SCI Institute, University of Utah, Technical Report No UUSCI-2006-024

A Coherent Grid Traversal Approach to Visualizing Particle-based Simulation Data

Christiaan P. Gribble   Thiago Ize   Andrew Kensler   Ingo Wald   Steven G. Parker

Scientific Computing and Imaging Institute, University of Utah
{cgribble|thiago|aek|wald|sparker}@cs.utah.edu

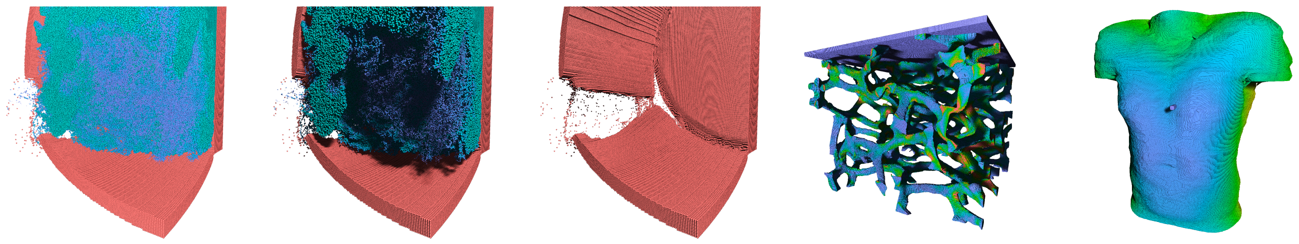


Figure 1: Visualizing particle-based simulation data with efficient ray tracing. We describe several optimizations that tailor the coherent grid traversal algorithm for efficient and effective visualization of complex particle-based simulation datasets. Our approach renders images of datasets with millions of particles at highly interactive rates and also provides run time control of several advanced visualization features, including time-varying data, color mapping, illumination effects from soft shadows, and parameter range culling. The interactive performance of our approach compares favorably with other systems that represent the current state-of-the-art in particle visualization.

Abstract

We investigate the use of interactive ray tracing for visualizing particle-based simulation data, and present an algorithmic enhancement called the sphere-center method that exploits the properties of these datasets to provide interactive performance and reduce storage requirements. This new algorithm for fast packet-based ray tracing of multi-level grids enables interactive visualization of datasets with millions of particles and incorporates advanced features like soft shadows. The size and complexity of typical particle datasets make efficient rendering a difficult task, but a variety of techniques are employed to achieve interactive performance for large, time-varying datasets. We compare the performance of our approach with two recent particle visualization systems: one based on an optimized single ray grid traversal algorithm, the other on programmable graphics hardware. This comparison demonstrates that the new algorithm offers an attractive alternative for interactive particle visualization.

1 Introduction

Over the past decade, ray tracing has become a viable option for visualizing a wide variety of scientific datasets, including those produced by particle-based simulation methods. These methods are commonly used to simulate complex phenomena in several scientific domains. Using particle techniques, computational scientists model such phenomena as a system of discrete particles that obey certain laws and possess certain properties. These 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, leading to very large, very complex datasets.

We are motivated by the need to visualize data from a particular particle simulation technique called the material point method (MPM) [21, 22]. MPM is gaining popularity among computational scientists for problems with high deformations and complex geometries such as those depicted in Figure 1. Effective visualization of particle-based simulation data will communicate subtle changes in the three-dimensional structure, spatial organization, and qualitative trends within the simulation as it evolves, as well as enable easier navigation and exploration of the data through interactivity.

Unfortunately, the size and complexity of typical particle datasets make interactive visualization a difficult task. Particle values can be projected to a three-dimensional grid, and the transformed data can then be visualized using standard techniques such as direct volume rendering [13] and isosurface rendering [14]. Grid-based representations of the data are suitable for some, but not all, particle visualization tasks, however. For example, the need to simultaneously visualize both the large- and small-scale features within the data often make grid-based representations problematic. Additionally, interpolation may hide features or problems present in the original particle data, while interpolation and isosurface extraction can be very time-consuming, 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. Combining this approach with programmable graphics hardware, particle data can be visualized directly by rendering either highly tessellated spheres or high quality spherical impostors (textured billboards). Unfortunately, tessellating millions of particles often results in too many triangles to be rendered at interactive rates, while implementing advanced lighting features such as soft shadows with impostor-based geometry is non-trivial.

In this paper, we investigate the use of interactive ray tracing for visualizing large, time-varying particle datasets. We present an efficient algorithm using fast packet-based ray tracing and multi-level grids. Currently, there are three acceleration structures that support packet-based ray tracing: multi-level kd-trees [20], bounding volume hierarchies (BVHs) [27], and multi-level grids [28]. We explore the latter for three reasons: First, the radii of particles within these datasets are typically uniform in size or fall within a well-defined range, and grids typically perform well for uniformly sized primitives. Second, the particles are typically either uniformly distributed throughout the environment or densely packed with large regions of empty space between them. While hierarchical data structures like kd-trees or BVHs often provide superior performance for scenes with varying primitive density, a grid can skip empty space as fast as these structures by using a macrocell hierarchy [17]. For densely packed regions, however, grids can be advantageous and often provide the best performance. As a result, multi-level grids are competitive with, if not superior to, a kd-tree or BVH for our application. Third, because of their fast rebuild times [9], grids easily handle time-varying data and offer the potential for computational steering within integrated problem solving environments. We introduce optimizations that exploit the properties of particle-based simulation data to tailor the coherent grid traversal algorithm [28] for particle datasets, achieving both improved performance and reduced storage requirements.

2 Background and Related Work

Our approach to particle visualization builds upon existing techniques in interactive ray tracing; we briefly review the relevant research related to our approach.

Particle Visualization. Investigators use particle visualization to assist efforts in data analysis and feature detection, as well as in debugging ill-behaved solutions. As such, interactivity plays an important role in the particle visualization process.

Two recent systems represent the current state-of-the-art in interactive particle visualization. On the one hand, interactive ray tracing on tightly-coupled supercomputing platforms is used to visualize large particle datasets at interactive rates [2]. Unfortunately, the hardware costs of such a system are often prohibitive and impede accessibility. At the other extreme, programmable graphics hardware brings interactive visualization of large, time-varying particle datasets to the desktop [6]. Though such hardware is widely accessible, interactive performance is limited by the fill rates of current GPUs, and advanced visualization features such as soft shadows are difficult to implement with impostor-based rendering. In this paper, we present an efficient algorithm for ray tracing large, time-varying particle datasets at interactive rates. This approach not only satisfies the requirements of effective particle visualization, but is more accessible than previous systems that require expensive hardware and easily incorporates advanced features that are difficult to implement using current graphics hardware.

