Hybrid sort-first/sort-last rendering for dense material particle systems

Simon Latapie

CEA
DAM, DIF
F-91297 Arpajon, France

Laboratoire MAS
Ecole Centrale Paris
Grande Voie des Vignes
92290, Châtenay-Malabry, France

simon.latapie@gmail.com

ABSTRACT

This paper describes a solution designed for efficient visualization of large and dense sets of particles, typically generated by molecular dynamics simulations in materials science. This solution is based on a hybrid distributed sort-first/sort-last architecture, and meant to work on a generic commodity cluster feeding a tiled display. The package relies on VTK framework with various extensions to achieve statistical occlusion culling, smart data partitioning and GPU-accelerated rendering.

Keywords

Hybrid sort-first/sort-last rendering, Statistical culling, VTK, Ice-T, Particle rendering, Dense particle system

1. INTRODUCTION

Materials science increasingly uses numerical simulations at different scales of space and time to better understand and predict the properties of matter. Molecular dynamics is one of the most widely used approaches in computational materials science. Thanks to the joint advances in parallel computing and in physics modeling, molecular dynamics can now be used to simulate systems with millions to hundreds of millions of particles[Stre 05]. We focus here on such simulations at nanoscopic to microscopic scales, which describe matter in dense states by large sets of particles.

Suitable and efficient visualization tools must be provided besides simulation codes in order to benefit from these very detailed computations, and especially interactive tools that help to explore complex 3D features such as blast waves, solidifications, dislocations, etc. We are going to describe how we built a distributed visualization tool to exploit a small graphics cluster and tiled display for almost interactive exploration of such datasets. The solution is quite standard since it relies on widely used software components, such as VTK[Schr 06], with various optimizations.

After reviewing related work which partly inspired us, we are going to describe the overall organization of the system, then give more details on some technical aspects of the algorithms we combined: culling by space partitioning and statistical occlusion, particle rendering and parallelization.

2. RELATED WORK

Only few existing solutions are specifically designed for visualization of global phenomena inside large particle sets on a tiled display.

Many systems are especially designed for biological molecular dynamics, like VMD[Hump 96] or Molekel[Fluk 00]. Some exhibit very advanced rendering techniques, via GPU programming, accelerating complex shape rendering like TexMol[Baja 04], or global illumination like QuteMol[Tari 06]. Such tools are generally optimized to represent domain specific features or sub-structures with non-spherical shapes, like ribbons, tubes, or molecular surfaces. These representations cannot be used in materials physics where there are no such apparent structures as proteins parts, or identified zones like in Terascale Particle Visualization[Ells 04].

Our application domain requires to focus on efficient raw rendering of particles, basically represented as colored opaque spheres. Opaque sphere representation is very important because on dense particles systems from materials science it can preserve graphical aspects of several structure properties, such as some surfaces granularities, which are lost with non-opaque, point-only

represented particles, or volume rendering solutions, such as in Liang et al.[Lian 05] solution.

Moreover, most of the aforementioned tools cannot be easily integrated in a distributed rendering architecture. Another example in another application domain is Kruger et al.[Krug 05] solution, a very efficient particle system rendering to visualize 3D flow fields. Such a solution takes advantage of new GPUs rendering capabilities, but as all data is stored in graphic card memory, scalability and possible extensions to a distributed architecture are compromised.

Very few solutions are scalable and designed to display raw real sphere representation of large sets of particles on large definition displays such as tiled displays. Atomsviewer[Shar 03] is one of them: it uses efficient optimizations for sort-first rendering of large sets of particles: Z-order data organization, octree space partitioning, probabilistic culling method. However, Atomsviewer has been adjusted to a specific hardware and display configuration (ImmersaDesk^{TM})[Shar 02b].

All these observations have lead us to work on the integration of culling methods and hardware-accelerated rendering in a generic distributed architecture.

