particles. A Monte Carlo path tracer satisfies these requests by sampling the illumination across the requested particles and storing the results in luminance textures. These textures are cached throughout a given interactive session and reused by the interactive renderer when appropriate. Textures are generated and stored for only visible particles, so the memory required by this approach is considerably less than that of the PLT approach. As a result, dynamically generated textures need not be compressed.

These algorithms offer not only enhanced visual cues, but also provide a testbed for future studies examining the perceptual impact of advanced shading modes on an interactive particle visualization process.

2 Background and Related Work

Our algorithms are motivated by the need to visualize data from a particle-based simulation technique called the material point method (MPM) [21,22]. MPM is a particle-in-cell simulation technique that is particularly well-suited to problems with high deformations and complex geometries. Although we evaluate our algorithms using MPM data from simulations of structural mechanics problems, both approaches are applicable to particle data from other simulation methods and other application domains as well.

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 grid, and the transformed data is then visualized using techniques such as isosurface rendering [16] and direct volume rendering [14].

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. Moreover, interpolation may hide features or problems present in the original particle data, and isosurface extraction can be a time-consuming task, 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. This representation is 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 [13], visualization clusters [15], programmable graphics hardware [9], and special-purpose hardware [28], to interactive ray tracing on tightly coupled supercomputers [3] and highly parallel multicore systems [7]. The algorithms we describe in this work target upcoming, highly parallel multicore desktop systems.

Interactive global illumination. Interactively rendering images of complex environments with full global illumination computed in every frame currently remains out of reach. However, several 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 of paths traced in 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 [1] and grid-based structures [6] to representations in complex bases like spherical harmonics [19] and non-linear wavelets [17].

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 [6], the render cache [26], and several others utilize this approach.

Other algorithms decouple parts of the rendering process to facilitate better user interaction and faster image generation [2,26]. Typically these methods decompose the interactive pipeline into two 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.

The PLT and DLT algorithms described in this work leverage these ideas to make computationally expensive global illumination effects practical for an interactive particle visualization process.

3 Luminance Textures

Texture mapping is a well-known technique in computer graphics that is used to add surface detail to the objects in a scene without explicitly modeling these details. Typically, an image (texture) is applied (mapped) to a surface in a process similar to pasting a decal onto the surface. Texture mapping often reduces the level of geometric detail that must be explicitly modeled while producing visually compelling results by encoding these details in an image.

Illumination maps [1] are simply texture maps that encode computationally expensive illumination effects such as diffuse or specular interreflection. These texture maps are created by computing the light that reaches a particular point in the scene and storing the result in the texel that maps to that point. In this way, global illumination effects can be computed and applied to the objects, adding illumination detail to the scene.

3.1 Perceptual Concerns

Shadows have been shown to provide important visual cues about the relative position of the objects within a scene. Understanding spatial relationships within complex particle datasets is a fundamental task in the data analysis process, but concerns about the use of shadows arise for two reasons. First, when used with local shading models, shadows often introduce ambiguities that result from the crude approximation to indirect illumination employed by these models. Second, discontinuities in shading resulting from hard shadows can be mistakenly interpreted as discontinuities in the underlying data.

We use soft shadows from area light sources to alleviate these concerns. Although soft shadows require integration of the incident illumination over the area of the light sources, Monte Carlo path tracing easily captures soft shadows in addition to other global illumination effects like indirect illumination.

The second potential artifact arises from the interplay of color mapping and illumination effects from advanced shading models. Color mapping is an effective way to communicate pertinent information beyond the spatial organization of objects within complex datasets [24]. Scalar values from the simulation

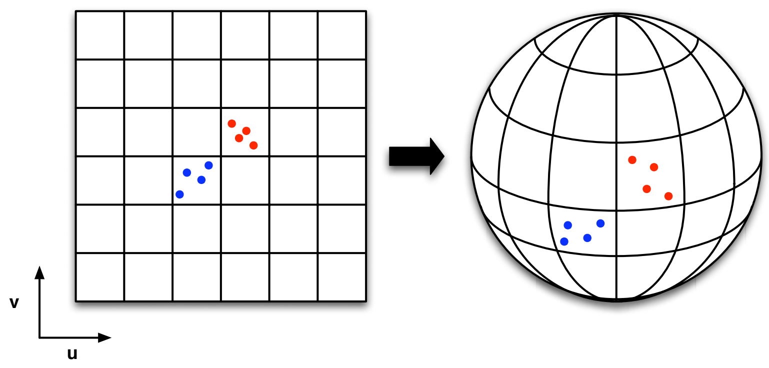


Fig. 2. *Generating luminance textures.* Jittered samples in the (u, v) parameter space of the texture are mapped to the particle surfaces. Rays originating at these points are traced through the scene, and the results are stored in the corresponding texel.

data associated with each particle are assigned a color that is used as the surface color during shading to convey these values. When color mapping is used, effects from color bleeding become a concern with shading models that include diffuse interreflection. For example, a white particle may appear pink if it reflects light from a nearby red particle. The reflection of red light from a white surface may misrepresent the values associated with the white particle, resulting in confusion or misinterpretation during data analysis.