Interactive Ray Tracing. Though the first interactive ray tracers utilized multi-level grids [16, 17, 18], algorithmic developments for traversal schemes based on kd-trees [20, 26] have significantly improved the performance of these structures. Packet-based ray tracing [26] creates groups of spatially coherent rays that are simultaneously traced through a kd-tree: each ray in the packet performs each traversal operation in lock-step. This packet-based approach enables effective use of SIMD instructions on modern CPUs, increases the computational density of the code, and amortizes the cost of memory accesses across multiple rays. Packet tracing has also lead to a new frustum based traversal algorithm for kd-trees [20] in which a bounding frustum guides the traversal of ray packets. The cost of a traversal step is thus independent of the number of rays bounded by the frustum and encourages large ray packets to achieve lower per-ray cost. This frustum based technique has recently been extended to bounding volume hierarchies [27] and multi-level grids [28], both of which support dynamic scenes.

Coherent Grid Traversal. For particle-based data visualization, we extend the coherent grid traversal (CGT) algorithm [28] to efficiently visualize large numbers of particles represented by spherical glyphs. We summarize the original algorithm here.

In CGT, ray packets are traversed through the grid by considering vertical slices rather than individual cells. Multiple cells in each slice are traversed by all of the rays in a packet, and each ray is tested against all of the objects within a given cell. Although this approach implies that some rays will traverse cells they would not have otherwise considered, the packet is traced as a coherent whole in each step, and no splitting or merging operations are required. The illustration in Figure 2 demonstrates this behavior.

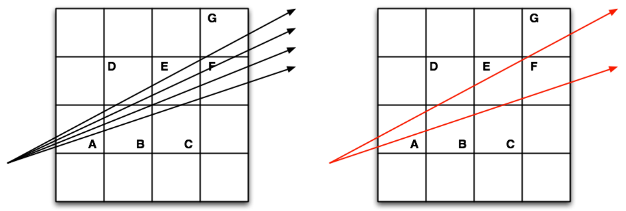


Figure 2: *Packet traversal in a grid.* In coherent grid traversal, rays step along vertical slices in the major traversal direction. Rays traverse the grid as a coherent whole, so no splitting or merging operations are required (left). The bounding frustum overlaps the same cells as the individual rays, and this frustum can be used to guide ray traversal (right).

Rays are first transformed into the canonical grid coordinates, so the location of any three-dimensional point p within the grid is determined simply by truncating its position (represented by floating point values) to the integer coordinates of the cell in which it lies. The dominant component of the first ray in the packet becomes the major traversal axis, $\vec{K}$, and the orthogonal directions are called $\vec{U}$ and $\vec{V}$. The cells overlapped by a ray packet are determined by stepping along $\vec{K}$. Each step k divides the grid into a two-dimensional plane of cells, and a rectangular subset of this slice contains all of the cells touched by the rays in the packet. This subset is determined by finding the extremal values in $\vec{U}$ and $\vec{V}$ over all rays in the packet, and simply truncating these values yields the u and v extents of all cells in slice k that are overlapped by at least one ray in the packet.

Rather than iterate over all of the rays in each step, however, a frustum is determined by simply computing the four planes bounding the ray packet in $\vec{U}$ and $\vec{V}$, as well as the near and far planes in $\vec{K}$. This bounding frustum will overlap the same cells as the rays (see Figure 2). Using these plane equations, the bounding frustum is intersected with the bounding box of the entire grid; a non-empty interval indicates that the grid should be traversed, and the extremal coordinates of the overlap determine the first and last slice along $\vec{K}$ that are traversed.

Due to the linearity of the plane equations, the grid is traversed by incrementing the bounding box of the current slice by a constant vector, the elements of which give the slopes of the bounding planes in the grid coordinate space. Assuming four-component SIMD operations, the incremental traversal step requires a single SIMD addition. Similarly, the truncation of the four values expressing the extents in $\vec{U}$ and $\vec{V}$ to the corresponding grid coordinates requires just a single SIMD float-to-int conversion. Thus, the entire process of computing the cells overlapped in the next slice requires only two SIMD operations.

3 Motivation

Interactivity is an important requirement of many scientific visualization applications, including particle visualization. Interactive visualization of simulation data typically serves one of three purposes: data analysis, code development, or generation of presentation and publication quality images. First, an interactive visualization process enables users to identify and explore the salient features of their data more effectively. For example, it is possible to correlate specific events in a given simulation with the behavior of each element in the simulation as it evolves. By determining visually exactly when these events occur, it is possible to recognize what relationship they have to the behavior of the system. Such observations enable specific insights like those described by Bardenhagen et al. [1]. Second, the ability to debug ill-behaved solutions is an obvious, but important, consequence of highly accessible interactive visualization. For example, recognition of incorrect particle behavior has led to important advances in MPM algorithms such as the modification described by Guilkey et al. [8]. Finally, interactive visualization makes generation of high quality animations and still images for presentation or publication fast and straightforward. An interactive environment allows a user to quickly identify optimal views in which each image or frame of an animation will convey the most pertinent information.

Interactivity in the context of particle visualization encompasses a wide range of activities. For example, interactive viewing and lighting enable investigators to identify and interrogate specific features within the data more easily. Interactivity also provides important visual cues from relative motion [24, 25] and environmental frame of reference [29], while advanced features such as parameter range culling and color mapping provide opportunities for additional insights. Parameter range culling allows investigators to isolate particles with properties that lie within some range of values, and color mapping offers an effective way to communicate pertinent information beyond the spatial organization of objects within complex datasets [23]. Using the modified CGT algorithm we describe in Section 4, each of these activities is under the full control of a user at run time and can be changed at interactive rates.

In addition to these advanced visualization features, important perceptual cues from non-local shading effects are easily integrated into our algorithm because it is based on ray tracing. For example, shadows are a well-studied visual cue that provide important information about shape and relative position [15, 29]. Unfortunately, hard shadows produced by point light sources often introduce discontinuities in the shading patterns on a surface that can be mistakenly interpreted as discontinuities in the underlying data. To combat this issue, we use soft shadows from area light sources. Soft shadows typically exhibit a smooth transition from shadowed to unshadowed regions, are less likely to be misinterpreted, and provide additional visual cues about the relative position of objects in complex datasets. Using our approach, users are able to control both the size and position of the light source, as well as the number of shadow rays, interactively. Recent research has also shown that visual cues from advanced shading models such as physically based diffuse interreflection can also be beneficial in the context of particle visualization [7]. Although these features have not yet been implemented in this system, they are, in principle, easily integrated as well.

Finally, fast grid construction algorithms such as those described by Ize et al. [9] accommodate the time-varying nature of particle-based simulation data in a straightforward manner. These datasets are quite large, containing many millions of particles across tens or hundreds of time steps. With our approach, the user can easily cycle through all of the time steps at run time because a grid can be rebuilt on-the-fly whenever necessary. This process accommodates the changing structure of the data as the simulation evolves, enabling interaction with millions of particles across the entire simulation. Moreover, because scientists can interact with the whole dataset, a clear understanding of the physical state of each particle, as well as its relationship to the full computational domain, can be achieved.