3. GENERAL OVERVIEW

The main objective of our architecture is to provide new optimizations while re-using VTK/Ice-T[More 01]. Ice-T is a sort-last rendering solution for tiled displays, which has been proven[More 03] to be more efficient than generic solutions for tiled displays such as Chromium[Hump 02]. Sort-last rendering is scalable with respect to the size of the data, but the known bottleneck of such an approach is the network bandwidth. Ice-T brings improvements to usual sort-last rendering, such as an efficient distribution of images to be composed, or a floating viewport technique. Nethertheless, network bandwidth remains the bottleneck of such a method. Our strategy is to achieve distributed sort-first operations to increase spatial coherence of per process data to reduce network bandwidth usage, and lower the number of spheres to be displayed, to reduce the actual sphere rendering stage. This can be considered to be an attempt to transpose Samanta et al.[Sama 00] hybrid architecture to non-structured particle systems visualization.

Before data is to be displayed, we generate a space partitioning Kd-Tree, and we reorganize the dataset to gather all data attached to a same leaf of the tree. Each node of the tree contains its bounds, its data storage location, a link to its children if it has any, and a density factor, which is described in section 4.

The global architecture of our system, described in Figure 1, is meant to run on a cluster of N nodes, with P of them actually connected to a display with P physical or logical tiles. Each node runs a MPI process. Each process has a complete copy of the tree, but only part of the data, in memory. During a frame rendering, each process computes frustum and occlusion culling for its own part of the tree, then all processes merge their results to have a complete up-to-date tree. At that time all processes know which data is really to be displayed. Then they compute a balanced per process redistribution of the data, load missing data and unload useless data if necessary, and render them. For the sort-last part of the architecture, Ice-T library performs a distributed rendering, and displays the frame.

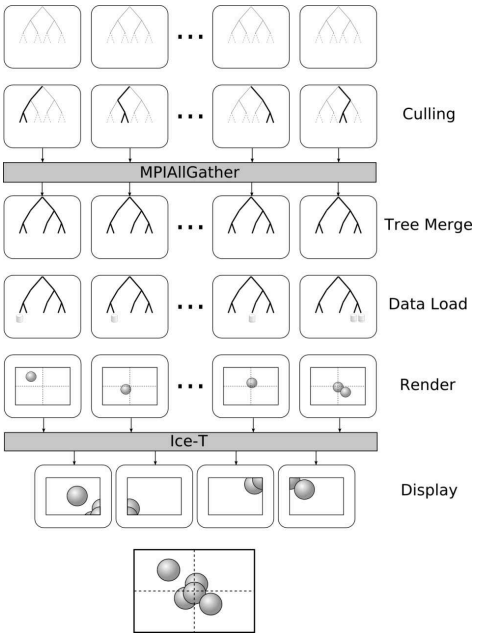


Figure 1: Global Architecture.

In the following sections, we are going to describe the tree for space decomposition and the culling algorithms we designed to organize datasets and optimize Ice-T rendering process. Then we will mention the GPU optimizations for the sphere rendering stage, the parallelization strategies, and finally we will highlight the implementation and a few results.

4. CULLING IN DENSE MATERIAL PARTICLE SYSTEMS

Although Atomsviewer is based on a specific architecture, some of the pre-rendering techniques it uses are very effective, such as the idea of probabilistic occlusion culling[Shar 02a]. This approach introduces a notion of density of particles in the cells of an octree, which is used to randomly drop part of the particles displayed.

Dense material systems are often composed of large groups of particles which are very close to each other. The probabilistic occlusion can be pre-computed, as our datasets are non-interactively generated. Atomsviewer's density computation, which is the volume

of particles divided by the volume of the cell, is not a very good factor to use for culling, because it assumes the distribution of particles is uniform in all cells.

In this section, we propose a complete pre-processing solution adapted to dense material systems visualization, which consists in a space partitioning algorithm, and a cell density computation. We also propose a culling algorithm which takes the pre-processing specificities into account.

Space partitioning

We use a Kd-Tree structure for space partitioning, which aims at separating dense space and empty space, in a very quick and simple way. We do not use the Kd-Tree VTK implementation, because of specific tree parsing methods and data order tests needs.

4.1.1 Empty space detection Kd-Tree

The algorithm described below is a simple way of generating a Kd-Tree for empty space detection.