To mitigate this issue, the algorithms we describe ignore the color of a particle during illumination computation and consider only reflected luminance. Luminance is simply the amount of light that passes through or is emitted from a particular area within a given solid angle, and does not account for the chromaticity of a surface; that is, luminance acts only as an indicator of how bright a surface appears. By computing only reflected luminance, our particle visualization algorithms avoid the potential problems associated with color bleeding. Moreover, both the color map and the particular data value used to determine surface color can be changed during rendering because these elements do not affect the reflected luminance within a scene.

3.2 Texture Generation

Luminance textures are simply illumination maps that store reflected luminance as described above. To compute the luminance textures, jittered samples in the (u, v) parameter space of a particular texture are generated and mapped to the current particle, as illustrated in Fig. 2. 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. This process continues until a texture has been generated for each particle.

We use a straightforward latitude-longitude mapping from the (u, v) parameter space of the textures to the world space coordinates of the particle surfaces. This mapping was chosen because of its simplicity and low overhead. We have found that a relatively low resolution texture (16×16 texels) with a small number of samples per texel (typically 25–49 samples) and 8-bit luminance values capture the illumination adequately. A uniform area mapping [20] may permit fewer samples to be used; however, for 16×16 textures, at most 65% more samples would be required under the current mapping to achieve an equivalent or better sampling density for all texels.

3.3 Texture Compression

Luminance textures may require a large amount of memory, even for relatively low resolution textures. For example, using a 16×16 texture and 8-bit luminance values, the textures for a single time step consisting of one million particles will consume over 244 MB of memory, or more than 20 times that consumed by the particle positions. The requirements for datasets with multiple time steps quickly become prohibitive.

Particles within some local vicinity typically exhibit similar illumination patterns, so we explore two texture compression schemes that exploit this redundancy. One is based on vector quantization, the other on principal component analysis.

Vector quantization. VQ maps k-dimensional vectors in the space $\mathbb{R}^k$ to a set of k-dimensional vectors $C = \{c_i : i = 1, 2, \ldots, N\}$. The set of vectors C is called the codebook, and each c_i is a codeword. Associated with each codeword is a Voronoi region defined by:

$$V_i = \{x \in \mathbb{R}^k : \|x - c_i\| \leq \|x - c_j\| \; \forall i \neq j\}.$$

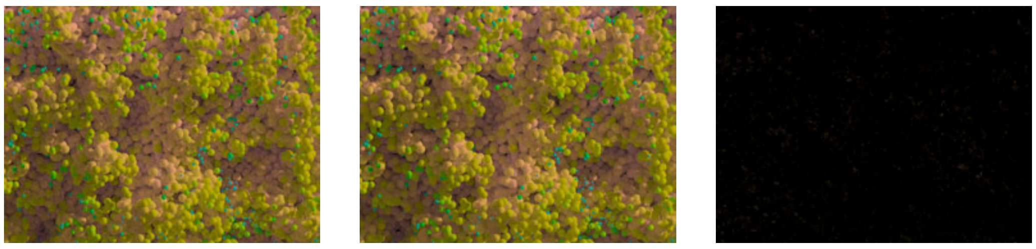
We use VQ to compress the luminance textures by treating each texture as a k-dimensional vector, where k corresponds to the product of the width and height of the texture.

There are two basic steps to compressing textures using VQ. First, vector pairs that minimize the distortion among the input vectors are found. The distortion is simply a measure of the distance between two vectors, x and y:

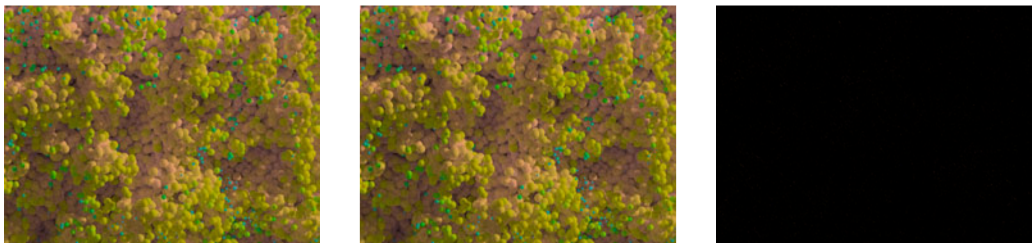
$$d(x, y) = \sum_{i=1}^{k} (x_i - y_i)^2.$$

Next, the minimum distortion pair is merged by computing the centroid of its vectors. The set of vectors that remain is then used in subsequent iterations. These steps are repeated until the desired number of codewords or a user-specified error threshold has been reached.