4 Coherent Grid Traversal for Particle Data

Our approach to particle visualization is inspired by the CGT algorithm. We use fast packet-based ray tracing and multi-level grids to achieve interactive performance for large, time-varying particle datasets. Frustum based traversal achieves lower per-ray cost by amortizing traversal operations over multiple rays in a packet, and the algorithm is well-suited to SIMD implementation. Further, advanced visualization features such as soft shadows are integrated easily because these features can be implemented naturally in a ray tracing framework. We thus extend the original CGT algorithm to efficiently and effectively visualize large, time-varying particle datasets by exploiting the properties of particle-based simulation data to improve performance and reduce storage requirements.

4.1 The Sphere-Center Method

The coherent grid traversal algorithm discussed in Section 2 can be optimized for glyph-based particle visualization by leveraging the fact that all primitives are spheres. Typically, the radii of particles within these datasets are of roughly equal size or fall within a well-defined range. As a result, several observations permit optimizations over and above those employed by the original CGT algorithm. First, a sphere S with center C and radius r is symmetric, so determining whether S overlaps a frustum F is analogous to testing whether C is in the r-neighborhood of F. In particular, we can test whether the distance from C to any of the bounding planes of F is less than r. Second, testing whether the distance from C to the planes of F is less than r is the same as testing whether C is inside another frustum F_r that has been enlarged by r. Thus, if we traverse the grid using an enlarged frustum, we only need to intersect those spheres whose center lies inside that frustum, and therefore only have to store each sphere at exactly one location: the cell in which its center is located.

Using the enlarged frustum F_r for traversal requires a priori knowledge of r, a value that (potentially) varies with each sphere. However, for our application, the radii are either uniform or lie within some small range, so the maximum radius r_{max} across all particles can be used to generate the enlarged frustum. We call this approach the *sphere-center* method.

Constructing the Enlarged Frustum. The enlarged frustum F_r is determined as follows. Consider a bounding plane defined by $u = u_0 + kdU$. We require the distance, s, to add to u_0 such that the shifted plane includes the centers of all potentially intersecting spheres with radii less than or equal to r_{max}.

Without loss of generality, we reduce the problem to two dimensions, as in Figure 3, so that the direction vector of the normal to the bounding plane is $\langle -dU, 1\rangle$ and the vector corresponding to the shift distance is $\langle 0, s\rangle$. By scalar projection, we see that:

$$r_{max} = \frac{\langle -dU, 1\rangle \cdot \langle 0, s\rangle}{|\langle -dU, 1\rangle|}.$$

Algebraically expanding and solving this equation for s gives the required distance, $s = r_{max}\sqrt{1+dU^2}$. To properly account for potential intersections, the distance s must be added to u_0 for the maximum bounding plane and subtracted for the minimum plane. Thus, the shifted frustum can be computed in just five SIMD operations: three additions, two multiplies, and a single square root. Finally, the near and far planes of the frustum must be shifted by r_{max} in the $-\vec{K}$ and $\vec{K}$ directions, respectively.

Discussion. The sphere-center approach alleviates many of the problems traditionally associated with grids. Discretization implies that some primitives will overlap multiple grids cells, and this behavior can lead to many problems.

First, data must be duplicated in each cell the primitive overlaps or, alternatively, references to the primitive data must be stored in these cells. The former approach can lead to a significant amount of redundant data, while the latter approach implies that spatially local primitives do not necessarily exhibit locality of reference; that is, primitives close to each other in the three-dimensional space of the scene may be separated by many bytes in address space. Although a preprocessing phase can be used to place spatially local primitives nearby in address space as well [4], this approach is not well-suited to acceleration structures that are built on-the-fly during rendering.

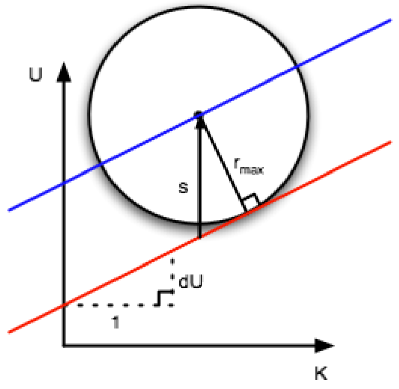


Figure 3: *Shifting the bounding planes.* We observe that testing whether a sphere of radius r intersects a bounding frustum (red) is equivalent to testing whether a frustum enlarged by r_{max} (blue) contains the center of the sphere.

Second, primitives that overlap many cells are problematic because rays will perform redundant intersections with these primitives as they traverse the grid. The original CGT algorithm uses a mailbox structure [11] to avoid these redundant intersections. Using mailboxes, primitives are tagged with a unique ray (packet) identifier, or the ray (packet) stores the identifiers of the last n primitives it has encountered. Intersection tests can be skipped when these identifiers match. Mailboxes do not typically improve performance for grids when intersection tests are inexpensive, as in the case of triangles or spheres, and it may actually reduce performance if avoiding primitive intersections does not outweigh the cost of querying and updating the mailbox. However, in a frustum based traversal scheme like CGT, many more redundant intersection tests are potentially required, and the overhead of managing the mailbox becomes less significant because this cost is amortized over multiple rays in a packet. Nevertheless, mailboxes require additional operations and storage, regardless of the effects of amortization.

The sphere-center method alleviates all of these problems. Data duplication is never required because the primitives are stored directly in the grid and the center of each sphere is guaranteed to lie in exactly one cell. As a consequence, locality of reference for spatially local primitives is improved without an explicit sorting or reorganization process. The sphere-center method also obviates the need for mailboxes: spheres are stored in exactly one grid cell, and will be intersected no more than once during traversal.

Some issues with the sphere-center method deserve mention. First, the maximum radius r_{max} is required to build the grid correctly and to adjust the bounding frustum during traversal. This value is determined once during initialization and then cached for later use during subsequent build and traversal steps. Second, adjusting the bounding frustum during traversal requires additional operations. However, using the SIMD instruction sets of modern CPUs, we can reduce the required operations to just five, which leads to efficient shifting. Finally, the improvement in performance is heavily dependent on the nature of the data itself. The greatest improvements in performance are noted when the spheres exhibit nearly uniform radii. However, if the radii span a wide range of values (for example, several orders of magnitude), the benefits gained by storing each sphere in exactly one grid cell may be outweighed by intersecting many more spheres than would otherwise be required. Frustum culling, discussed in Section 4.3, typically mitigates this behavior quite effectively.