As seen in Figure 2, for each tree node, we check particles positions for a given axis. If there is a left or right space between particles and node bounds, an empty space isolating split is done. Otherwise, a middle split is performed. Then the particles are sorted by comparing their axis position with the split position. Axis is alternatively x, y, then z.

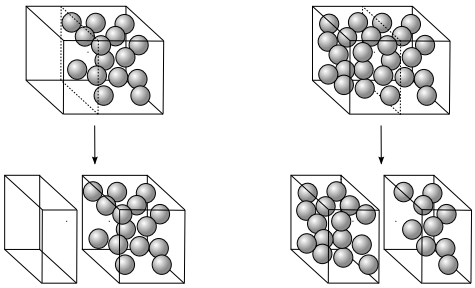


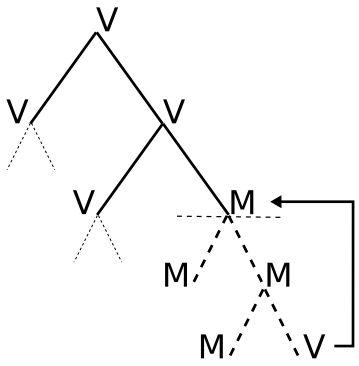
Figure 2: Kd-Tree generation: empty space partition

The complexity of the generation is $O(nln(n))$, where n is the number of particles, as the most consuming part is the particle sort, which is very much like a global quicksort, the split position being a pivot value. Optimum depth of the tree is very dependent on particles repartition in the scene, but empirical tests shows a 15 depth is a good overall value.

4.1.2 Kd-Tree optimization

The main goal of this space partition is to find dense cells which can occlude other cells. So we want the dense cells to be as large as possible. This is why we can optimize the Kd-Tree generation by moving up the effective splits, which are the empty space ones. Density computation is explained in section 4.2.

Figure 3 explains such an optimization: when a middle split is performed, a temporary tree branch is computed, until an empty space split is found, or maximum depth is reached. On the first case, the empty space split is applied on base node and the branch is computed again, otherwise the branch is kept as it is.



V node is an empty space splitting node (Void split)
M node is a middle splitting node

Figure 3: Kd-Tree optimization

Statistical occlusion culling

The following step in pre-processing is tree cells density computation. Density must describe the culling effect of a cell in relation to another one. We use a Monte-Carlo method to compute the probability for a ray to go through a space-partitioning cell containing spheres.

We launch N rays with random position and direction through the cell, and we check if the ray goes through any of the spheres or not (see Figure 4). Sphere radii are fixed attributes of the particles. It is a five degrees of freedom problem, and Monte-Carlo basic method provides a $1/\sqrt{N}$ error convergence. A typical number of casted rays for a 1% error on density for one cell is 100,000 casts.

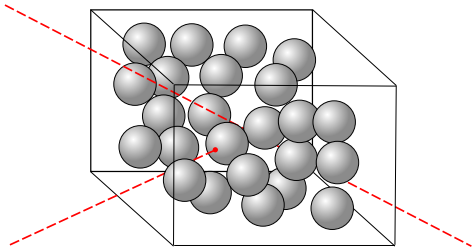


Figure 4: Monte-Carlo method for computing cell density.

This provides a 0 to 1 density factor, which represents the probability a ray has to go through the cell, and can be used as a trust percentage of this cell to occlude another. Each node of the tree has such a density factor.

Culling algorithms

We had to make a choice between the two most frequently used strategies for occlusion and frustum culling. The first one is silhouette comparison in viewport coordinates, like in [Coor 97]. The other one uses occlusion maps[Zhan 97]. We chose a strategy similar to the first one, because occlusion maps require a lot of GPU memory, which we would like to keep for the VBO cache system as described in section 5.2.

4.3.1 Silhouette computation

Occlusion culling is achieved by computing a silhouette of the occluding cell, and checking if a potentially occluded cell is inside the silhouette in viewport coordinates (cf Figure 5). Graham scan algorithm[Grah 72] gives us a y-sorted couple of point list which represents left side and right side of the silhouette.