Principal component analysis. Texture compression based on PCA is another alternative. PCA uses statistical techniques to compute an orthonormal set of basis vectors in which the textures, treated as k-dimensional vectors, can be expressed. By storing only a subset of these vectors, the mean vector,



(a) Uncompressed vs. VQ



(b) Uncompressed vs. PCA

Fig. 3. *Comparing compression schemes.* VQ and PCA are both effective compression schemes. The first column shows the original textures, the second column depicts the textures compressed with VQ (a) and PCA (b), and the third column is a difference image of the results. There is little noticeable difference among the images, even though the texture data has been reduced by a factor of four.

and the associated per-object coefficients, an approximation to the original collection of textures can be reconstructed during interactive rendering.

The algorithm consists of three basic steps. First, the mean vector, $m_x = E\{x\}$, and the covariance matrix, $C_x = E\{(x - m_x)(x - m_x)^T\}$, are computed. An ordered set of (λ_i, e_i) pairs, where λ_i is the eigenvalue that corresponds to the eigenvector e_i, are computed from C_x using singular value decomposition. These eigenvectors serve as the basis vectors. Finally, the per-object coefficients are determined by computing the dot product of each input vector with each basis vector.

Using these compression schemes, reasonable reconstructions of the original textures can be obtained while dramatically and efficiently reducing the storage requirements. Though compression introduces an additional element of approximation, we have not found this approximation to be noticeable (see Fig. 3).

3.4 Using Luminance Textures

Any particle visualization system capable of applying luminance textures to the particles can be used for interactive rendering. Texture mapping proceeds by calculating the appropriate (u, v) texture space coordinate of a visible point, querying the four values required to bilinearly interpolate the final luminance value at that point, and multiplying the resulting value by the color of the particle.

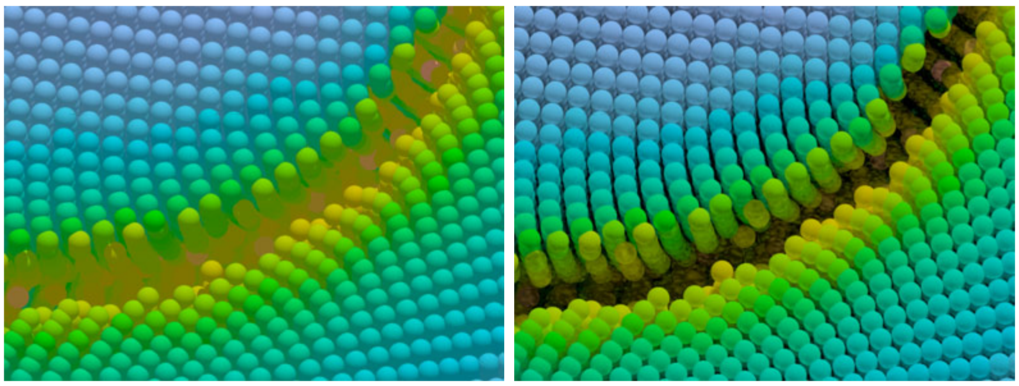


Fig. 4. *Visualizations of a particle dataset using PLTs.* The size and depth of the crack in this dataset are difficult to judge under Lambertian shading and shadows (left), but become more clear with diffuse interreflection captured by luminance textures (right).

Compressed textures impose some additional computations. VQ maps the original textures to a smaller set of codeword vectors using an integer index. In this case, the four luminance values are obtained by first determining the index of the codeword to which the texture of a particular particle has been mapped. The appropriate values of the codeword are then interpolated, and texture mapping proceeds normally. This simple indexing operation imposes no measurable impact on rendering performance with respect to using uncompressed luminance textures. In contrast, PCA computes an orthonormal basis that represents the axes of variation within the original textures, so the appropriate texels must be reconstructed before interpolating their values and assigning a color to the given pixel. Reconstruction requires a dot product between the per-object coefficients and the basis vectors. While this additional computation introduces some overhead, the impact on interactive performance is relatively small.

Fig. 4 compares the results of typical visualization using Lambertian shading and shadows with those obtained using luminance textures. The size and depth of the crack in this dataset become much more clear with diffuse interreflection effects captured by luminance textures.

4 Particle Visualization Algorithms

Our interactive particle visualization algorithms leverage the luminance textures described above to make the use of computationally expensive global illumination effects practical for interactive rendering.