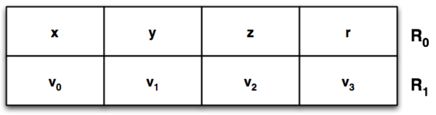
4.2 Grid Organization and Construction

The organization and design of our multi-level grid follows that of a typical hierarchical structure. Primitives are stored at the finest level of the grid, the resolution of which is determined such that the number of cells is a multiple of the total number of particles N, denoted by λ. Cubically shaped cells minimize surface area with respect to volume, and thus reduce the expected cost of traversal, so the resolution of the grid is given by:

$$N_x = d_x\sqrt[3]{\frac{\lambda N}{V}}, N_y = d_y\sqrt[3]{\frac{\lambda N}{V}}, N_z = d_z\sqrt[3]{\frac{\lambda N}{V}},$$

where $\vec{d}$ is the diagonal and V the volume of the grid.

Once the grid resolution has been determined, the data associated with each particle are inserted directly into the appropriate grid cells. To facilitate efficient insertion, culling, and ray/sphere intersection tests, these values are stored in two consecutive 16-byte aligned SIMD registers, as illustrated in Figure 4. The position and radius are stored in the first register, while up to four scalar properties from the simulation (v_0, v_1, v_2, and v_3) are stored in the second.



x	y	z	r	R_0
v_0	v_1	v_2	v_3	R_1

Figure 4: *Data layout for the modified CGT algorithm.* The data associated with each sphere are stored in two consecutive 16-byte aligned SIMD registers. This layout facilitates efficient insertion, culling, and ray/sphere intersection tests by leveraging the SIMD extensions on modern CPUs.

To facilitate a more efficient traversal, the grid is organized hierarchically. Hierarchical grids typically divide densely populated regions of space more finely than empty regions. There are several ways to accomplish this task [3, 10, 12, 17], and we leverage the macrocell hierarchy described by Parker et al. [17]. Each level in this hierarchy imposes a coarser grid over the previous level, and each macrocell corresponds to an $M \times M \times M$ block of cells in the underlying level. We use a simple two-level hierarchy: one level of macrocells imposed on top of the actual grid.

To support parameter range culling, the macrocell hierarchy differs from the one used in the original CGT algorithm and more closely resembles one used in interactive volume visualization applications [17]. In particular, a macrocell must store the minimum and maximum values of each data variable across all of the particles it contains, rather than a simple Boolean flag indicating an empty or non-empty condition. The data for an individual particle consists of its position, radius, and up to four additional properties (see Figure 4), requiring that each macrocell store the minimum and maximum value for eight parameters.

We use the sort-middle construction algorithm described by Ize et al. [9] to quickly rebuild the grid in each frame (if necessary). In this approach, the construction-related tasks corresponding to disjoint sections of the grid are statically distributed among all of the threads in the system. The sort-middle insertion essentially performs a coarse parallel bucket sort of the particles by their cell locations, and each thread inserts the particles in its set of buckets into the appropriate grid cells. The regions corresponding to each bucket are disjoint, so each thread inserts its particles into different parts of the grid. Write conflicts are thus avoided, and mutexes or other synchronization primitives are not necessary.

4.3 Additional Modifications

Mailboxes and fast SIMD frustum culling are critical components of the original CGT algorithm. Mailboxes prevent redundant intersections with a primitive that spans multiple grid cells, while culling operations often allow constant cost rejection of primitives. In addition, these operations are typically much faster than ray/primitive intersection tests, the cost of which is linear in the number of rays in each packet. Although the sphere-center method prevents redundant intersection tests and alleviates the need for mailboxes, two modifications based on frustum culling further improve performance and add flexibility to the data analysis process. We also add the ability to render images with soft shadows to enhance the perception of complex particle datasets.

SIMD Sphere/Frustum Culling. There are two sources of potentially unnecessary intersection tests with our approach. First, a sphere may lie within a cell through which the enlarged frustum passes, but the sphere does not overlap either the enlarged nor the original frustum. Second, the radius of a given sphere may be smaller than the maximum radius r_{max} used to compute the enlarged frustum, again implying overlap when there is actually no overlap. These situations are illustrated in Figure 5. We use frustum culling to efficiently reject non-overlapping spheres in the first case.

The original CGT algorithm employs SIMD shaft culling [5] to prevent unnecessary intersection tests by quickly discarding triangles that lie outside the current bounding frustum. In this case, triangles are culled if all four corner rays of the current frustum miss the triangle on the same edge.

Unfortunately, the SIMD shaft culling technique relies on primitives that posses planar edges, a property which spheres do not exhibit; as a result, this fast culling technique is not appropriate for our application. However, a much simpler test can be used to quickly cull spheres and avoid unnecessary intersection tests: if the distance from the center of a given sphere to any of the planes of the bounding frustum is greater than the radius of the sphere, the rays bounded by the frustum cannot intersect the sphere.

Our current implementation of this test uses the shifted bounding planes that are a by-product of the sphere-center method. If the center of the sphere is not contained by the enlarged frustum F_r, then none of the rays in the packet bounded by the original frustum can intersect the sphere and it can be culled. Using the enlarged frustum may seem superfluous because exactly that frustum was used to locate the cells with spheres that are potentially intersected by the rays; however, spheres that do not actually overlap F_r may be included because of discretization artifacts. In this case, an intersection test is required only if the center of the sphere lies within the enlarged bounding frustum. We note, however, that this approach does not cull spheres with radii less than r_{max} that are contained by F_r, but that do not overlap the original frustum (see Figure 5).

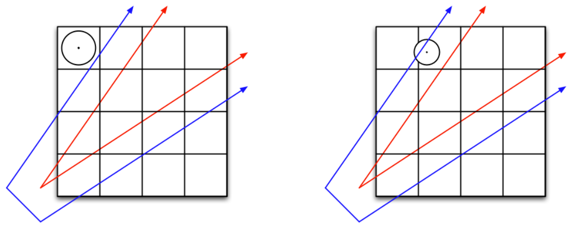


Figure 5: *Avoiding unnecessary ray/sphere intersection tests.* A sphere may lie within a cell through which the enlarged frustum passes, but the sphere does not overlap either the original or the enlarged frustum (left). SIMD sphere/frustum culling detects this situation and discards the spheres. A sphere whose radius is less than r_{max} may also lie in a cell through which the enlarged frustum passes, but the sphere may not overlap the original frustum (right). Our current culling algorithm does not handle this case, however.

Parameter Range Culling. In addition to its position and radius, up to four values representing properties from the simulation can be stored with each particle. To gain additional insight into the behavior of a simulation, investigators may isolate particles with parameters that take on a particular value or that lie within some range of values, as shown in Figure 6. We cull particles whose range of values do not overlap the currently valid range, thereby avoiding unnecessary intersection tests.