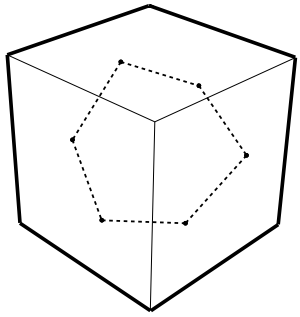


Figure 5: Block silhouette occluding another block

For each potentially occluded cell vertex, we find the left and right segments of the silhouette that share the same y-position as the top. Then we compare the x position of the top and the segments (see Figure6).

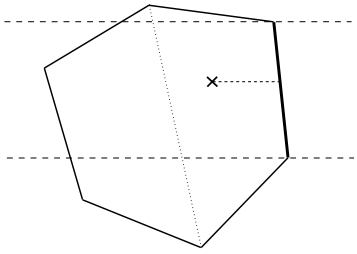


Figure 6: Silhouette occlusion check: X position compared to Y including segment

Depth check is done by comparing the z position of a vertex in relation to one or two planes. Planes are defined by triangles made by silhouette points close to the checked point, as seen in Figure 7. It is worth noting that this depth check is possible because we compare only disjoint cells.

4.3.2 Application to Kd-Tree

As seen in section 4.2, the Monte-Carlo pre-processing gives us a trust factor for cell opacity. If we set a threshold on this factor, we can consider some of the cells

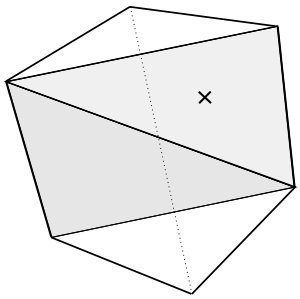


Figure 7: Depth check: triangles for planes definitions

are completely opaque, and then make a global culling comparison between tree nodes.

Each node has three possible states: *not occluded* (visible), *partially occluded*, and *occluded*. By default, all nodes are *not occluded* (visible). Leaves, as elementary undividable nodes, can only be tagged as *not occluded* (visible) or *occluded*.

As the tree is pre-computed, each node has a density attribute, and a maximum density of all the nodes beneath it. Then we recursively browse the tree from root to leaves, stopping as soon as a node with enough density is found.

For each occluding node, we perform the occlusion test, as seen in Figure 8: if the potentially occluded node (*PON*) contains no particle, or we already know it is occluded by another node, it is obviously occluded. If it is a child of the occluding node (*ON*), a test between the *ON*'s direct children and the *PON*. If the *ON* and the *PON* are the same node, and have children, they can potentially occlude a part of themselves, so we achieve a test between one of them and their children. If the *PON* is already partially occluded, we know that part of his children are already occluded, so we test his children. Otherwise, the silhouette of the *ON* is computed, and the occlusion is effective. Note that occlusion test between a node and one of is children can be meaningless because they can share bounds.

4.3.3 Frustum culling

Frustum culling is rather simple: the whole silhouette algorithm works in viewport coordinates. We only have to check if all cell coordinates are out of bounds per axis, i.e. if for any axis, cell coordinates are all either lower than -1 or higher than $+1$ in viewport coordinates.

5. PARTICLE RENDERING

In this section we describe some of the techniques we used to manage GPU memory and rendering.

Sphere rendering

We use GPU OpenGL shaders[Kess 06] programming to render particles as spheres. The big advantage of this technique is the per pixel precision of the rendering: thanks to the fragment shader program, all