4.1 Precomputed Luminance Textures

Precomputed luminance textures overcome the computational limitations of advanced shading models by removing expensive illumination calculations from the interactive rendering pipeline. In a preprocessing phase, the illumina-

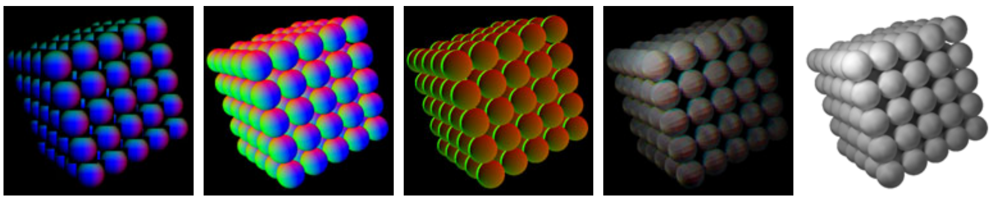


Fig. 5. *Using PLTs with programmable graphics hardware.* Precomputed luminance textures that have been compressed with PCA can be reconstructed and applied to the particles during interactive rendering using multipass fragment processing.

tion across each particle is sampled using Monte Carlo path tracing and stored in a luminance texture. To reduce the memory requirements imposed by this approach, either vector quantization (VQ) or principal component analysis (PCA) is used to compress the textures into a more manageable representation as described above. Then, during interactive rendering, the textures are mapped to the particles, approximating the illumination in the scene.

We have implemented two interactive particle visualization systems that support PLTs, one based on programmable graphics hardware, the other on interactive ray tracing.

Programmable graphics hardware. We extend the interactive particle visualization system described by Gribble et al. [9] to support PCA compressed PLTs. Using multipass fragment processing, texture coordinates are computed for each particle, and the PLTs are reconstructed from the compressed representation to produce the final image (see Fig. 5).

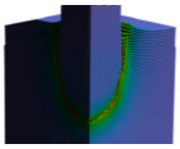
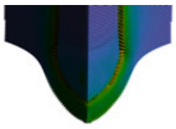
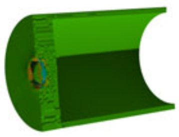
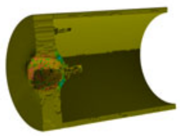
Interactive ray tracing. We also extend the basic particle visualization components of the Real-Time Ray Tracer (RTRT) [18] to support PLTs. RTRT efficiently determines the currently visible particles by traversing a multilevel grid, and the PLTs are then applied to these particles during shading.

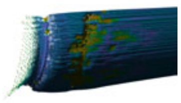
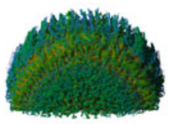
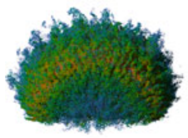
PLT results. The datasets depicted in Table 1 are used to quantify the performance of the PLT approach. The results reported in this section were gathered by rendering 1024×1024 images using a 16 core Opteron machine with 2.4 GHz processors and 64 GB of physical memory.

Textures were generated using physically based diffuse interreflection, 16×16 texels, and 49 samples per texel with 16 threads on the test machine. Preprocessing times are reasonable, despite the unoptimized path tracing engine used in our current implementation (see Table 2, Appendix A).

The size of the resulting texture collections motivates the need for texture compression. Many simulations contain tens or hundreds of time steps, so the memory demands imposed by the PLT approach will quickly overwhelm all but the most resourceful machines. To alleviate this issue, textures are compressed using either VQ or PCA and a user-specified compression ratio. Both VQ and PCA are effective compression schemes (see Table 3, Appendix A).

Table 1
Particle datasets used to evaluate our algorithms. These datasets exhibit a wide variety of sizes and geometric complexity, and each represents a single time step of the full simulation. We evaluate our algorithms using the viewpoints shown below.

				
	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

				
	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

Although textures compressed with VQ exhibit a lower mean distortion than those compressed with PCA, this quality comes at a price: VQ requires tens of hours to achieve even moderate compression ratios. Fortunately, execution times for PCA compression are very reasonable, typically just tens of seconds. Though the mean distortion exhibited by the PCA compressed textures is somewhat higher, this error does not have a noticeable impact (see Fig. 3).

Finally, we examine the impact of PLTs on interactive visualization performance using RTRT. Lambertian shading with shadows serves as the baseline. Frame rates were measured by rendering a series of 100 frames at 1024×1024 resolution with 16 threads on the test machine. Rendering performance improves by a factor of 1.14–1.91 when using PLTs (see Table 4, Appendix A). Our technique captures both shadows and diffuse interreflection during texture generation, which obviates the need for shadow computations during interactive rendering.

4.2 Dynamic Luminance Textures

Our second algorithm, which we call *dynamic luminance textures* (DLTs), is motivated by lazy evaluation of global illumination effects. As before, we use texture maps to capture global illumination effects, but these effects are now computed on-the-fly during interactive rendering.