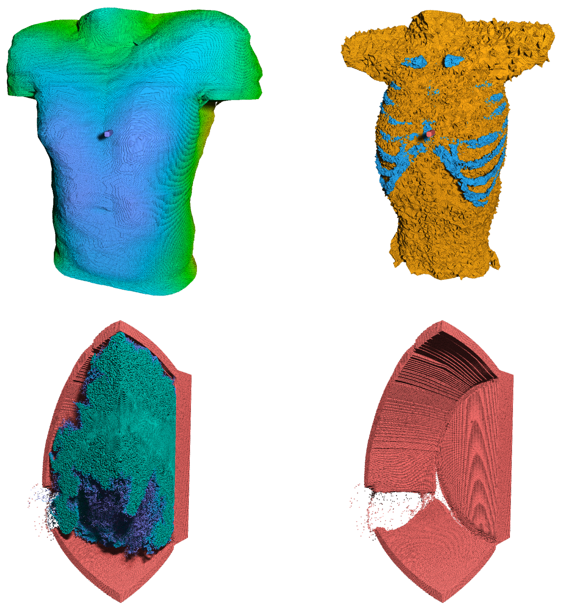
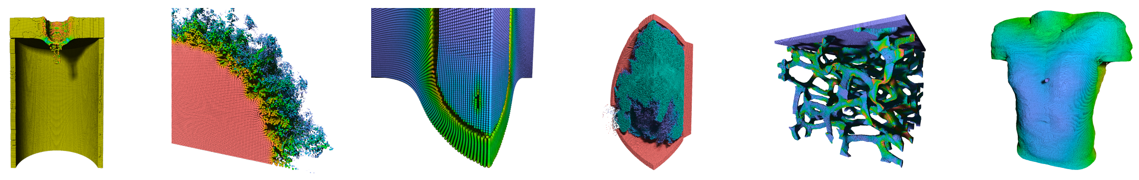


Figure 6: *Parameter range culling with particle datasets.* Using parameter range culling, particles representing the bone and internal tissues within the BulletTorso dataset (top) and only those representing the alloy container in the Thunder dataset (bottom) have been isolated. Parameter range culling puts the range of valid parameter values used during visualization under the full control of the user at run time, and these values can be changed interactively.

First, parameter range culling is applied to large groups of particles via the macrocell hierarchy. In this case, macrocells no longer store a simple Boolean flag indicating the empty/non-empty condition, but rather they store the minimum and maximum values for each data variable across all of the particles they contain. These extrema are then used during traversal to determine whether or not any spheres within a macrocell will potentially produce a valid intersection. If the macrocell range does not overlap the user-specified range, it can be skipped and all of the particles it contains are culled. For culling particles based on eight parameters, this check requires only eight SIMD operations: four comparisons, two Boolean *and* operations, and two masking operations. The cost of these operations becomes trivial when amortized over the number of particles contained within a typical macrocell, and is significantly less than the ray/sphere intersection tests that would otherwise be required for each of the particles within a macrocell.

When the values of at least one particle lie within the currently valid range, the macrocell cannot be skipped even though many of the particles may not be in range. To guarantee correctness, parameter range culling must also be applied at the level of individual particles. We ensure that each particle lies within the currently valid range before actually performing the intersection test. If any of the values lie outside the user-specified range, the intersection test can be skipped and the particle is culled. As with macrocell culling, this process requires eight SIMD operations per sphere: four comparisons, two Boolean *and* operations, and two masking operations.

Soft Shadows. As discussed in Section 3, soft shadows from area light sources provide important visual cues about the relative position of objects in complex datasets. Soft shadows are preferable to hard shadows because the smooth transition from shadowed to unshadowed regions is less likely to be misinterpreted as a discontinuity in the underlying data. Although shadows and other global effects are difficult to implement in systems based on rasterization, these effects are easily integrated into our approach because it is based on ray tracing.



	Cylinder	**JP8**	**Bullet**	**Thunder**	**Foam**	**BulletTorso**
# particles	212980	815345	2.1 M	2.8 M	7.2 M	34.9 M
Data size	6.50 MB	21.77 MB	47.75 MB	86.33 MB	136.52 MB	1.04 GB
Frame rate	100.20 fps	99.88 fps	126.93 fps	40.86 fps	14.34 fps	18.31 fps

Table 1: *Particle datasets used to evaluate the modified CGT 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 our modified CGT algorithm using the viewpoints and time steps shown above, but the system can render the full time-varying datasets as well. The frame rates reported in this table were achieved by rendering 1024×1024 images using 16 threads of the test machine, 8×8 ray packets, and simple Lambertian shading. (Performance with soft shadows is reported below.)

In particular, packets of coherent shadow rays can be generated by connecting the hit point corresponding to a given primary ray with some number of samples on an area light source [26]. By constructing shadows rays in this manner, secondary ray packets share a common origin and can traverse a grid using the modified CGT algorithm in a manner identical to that used for primary ray packets. Using this approach, both the number of shadow rays and the size of the light source can be interactively controlled by the user and permits performance-for-quality trade-offs.

5 Results

We evaluate the performance of the modified CGT algorithm using several particle datasets of varying sizes and complexity with a working implementation. The pertinent characteristics of these datasets and the viewpoints used during testing are given in Table 1. We first discuss the impact of the various parameters and optimizations in the modified CGT algorithm, and then compare the performance of our approach with other state-of-the-art particle visualization systems. Unless stated otherwise, the results were gathered by rendering 1024×1024 images using a 16 core Opteron machine with 2.4 GHz processors and 64 GB of physical memory.

Impact of Grid and Packet Resolution. Like the original CGT algorithm, the performance of our approach is governed by four parameters: grid resolution, macrocell resolution, ray packet size, and image resolution. As described in Section 4.2, the grid resolution is determined using λ, a parameter that relates the number of cells in the grid to the total number of particles. Unlike the original CGT algorithm, in which most scenes were largely insensitive to the value of λ, the performance of the modified algorithm varies widely with different values of λ, which is a result of the extremely large number of particles in the test datasets. After testing several values in the range $[0.2, 5]$, it is clear that $\lambda = 1$ provides the best performance for all of the datasets we use. Further testing shows that a macrocell resolution of $6 \times 6 \times 6$ yields reasonable performance for these datasets. Though tuning the parameters for each dataset may yield slight performance gains, we use these parameter values for all of our tests.

Ray packet size has a significant impact on interactive performance. For a given packet size, the cost of a traversal step is constant while the cost of intersecting the cells in a given slice increases with the number of cells the frustum overlaps. The frustum bounding a small ray packet will overlap fewer cells than that of a larger packet, but large packets amortize the costs over more rays, so there is an obvious trade-off between packet size and performance.

Table 2 gives the frame rates achieved when rendering each of the test datasets with a single rendering thread using various packet sizes. The number of particles in these datasets ranges from a few hundred thousand to tens of millions, so the resulting grids are often several hundred cells in each dimension. As a result, 4×4 and 8×8 packets typically provide the best performance by amortizing the traversal cost over a larger number of rays. Unless stated otherwise, we use 8×8 ray packets for the remainder of our tests.