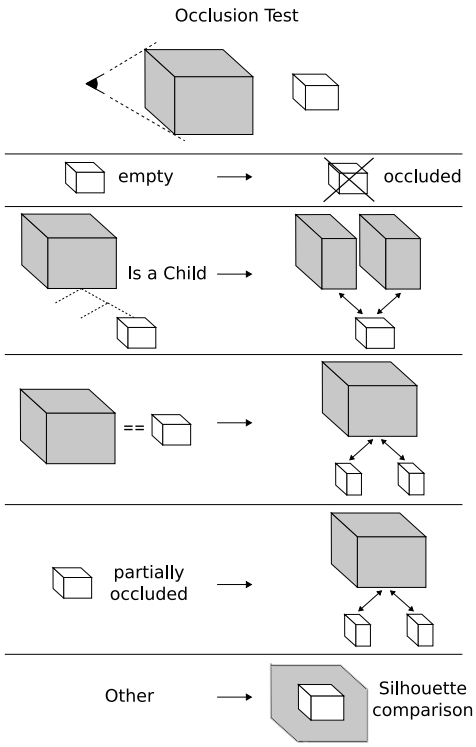


Figure 8: Occlusion Test

spheres are rendered pixel per pixel, and not with a group of vertices. This provides a direct level of detail feature, because rendering precision is only dependent on the number of pixels the sphere needs to be displayed. This also allows non standard lighting effects, like Phong[Phon 75] illumination, an example of which is shown in Figure 9.

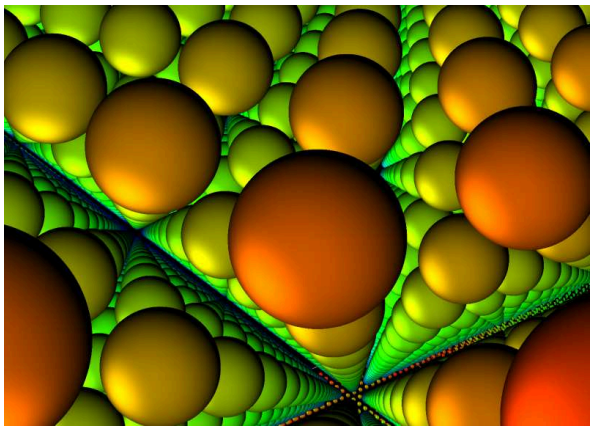


Figure 9: Phong illuminated spheres

GPU memory management

Vertex Buffer Objects (VBO)[Nvid 03] are a powerful way of managing GPU memory and RAM-to-GPU data transfers. We use them to create a cache system on GPU memory: for each tree leaf which is to be displayed, we create a VBO containing actual particles data used in rendering stage (positions, colors, radii). We unload it only if GPU memory is full and the leaf not displayed.

6. PARALLEL ALGORITHMS

This section presents the distributed strategies applied to previously described occluding and rendering algorithms.

Sort-last stage

As said before, we chose to use Ice-T, with *Reduce to Single Tile*[More 01] method: each process is assigned to one of the display tiles. First it renders the part of the scene which consists of the data loaded by this process. Then each process splits the rendered image, and for each part of the image, which is to be displayed by a tile, sends the part to one of the processes of the tile group. Each process receives a balanced number of partial images to be displayed by the tile. Then the processes of a same tile group compose their images by binary-swap before display. Moreland[More 03] tests on a generic architecture with a cluster and a tiled display were conclusive.

Since the Ice-T algorithm is very dependent on network bandwidth, it includes some optimizations, like floating viewport, which reduces the size of images transfered for compositing by detecting not rendered zones on the global viewport. This feature plays an important part in the sort-first algorithms efficiency and we used it as is.

Sort-first stage

6.2.1 *Partitioning of rendered tree parts*

We try to share parts of the tree to balance per cluster node rendering time and network transfers.

All processes have the complete tree structure loaded. Tree nodes numbering is in Z-order, like in [Shar 02a], which provides a good spatial coherence between nodes with close numbers.

For each frame, we assign the first not occluded leaves with an average number of particles to the first process, the following leaves to the second process, and so on, until all leaves are assigned, as seen in Figure 10. This average number or particles is the nearest integer to $(Visible particles)/(Number of cluster nodes)$, modulo the cardinality of the last leaf assigned to the current process.

The per node data is then spatially coherent, which is good for Ice-T floating viewport technique. Moreover, as two successive frames have relatively similar trees, cluster nodes have to render almost the same leaves, and the cache system is efficiently used.

6.2.2 *Overlapping culling algorithm*

The occlusion culling algorithm was easily modified to become distributed.