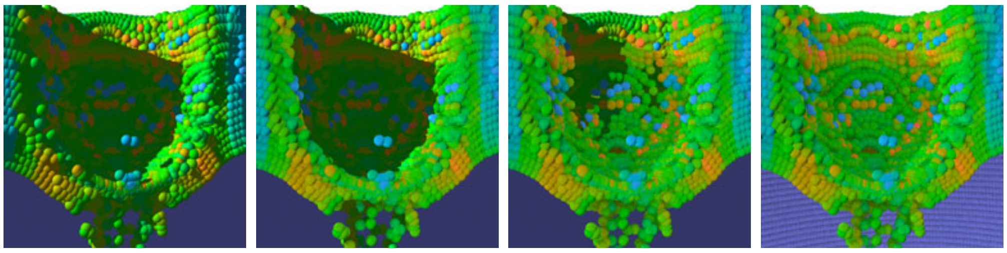


Fig. 6. *Interactive rendering and asynchronous texture generation.* The interactive rendering engine responds to user input, displays the currently visible particles, and generates luminance texture requests for these particles on-the-fly. The texture generation engine asynchronously satisfies these requests as quickly as possible.

Advanced illumination effects are evaluated lazily by decoupling high quality rendering from interactive display. Requests for computationally expensive effects such as soft shadows and diffuse interreflection are generated for the currently visible particles during interactive rendering. These requests are satisfied asynchronously by a Monte Carlo path tracing engine. The illumination across the visible particles is sampled and the results are stored in dynamically allocated luminance textures. These textures are cached throughout an interactive session and reused when appropriate.

Interactive rendering engine. The interactive front-end is responsible for responding to user input and displaying 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, it is applied to the particle. Otherwise, a request is submitted to the texture cache and the particle is temporarily shaded using the Lambertian shading model (see Fig. 6).

The Manta interactive ray tracing system [4] serves as the front-end in our current implementation. A front-end based on graphics hardware, similar to the one described above, could be modified to support DLTs as well.

Texture generation engine. Textures are generated using the process described above. However, the engine is implemented using either a collection of dedicated threads or a per-frame callback mechanism (see Fig. 7). In either case, when work is available, requests are dequeued and memory is allocated for the textures as necessary. When complete, the status of the corresponding entries in the texture cache is updated, and the textures become available for use in subsequent frames.

Using dedicated threads potentially reduces the latency between texture requests and completion because they continually satisfy outstanding requests and update the texture cache as quickly as possible; however, this approach will limit the achievable frame rate because the total number of threads in the system must be divided between texture generation and interactive rendering.

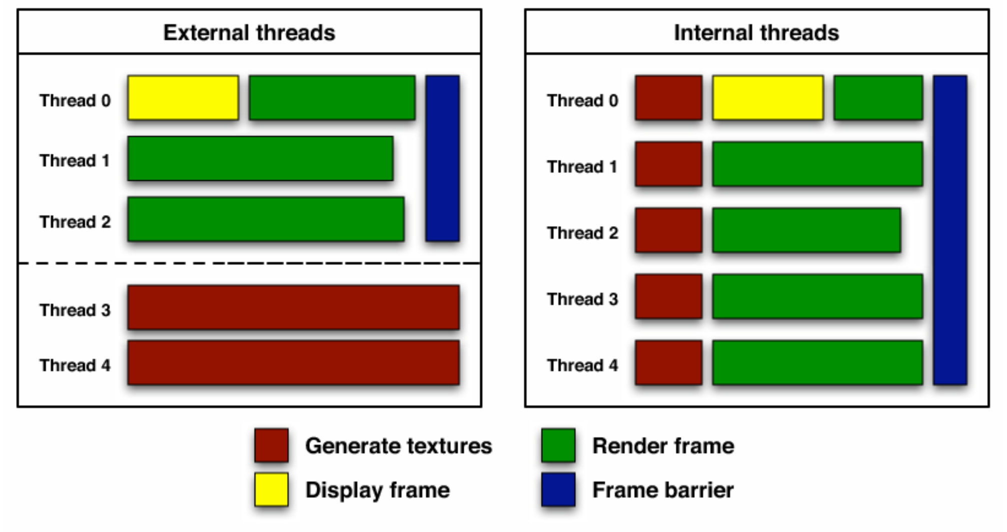


Fig. 7. *The texture generation engine.* Texture generation requests are processed either by a collection of dedicated threads (left) or by a per-frame callback executed within the interactive rendering pipeline (right).

Generating textures using a callback mechanism also imposes an upper bound on the achievable frame rate. For example, if each thread dedicates $\frac{1}{30}$ seconds to texture generation, the frame rate will necessarily be less than 30 frames per second. Similarly, this approach places a lower bound on the texture generation latency; in this example, the minimum latency will be $\frac{1}{30}$ seconds.

Luminance texture cache. The luminance texture cache 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 are communicated to the texture generation engine via the request queue. When the interactive front-end encounters a visible particle, the cache is queried concerning the state of the corresponding texture. If the texture is invalid, a prioritized request is submitted to the queue and awaits processing. The following heuristic determines the priority of each request:

$$P(x) = f(T(x) - D(x)),$$

where the priority $P(x)$ of a request for particle x is a function of 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 recently encountered particles that are close to the current viewpoint. While several other priority metrics can be used, this heuristic produces enough randomization to avoid distracting artifacts and ensures that recently encountered particles are given preference.