Dataset	**2×2**	**4×4**	**8×8**	**16×16**
Cylinder	3.99	6.94	**7.29**	4.32
JP8	2.78	5.92	**8.10**	5.93
Bullet	4.26	8.04	**9.42**	6.41
Thunder	2.96	**3.80**	2.95	1.32
Foam	1.37	**1.65**	0.98	0.25
BulletTorso	1.42	**1.83**	1.32	0.43

Table 2: *Impact of primary ray packet size.* Frame rates achieved using a single thread on the test machine for various packet sizes. In general, 4×4 and 8×8 packets provide the best performance for the datasets tested.

Impact of Image Resolution. In addition to grid resolution (and thus the number of particles), the optimal packet size is also influenced by the image resolution: high resolution images result in higher ray density and permit larger packet sizes. As noted, a resolution of 1024×1024 pixels is used as the default value for these experiments, which is suitable for current displays. However, the aliasing problem, which is particularly acute for the large numbers of particles and complex geometries typical of particle-based simulation data, pushes the demand for oversampling—or, equivalently, larger image resolutions—forward.

Ray tracing cost is typically linear in the number of pixels, but because higher resolution images allow larger ray packets, the modified CGT algorithm scales sublinearly with image resolution, as demonstrated by the data in Table 3.

	1024^2		**2048^2**		
Dataset	**4×4**	**8×8**	**8×8**	**16×16**	**Ratio**
Cylinder	6.83	**7.25**	**2.95**	2.32	0.41
JP8	5.84	**8.06**	**3.04**	2.91	0.38
Bullet	8.01	**9.37**	**3.32**	2.97	0.35
Thunder	**3.56**	2.79	**1.74**	0.98	0.49
Foam	**1.57**	0.95	**0.74**	0.31	0.47
BulletTorso	**1.79**	1.30	**0.90**	0.46	0.50

Table 3: *Impact of image resolution.* Frame rates achieved using a single thread on the test machine for various packet sizes and image resolutions. The modified CGT algorithm scales sublinearly with image resolution.

Impact of the Sphere-Center Method. The sphere-center method introduced in Section 4.1 overcomes many problems associated with grids and improves interactive rendering performance. In this method, each particle is stored in exactly one grid cell so data is not duplicated, locality of reference is improved, and schemes to prevent redundant intersection tests become unnecessary. However, it is not immediately clear that traversing the enlarged frustum required by the sphere-center method would not simply cancel these benefits or actually degrade performance. As the data in Table 4 demonstrates, however, frame rates improve by a factor of 1.02–1.27 over a standard grid that stores references to the particle data in (possibly) many cells. The operations required to compute the enlarged frustum can be implemented very efficiently using the SIMD extensions of modern CPUs, so the cost of these additional operations is subsumed by the benefits of storing each primitive in exactly one cell.

Dataset	Standard	Sphere-Center	Speed-up
Cylinder	6.58	7.29	1.11
JP8	7.91	8.10	1.02
Bullet	9.21	9.42	1.02
Thunder	2.37	2.95	1.24
Foam	0.81	0.98	1.21
BulletTorso	1.04	1.32	1.27

Table 4: *Impact of the sphere-center method.* Frame rates achieved using a single thread for standard and sphere-center grids. Though originally designed for reducing the storage overhead and simplifying data access, the overall performance improves by a factor of 1.02–1.27.

This method also reduces the memory footprint of our application. Primitive data is stored directly in the finest level of the grid, and is neither duplicated nor referenced by pointers. For example, the BulletTorso dataset, which consists of nearly 35 million particles and consumes just over 1 GB of storage, results in a $420 \times 198 \times 432$ grid. The average particle overlaps 16.19 grid cells in this case, and a standard grid implementation that stores particle identifiers (as 4-byte integers) in each cell adds an additional 2.11 GB. However, using the sphere-center method, only 1.04 GB of storage is required: 32 bytes (8 data values $\times$ 4-byte floating point numbers) for each of 34.9 M spheres.

Similarly, the sphere-center method also improves grid construction times because potentially expensive primitive/cell or bounding box/cell overlap tests are no longer required. Spheres are placed in exactly one cell by simply truncating the floating point values expressing their centers in the grid coordinate space to integers. This conversion requires only one SIMD operation on modern CPUs. As an example, total grid construction time using a single thread on the test machine for the JP8 dataset improves from 1407.27 *ms* to 510.96 *ms*, which represents a factor of about 2.75 overall. In particular, the time required by the insertion phase is 797.71 *ms* for the standard grid, but it is reduced by a factor of 3.07 to 260.13 *ms* using the sphere-center method.

Impact of Frustum and Parameter Range Culling. Efficient frustum culling plays an important role in the original CGT algorithm, and the same holds true for our modified algorithm. Uniform grids do not adapt to the local variations in primitive density as well as structures like kd-trees or BVHs. As a result, more primitive intersection tests are required during traversal of a grid than for other structures. Frustum culling cancels this behavior and reduces the number of ray/primitive intersection tests actually performed, as demonstrated by the results in Table 5. The efficient SIMD sphere/frustum culling procedure described in Section 4.3 reduces the number of ray/sphere intersection tests performed to 38–81% of the total potential tests required for the test datasets, improving interactive performance by a factor of 1.17–1.68 as shown in Table 6.

As described in Section 4.3, parameter range culling is applied at the level of both the macrocells and the individual particles. The results in Table 7, which correspond to the images in Figure 6, indicate that this ability adds some additional overhead, but efficient SIMD implementation of the range checking decreases performance by only a factor of 1.38–2.28 over preprocessed versions of the data. Some of this performance difference can be attributed to the slight difference in the grid bounds (with smaller bounds leading to fewer packet traversals), but the 16 SIMD operations implementing macrocell and particle range checking also adds some computational overhead. Nevertheless, this feature provides additional flexibility during the data analysis process, a benefit which clearly outweighs the relative impact on interactive performance.

Dataset	Potential tests	# skipped	Culled
Cylinder	219470	135722	61.84%
JP8	284990	231832	81.34%
Bullet	161938	108858	67.22%
Thunder	593191	346076	58.34%
Foam	2120452	1179271	55.61%
BulletTorso	925835	355538	38.40%

Table 5: *Culling statistics for SIMD sphere/frustum culling.* Number of potential ray/sphere intersection tests and the number of tests skipped by frustum culling. Efficient SIMD frustum culling significantly reduces the number of ray/sphere intersection tests required during grid traversal.

Dataset	No culling	Culling	Speed-up
Cylinder	71.04	99.16	1.40
JP8	59.13	99.79	1.68
Bullet	87.20	123.37	1.41
Thunder	29.22	40.78	1.40
Foam	9.64	14.21	1.47
BulletTorso	15.69	18.31	1.17

Table 6: *Impact of SIMD sphere/frustum culling.* Frame rates achieved using 16 threads on the test machine with and without frustum culling. Interactive performance improves by a factor of 1.17–1.68 for the test datasets.