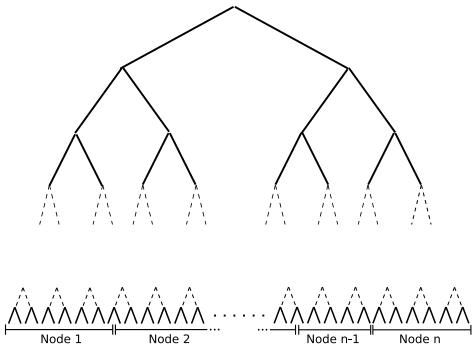


Figure 10: Displayable leaves assignment

Each frame, each cluster node performs an occlusion test, with only his previously assigned tree parts as potentially occluded cells. Then all nodes broadcast a signature of the tree, and each node merge them to have the entire tree configuration. The tree signature is a $(Numberofleaves)/4$ bytes buffer: each leaf is coded with 2 bits, because it has three states. All untested nodes are *not occluded*.

Although signature broadcast is not a very good scalable method, the really small size of the signature implies that this part does not slow down the overall process. As an example, with a typical depth of 15 for the tree for the 32 million particles dataset described in next section, the signature has a size of 8192 bytes, which is no more than Ethernet Gigabit maximum MTU (9000 bytes).

7. IMPLEMENTATION AND FIRST RESULTS

Base framework

Ice-T is integrated in Paraview[Ahre 05], based on the generic framework VTK. For our current implementation we use only core VTK and Ice-T, with a number of additional C++ classes compliant with VTK. We also created a *vtkMapper* family of classes to integrate VBO usage in VTK rendering process.

Protocol

We use a 32 millions particles dataset, describing a typical atomic system used in molecular dynamics detonation wave simulations. The graphics and display hardware is a four-tiled, 2048x1536 display, with an 8-node Gigabit Ethernet commodity cluster. Each node is a Bi-Xeon 3.4Ghz with a NVidia Quadro FX 4500 graphics board.

We compared three sort-first methods: a non hierarchical non ordered method, i.e. only a big VBO per cluster node; a Z-ordered repartition without occlusion; and the full occlusion method. In the first method, network is only used to send synchronization signals: data is distributed among nodes, and loaded once in GPU memory.

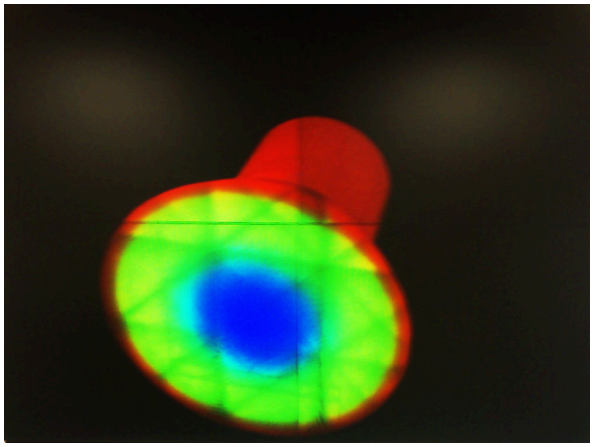


Figure 11: 32 million particle cylinder displayed on the 4-tile, 2m wide screen (around 3.3 M pixels)

As the full solution was designed to globally explore datasets, we tested two different camera movements: a rotation with constant long or short distance from dataset center; and a zoom towards a point of interest.

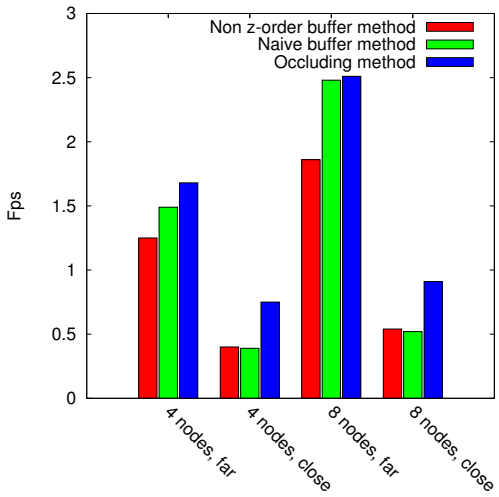


Figure 12: Camera rotating around dataset

On Figure 12 we can see the importance of Z-order in cluster nodes repartition.