DLT results. The datasets depicted in Table 1 are used to quantify the performance of the DLT approach. The results reported in this section were gathered by rendering 512×512 images using a 16 core Opteron machine with 2.6 GHz processors and 64 GB of physical memory.

The DLT approach generates and caches textures for only the particles that are visible throughout an interactive session. The number of visible particles is significantly less than the total number of particles in each dataset (ranging from 2.38% to roughly 12%; see Table 5, Appendix A). Thus, the memory consumed by uncompressed PLTs can be reduced by factor of 8.24–42.03 using uncompressed, dynamically generated luminance textures. These requirements are thus comparable to those of PCA compressed PLTs.

We also examine the impact of DLTs on interactive visualization performance using Manta and per-frame callbacks for texture generation. Lambertian shading with shadows serves as the baseline. Frame rates were measured by rendering a series of 100 frames at 512×512 resolution with 16 threads on the test machine. The operations required to maintain the luminance texture cache reduce performance by roughly a factor of three (see Table 6, Appendix A). Performance drops by an additional factor of 2.81–6.06 while outstanding texture generation requests are processed, but still permits fluid interaction 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, Fig. 8 summarizes the scaling characteristics of the DLT algorithm. While the number of textures generated in each frame scales roughly linearly with the number of threads, interactive performance does not improve substantially when using a large number of threads. The operations imposed by texture generation and cache updates constitute a bottleneck and do not permit interactive rendering performance to scale efficiently. Once these requests have been satisfied, however, the algorithm scales to 16 threads with approximately 80% efficiency.

5 Conclusions and Future Work

The algorithms described above make perceptually beneficial effects from global illumination practical for an interactive visualization process. Using these methods, investigators can interact with the whole dataset and achieve a better understanding of the state of each particle, as well as its relationship to the full computational domain. In fact, informal feedback from application scientists indicates that the results of these algorithms enhance the data analysis tasks necessary for understanding complex particle datasets.

Discussion. The implementations described above do not leverage coherent grid traversal for particle visualization (PCGT) [7]; instead, they are based on highly optimized single ray traversal for multilevel grids [18]. Although RTRT does not facilitate packet-based ray tracing, Manta has been explicitly

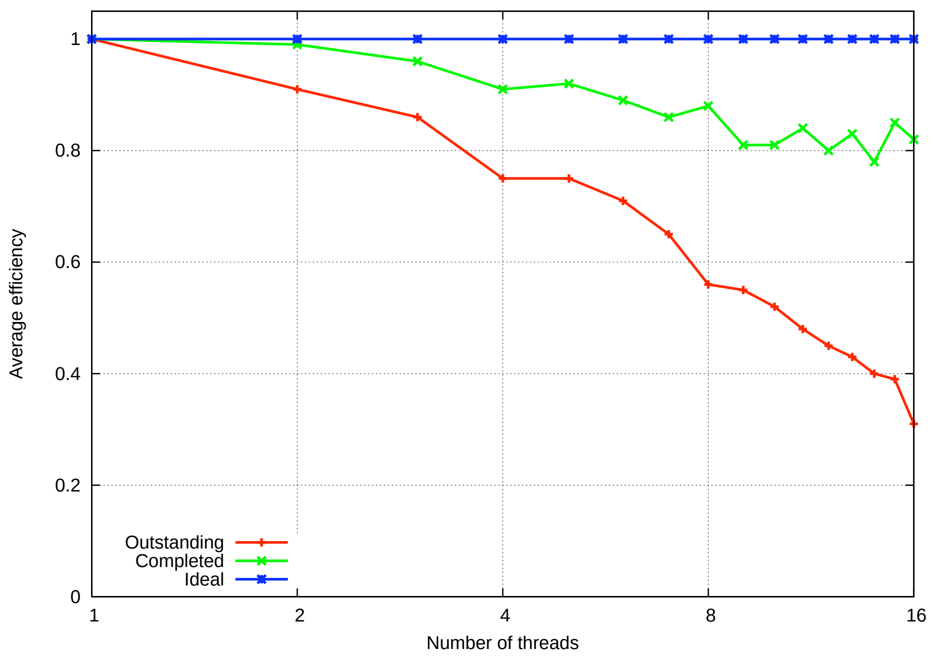


Fig. 8. *Average efficiency achieved by the DLT algorithm.* Frame rates do not improve while servicing texture generation requests, but the algorithm scales efficiently once these requests have been satisfied.

designed for such algorithms, and we have begun to explore the impact of PCGT on our algorithms.

Using PCGT to determine the currently visible particles is straightforward, and our algorithms simply proceed as described above. The performance improvements we observe are commensurate with the packet-based traversal scheme [7]. However, the constraints of PCGT limit its utility in texture generation: performance is dependent on packets of highly coherent rays, which Monte Carlo path tracing tends not to produce. We are currently investigating *coherent path tracing* using SIMD ray stream tracing [25], which extracts and exploits the coherence in arbitrarily sized groups of arbitrary rays.