Dataset	% total	PR cull (no mcells)	PR cull	Pre-crop
Thunder	43.00	14.23	43.06	59.55
BulletTorso	34.17	3.17	11.18	25.49

Table 7: *Impact of parameter range culling.* Frame rates achieved using 16 threads on the test machine for on-the-fly parameter range culling (with and without macrocell culling) and preprocessed data. Parameter range culling adds some additional overhead, but interactive performance degrades only slightly when compared to preprocessed datasets composed of the same particles.

Impact of Soft Shadows. To this point, we have only considered simple ray casting and local shading; non-local effects such as shadows have not been considered. However, using the approach described in Section 4.3, packets of coherent shadow rays can be generated for each primary ray and then traced through the scene using the modified CGT algorithm.

Although interactive performance with soft shadows depends heavily on the coherence exhibited by secondary ray packets, the impact is sublinear in the number of rays traced, as demonstrated by the data in Table 8. Interactive performance varies widely for the datasets and lighting configurations tested, with the impact ranging from a factor of 2.42 (for the fastest 2×2 packets) to as much as 19.35 (for the slowest 8×8 packets).

The flexibility of an interactive visualization environment puts these parameters under the full control of the user at run time, allowing trade-offs between image quality and interactive performance. For example, Figure 7 shows the results of using 2×2, 4×4, and 8×8 shadow rays per primary ray. Quality can be traded for performance by simply using fewer shadow rays or disabling shadows altogether.

Comparison with Other Approaches. Finally, we compare the performance of our approach with two recent systems that represent the current state-of-the-art in interactive particle visualization.

Dataset	No shadows	2×2	4×4	8×8
Cylinder	100.20	14.91	10.47	5.34
JP8	99.88	16.31	12.21	6.65
Bullet	126.93	19.55	13.69	6.56
Thunder	40.86	14.07	10.74	6.45
Foam	14.34	4.38	2.51	1.17
BulletTorso	18.30	7.39	6.09	3.82

Table 8: *Impact of soft shadows.* Frame rates achieved using 16 threads on the test machine for various soft shadow settings. In these tests, the light source area is rather large, at 5% of the area of the bounding box of the scene.

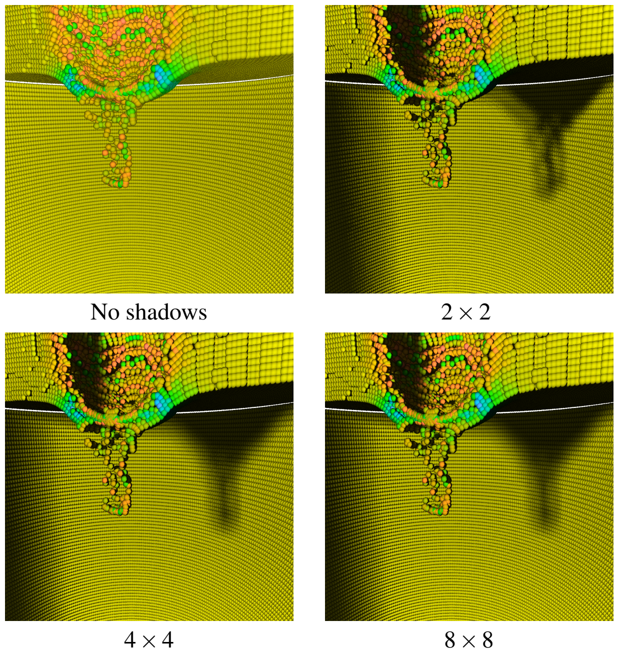


Figure 7: *Rendering with soft shadows.* Soft shadows from area light sources provide important visual cues about the relative position of objects in complex datasets. Shadows and other global effects are easily integrated into an approach based on ray tracing such as the one described here.

The first is based on an optimized single ray grid traversal algorithm in the Real-Time Ray Tracer [16], while the second leverages programmable graphics hardware and software-based acceleration techniques [6].

The results in Table 9 show that our modified CGT algorithm compares favorably with these systems for the test datasets. The benefits of packet-based traversal become evident when compared to single ray traversal: interactive performance improves by a factor of 1.35–14.48 for the test datasets. Though some of the improvement results from our use of SIMD extensions that are not easily employed in single ray scheme, such an implementation usually provides an improvement of only a factor of 2–3; the remainder is a result of the cost amortization and algorithmic improvements inherent to a packet-based traversal method.

Surprisingly, our approach also outperforms the system based on programmable graphics hardware. This system uses view-aligned, textured billboards to represent each particle. Vertex and fragment programs manipulate this data to provide a high-quality representation of each particle that is consistent with the results of an approach based on ray tracing. In addition, software-based acceleration techniques (including basic frustum culling and more sophisticated occlusion culling algorithms) are used to reduce the rendering workload in each frame. Nevertheless, and despite the fact that our test machine actually provides less raw FLOPS than the Nvidia GeForce 7800 GT used for testing, our approach outperforms this system by a factor of about 5 to almost 50 for the datasets tested.

Dataset	Our CGT	RTRT	GPU-based
		Parker et al. [16]	Gribble et al. [6]
Cylinder	100.20	12.60	5.78
JP8	99.88	6.90	17.40
Bullet	126.93	11.90	2.56
Thunder	40.86	14.90	8.10
Foam	14.33	6.40	2.04
BulletTorso	18.31	13.50	1.56

Table 9: *Comparison of visualization methods.* Frame rates achieved using our modified CGT algorithm and two state-of-the-art interactive particle visualization systems. The benefits of packet-based traversal become evident when compared to single ray traversal, and our approach also significantly outperforms an approach leveraging programmable graphics hardware.

6 Conclusions and Future Work

We have presented an approach to particle-based data visualization based on optimizations to the coherent grid traversal algorithm. Our modified algorithm employs fast ray tracing methods for multi-level grids, including ray packets, frustum based traversal, frustum culling, and SIMD extensions. We have also introduced the sphere-center method, which exploits the properties of particle-based simulation data to improve the performance of coherent grid traversal and reduce storage requirements. The sphere-center method attacks some classic problems associated with grids, namely duplicate data or no locality of reference, and redundant ray/primitive intersection tests. In addition, the rebuild process is made more efficient with this method by replacing primitive or bounding box overlap tests with a simple float-to-int truncation. Optimizations based on efficient sphere/frustum culling further improve the interactive performance of the modified CGT algorithm.

We have evaluated the performance of our approach using a system with sixteen 2.4 GHz Opteron cores (8 Opteron 880 dual-core CPUs). The theoretical peak available on this machine is less than 155 GFLOPS, which is an order of magnitude less than the terascale performance of, for example, the ATI X1900 graphics processing unit [19]. While it is only a matter of time before compute power of this magnitude is available for ray tracing, the evaluation of our algorithm shows highly interactive frame rates on reasonably priced multi-core platforms (a system with compute power similar to the test machine would cost less than $35,000 at the time of this writing). In addition, our approach compares favorably with two recent systems that represent the current state-of-the-art in particle visualization. A previous approach based on interactive ray tracing [2] require tightly-coupled supercomputing platforms, and although this approach satisfies the requirements of effective particle visualization, the associated hardware costs are prohibitive and impede accessibility. Systems using programmable graphics hardware [6] also offer a way to visualize large, time-varying particle datasets at interactive rates. This hardware is widely available, and a desktop system so equipped is considerably less expensive (roughly a factor of 7) than the system used to evaluate our algorithm. However, GPU-based approaches are not easily extended to include visualization features like soft shadows or advanced shading models, while an algorithm based on ray tracing can be extended to include these features naturally. Already we achieve reasonably interactive performance with a naive implementation of soft shadows, and advanced shading models such as ambient occlusion or physically based diffuse interreflection will become feasible with continued improvements in both algorithmic design and CPU performance. Moreover, as multi-core systems become more prevalent, the price-performance ratio of a particle visualization system based on interactive ray tracing will only improve.

Several areas require further attention. First, techniques similar to the sphere-center method can be applied to other types of primitives such as triangles, and exploring these methods is of interest. In addition, the current implementation of soft shadows treats secondary rays in a manner identical to primary rays. Additional improvements in performance may result from optimizations specific to secondary ray packets. Accelerating performance of secondary rays is also important if the visual cues from advanced shading models like ambient occlusion and physically based diffuse interreflection are to be used during interactive rendering. Exploring efficient methods to include these effects is of particular interest. Finally, multi-modal visualization of particle and volumetric data, such as a container (particle-based simulation) in a pool fire (computational fluid dynamics simulation), would also be useful. Efficient techniques for packet-based volume rendering are required to combine this visualization modality with the particle visualization method we have described.

Acknowledgments

We are indebted to Jerry Seidler and Erin Miller, both of the University of Washington, and to Scott Bardenhagen, of the Los Alamos National Laboratory, for providing the Foam dataset. This work was funded by the DOE ASC program.

References

[1] S. G. Bardenhagen, J. U. Brackbill, and D. Sulsky. The Material-Point Method for Granular Mechanics. *Comput. Methods Appl. Mech. Engrg.*, 187:529–541, 2000.

[2] J. Bigler, J. Guilkey, C. Gribble, S. Parker, and C. Hansen. A Case Study: Visualizing Material Point Method Data. In *Proceedings of the Eurographics/IEEE Symposium on Visualization*, pages 299–306, May 2006.

[3] F. Cazals, G. Drettakis, and C. Puech. Filtering, Clustering and Hierarchy Construction: a New Solution for Ray Tracing Very Complex Environments. *Computer Graphics Forum (Proceedings of Eurographics '95)*, 14(3), 1995.

[4] D. E. DeMarle, C. Gribble, and S. Parker. Memory-Savvy Distributed Interactive Ray Tracing. In *Eurographics Symposium on Parallel Graphics and Visualization*, pages 93–100, 2004.

[5] K. Dmitriev, V. Havran, and H.-P. Seidel. Faster Ray Tracing with SIMD Shaft Culling. Technical Report MPI-I-2004-4-006, Max-Planck Institut für Informatik, 2004.

[6] C. P. Gribble, J. E. Guilkey, A. J. Stephens, and S. G. Parker. Visualizing Material Point Method Datasets on the Desktop. In *British HCI 2006 Workshop on Combining Visualization and Interaction to Facilitate Scientific Exploration and Discovery*, to appear, 2006.

[7] C. P. Gribble and S. G. Parker. Enhancing Interactive Particle Visualization with Advanced Shading Models. In *Proceedings of the Third Symposium on Applied Perception in Graphics and Visualization*, to appear, 2006.

[8] J. E. Guilkey, J. A. Hoying, and J. A. Weiss. Computational Modeling of Multicellular Constructs with the Material Point Method. *Journal of Biomechanics*, to appear, 2006.

[9] T. Ize, I. Wald, C. Robertson, and S. G. Parker. An Evaluation of Parallel Grid Construction for Ray Tracing Dynamic Scenes. Technical Report UUSCI-2006-025, SCI Institute, University of Utah, 2006.

[10] D. Jevans and B. Wyvill. Adaptive Voxel Subdivision for Ray Tracing. In *Proceedings of Graphics Interface '89*, pages 164–172, 1989.

[11] D. Kirk and J. Arvo. Improved Ray Tagging for Voxel-Based Ray Tracing. In *Graphics Gems II*, pages 264–266. Academic Press, 1991.

[12] K. S. Klimaszewski and T. W. Sederberg. Faster Ray Tracing using Adaptive Grids. *IEEE Computer Graphics and Applications*, 17(1):42–51, January/February 1997.

[13] M. Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.

[14] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *International Conference on Computer Graphics and Interactive Techniques*, pages 163–169, 1987.

[15] P. Mamassian, D. C. Knill, and D. Kersten. The perception of cast shadows. *Trends in Cognitive Sciences*, 2(8):288–295, 1998.

[16] S Parker, W. Martin, P.-P. J. Sloan, P. Shirley, B. Smits, and C. Hansen. Interactive Ray Tracing. In *Symposium on Interactive 3D Graphics*, pages 119–126, 1999.

[17] S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, C. Hansen, and P. Shirley. Interactive Ray Tracing for Volume Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):238–250, 1999.

[18] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive Ray Tracing for Isosurface Rendering. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of IEEE Visualization 1998*, pages 233–238, 1998.

[19] D. E. Polkowski. ATI's Radeon X1900 Heats Up With 48 Shader Units. http://www.tomshardware.com/2006/01/24/, January 2006.

[20] A. Reshetov, A. Soupikov, and J. Hurley. Multi-Level Ray Tracing Algorithm. *ACM Transacions on Graphics*, 24(3):1176–1185, July 2005.

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

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

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

[24] S. Ullman. *The Interpretation of Visual Motion*. MIT Press, Cambridge, Massachusetts, 1979.

[25] H. von Helmholtz. *Handbook of Physiological Optics*. Optical Society of America, New York, 1925.

[26] I. Wald, C. Benthin, M. Wagner, and P. Slusallek. Interactive Rendering with Coherent Ray Tracing. *Computer Graphics Forum*, 20(3):153–164, September 2001.

[27] I. Wald, S. Boulos, and P. Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics*, to appear, 2006.

[28] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. G. Parker. Ray Tracing Animated Scenes using Coherent Grid Traversal. *ACM Transactions on Graphics*, to appear, 2006.

[29] L. R. Wanger, J. A. Ferwerda, and D. P. Greenberg. Perceiving Spatial Relationships in Computer-Generated Images. *IEEE Computer Graphics and Applications*, 12(3):44–58, 1992.