Computing an accurate solution to the light transport equation at highly interactive rates currently remains out of reach, even on multicore platforms like the test machines used to evaluate our algorithms. However, with the advent of massively multicore processors like the CELL Broadband Engine [10], it is only a matter of time before compute power exceeding that of the test machines is available on commodity desktop systems. Our algorithms are designed for such highly parallel architectures, and can be adapted to future systems in a straightforward manner. The anticipated architectures thus suggest that these algorithms may facilitate interactive particle visualization with global illumination effects on a single, very inexpensive processor.

Future work. Some aspects of these algorithms warrant further investigation.

For example, improving the performance of texture generation by aggressive optimization of the path tracing engine would be valuable. As noted above, methods for coherent path tracing offer one promising direction. Texture level-of-detail could also 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 [5] and visual attention [23]. Additionally, alternate representations for storing illumination effects may 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 required by uncompressed PLTs, so we have not explored texture compression in this context. However, there is nothing inherent to the algorithm that precludes the use of texture compression with dynamically generated textures. Incremental compression schemes, for example, those based on PCA [27,29], could be used to further reduce the DLT memory requirements.

Although the number of texture requests satisfied in a given frame scales roughly linearly with the number of texture generation threads, analysis indicates that the luminance texture cache represents a bottleneck in our current 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.

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

References

[1] J. Arvo, Backward ray tracing, Developments in Ray Tracing (Siggraph '86 Course Notes) 12.

[2] K. Bala, J. Dorsey, S. Teller, Radiance interpolants for accelerated bounded-error ray tracing, ACM Transactions on Graphics 18 (3) (1999) 213–256.

[3] J. Bigler, J. Guilkey, C. Gribble, S. Parker, C. Hansen, A case study: Visualizing material point method data, in: Proceedings of the Eurographics/IEEE Symposium on Visualization, 2006.

[4] J. Bigler, A. Stephens, S. G. Parker, Design for parallel interactive ray tracing systems, in: Proceedings of IEEE Symposium on Interactive Ray Tracing 2006, 2006.

[5] I. Fujishiro, R. Tanaka, T. Maruyama, 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), 1997.

[6] G. Greger, P. Shirley, P. Hubbard, D. Greenberg, The irradiance volume, IEEE Computer Graphics and Applications 18 (2) (1998) 32–43.

[7] C. P. Gribble, T. Ize, A. Kensler, I. Wald, S. G. Parker, A coherent grid traversal approach to visualizing particle-based simulation data, IEEE Transactions on Visualization and Computer Graphics 13 (4) (2007) 758–768.

[8] C. P. Gribble, S. G. Parker, Enhancing interactive particle visualization with advanced shading models, in: ACM Siggraph Third Symposium on Applied Perception in Graphics and Visualization, 2006.

[9] C. P. Gribble, A. J. Stephens, J. E. Guilkey, S. G. Parker, Visualizing particle-based simulation data on the desktop, in: British HCI 2006 Workshop on Combining Visualization and Interaction to Facilitate Scientific Exploration and Discovery, 2006.

[10] M. Gschwind, The cell broadband engine: Exploiting multiple levels of parallelism in a chip multiprocessor, International Journal of Parallel Programming 35 (3) (2007) 233–262.

[11] D. S. Immel, M. F. Cohen, D. P. Greenburg, A radiosity method for non-diffuse environments, in: Proceedings of Siggraph 1986, 1986.

[12] J. T. Kajiya, The rendering equation, in: Proceedings of Siggraph 1986, 1986.

[13] M. Krogh, J. Painter, C. Hansen, Parallel sphere rendering, Parallel Computing 23 (7) (1997) 961–974.

[14] M. Levoy, Display of surfaces from volume data, IEEE Computer Graphics and Applications 8 (3) (1988) 29–37.

[15] K. Liang, P. Monger, H. Couchman, Interactive parallel visulization of large particle datasets, in: Eurographics Symposium on Parallel Graphics and Visualization, 2004.

[16] W. E. Lorensen, H. E. Cline, Marching cubes: a high resolution 3d surface construction algorithm, in: International Conference on Computer Graphics and Interactive Techniques, 1987.

[17] R. Ng, R. Ramamoorthi, P. Hanrahan, All-frequency shadows using non-linear wavelet lighting approximations, ACM Transactions on Graphics 22 (3) (2003) 376–381.

[18] S. Parker, W. Martin, P.-P. J. Sloan, P. Shirley, B. Smits, C. Hansen, Interactive ray tracing, in: Symposium on Interactive 3D Graphics, 1999.

[19] R. Ramamoorthi, P. Hanrahan, An efficient representation for irradiance environment maps, in: Proceedings of Siggraph '01, 2001.

[20] P. Shirley, K. Chiu, A low distortion map between disk and square, Journal of Graphics Tools 2 (3) (1997) 45–52.

[21] D. Sulsky, S. Zhou, H. L. Schreyer, A particle method for history dependent materials, Computer Methods in Applied Mechanical Engineering 118 (1994) 179–196.

[22] D. Sulsky, S. Zhou, H. L. Schreyer, Application of a particle-in-cell method to solid mechanics, Computer Physics Communications 87 (1995) 236–252.

[23] V. Sundstedt, A. Chalmers, K. Cater, K. Debattista, Top-down visual attention for efficient rendering of task related scenes, in: Vision, Modelling and Visualization, 2004.

[24] E. R. Tufte, The Visual Display of Quantitative Information, Graphics Press, Cheshire, Connecticut, 2001.

[25] I. Wald, C. P. Gribble, S. Boulos, A. Kensler, Simd ray stream tracing, in: Proceedings of IEEE Symposium on Interactive Ray Tracing 2007, 2007, poster abstract.

[26] B. Walter, G. Drettakis, S. Parker, Interactive rendering using the render cache, in: Eurographics Workshop on Rendering, Eurographics Association, 1999.

[27] J. Weng, Candid covariance-free incremental principal component analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (8) (2003) 1034–1040.

[28] P. Zemcik, A. Herout, L. Crha, O. Fucik, P. Tupec, Particle rendering engine in DSP and FPGA, in: 11^{th} International Conference and Workshop on the Engineering of Computer-based Systems (ECBS '04), 2004.

[29] H. Zhao, P. C. Yuen, J. T. Kwok, 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.

Appendix A

The tables below present detailed results concerning various aspects of our interactive particle visualization algorithms. The main body of the text references these tables where appropriate.

Table 2
Texture generation statistics. Texture sizes (MB) and run times (hh:mm) for the test datasets. Preprocessing times are reasonable, despite the unoptimized path tracing engine used in our current implementation.

Dataset	**# particles**	**Texture size**	**Run time**
Bullet-2	569523	139.04	1:27
Bullet-7	549128	134.06	1:13
Cylinder-6	214036	52.26	0:24
Cylinder-22	212980	52.00	0:26
Fireball-10	954903	233.13	3:53
Fireball-12	951449	232.39	6:07
JP8-128	834271	203.68	2:01
JP8-173	809533	197.64	2:07

Table 3
Texture compression statistics for JP8-173 using VQ/PCA. VQ typically leads to a lower mean distortion than PCA, but requires tens of hours to achieve the desired compression ratios. PCA typically requires just tens of seconds, however.

Ratio	**# textures**	**Run time**	**Distortion**
2:1	126979 / 122	21 h / 15 s	11.63 / 91.22
4:1	57165 / 55	34 h / 12 s	29.81 / 109.86
16:1	4171 / 10	30 h / 10 s	59.33 / 210.63

Table 4
Impact of PLTs on interactive visualization performance. Rendering performance improves by a factor of 1.14–1.91 when using PLTs. (The PCA compressed PLTs used in these tests store eight basis textures.)

Dataset	**Lambertian**	**Uncompressed**	**VQ**	**PCA**	**Speedup**
Bullet-2	5.21	6.89	6.89	6.15	1.32/1.18
Bullet-7	6.01	7.82	7.82	6.87	1.30/1.14
Cylinder-6	3.60	6.57	6.57	5.89	1.82/1.64
Cylinder-22	3.12	5.34	5.34	4.88	1.71/1.56
Fireball-10	1.54	2.91	2.91	2.79	1.89/1.81
Fireball-12	0.67	1.28	1.28	1.25	1.91/1.87
JP8-128	0.73	1.12	1.12	1.11	1.53/1.52
JP8-173	0.73	1.11	1.11	1.01	1.52/1.38

Table 5
Memory requirements for our algorithms. The memory, in megabytes, consumed by DLTs is 8.24–42.03 times less than that consumed by uncompressed PLTs, and are thus comparable to the requirements of PCA compressed PLTs that store eight basis textures.

Dataset	**# particles**	**% visible**	**DLT**	**PLT**	**PCA**
Bullet-2	569523	4.53%	6.31	139.04	4.35
Bullet-7	549128	2.38%	3.19	134.06	4.19
Cylinder-6	214036	11.64%	6.34	52.26	1.64
Cylinder-22	212980	11.93%	6.21	52.00	1.63
Fireball-10	954903	8.07%	18.84	233.13	7.29
Fireball-12	951449	7.32%	17.01	232.39	7.26
JP8-128	834271	7.66%	15.61	203.68	6.37
JP8-173	809533	8.91%	17.62	197.64	6.18

Table 6
Impact of DLTs on interactive visualization performance. Operations related to the DLT caching mechanisms 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.

Dataset	**Lambertian**	**Query only**	**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