We can notice that with eight nodes on the far view tests, culling (in blue) does not bring much better results than the naive (in green) method. The reason is that good Z-order data rendering repartition strongly reduces the floating viewport surfaces, which lower the network bandwidth usage. For example, in the 8 nodes far view test, bandwidth usage drops from 750Mb/s (effective maximum bandwitdth) with non Z-order method to an average 250Mb/s in both native and occluding method. On the close tests, Figure 12 shows the obvious efficiency of culling in such a situation. Z-order repartition and culling each reveal their efficiency in one of the different rotation scenarios.

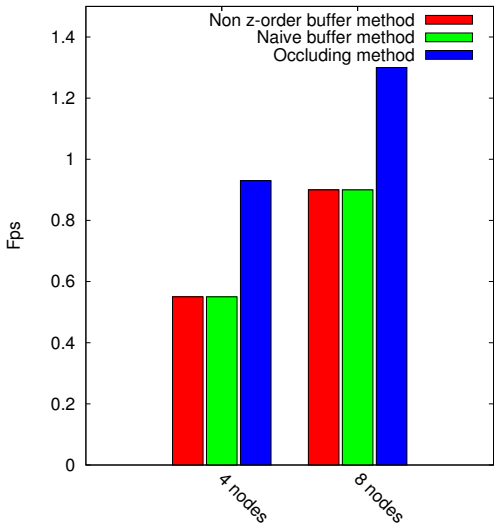


Figure 13: Camera zooming in dataset

Zooming results confirm this (Figure 13). As the camera gets closer to the dataset, the scene takes more and more space in the viewport. The network is saturated (like rotation tests, from an average 250Mb/s to 750Mb/s), and actual rendering becomes more and more time-consuming. Z-order is not very important, because the floating viewport is not activated here.

The rendering quality is a step function of the density threshold. If the density threshold is high enough, there is almost no difference between occluding and non occluding rendering. The threshold used for above results is 95%. For the camera rotation movement, 30% to 60% of the particles were culled. In the zoom test, culling got up to 94%. Lower threshold values obviously provide more interactive rendering, but artefacts do appear, like holes in the displayed dataset (cf Figure 14).

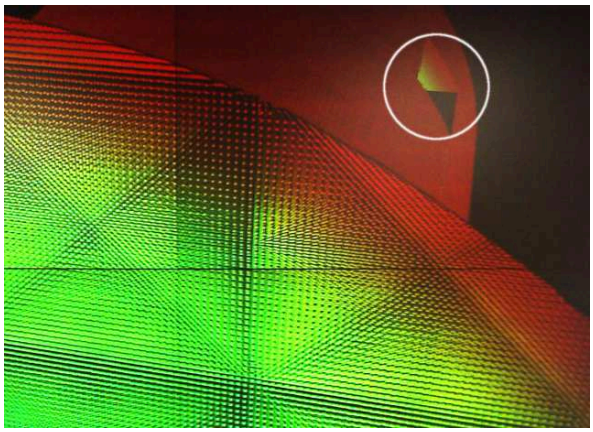


Figure 14: Artefacts with low density threshold

As priority for such a solution is global exploration and zone of interest detection, the most important factor in results is close rotation framerate. Our solution provides twice the framerate as the GPU only solution.

It is important to note that density is used the same way in far and in close cases. It should be interesting to lower threshold as camera's distance from datae increases. This could greatly improve results in far cases.

8. CONCLUSION AND FUTURE WORK

Our system provides a good framework for the exploration of large particle datasets representing dense material simulations. Sort-first rendering based on a specific Kd-Tree partitioning and statistical culling, with Monte-Carlo density computation, improves IceT sort-last strategies to reduce network bandwidth usage. Spheres rendering is also accelerated by GPU shaders. Future scale up tests will be achieved on a 12 tiled display fed by a 12 nodes cluster. We could further improve rendering by using better illumination methods, like ambient occlusion[Tari 06], to exploit the maximum potential of modern GPU power. Tests on a lower-latency and higher bandwidth network such as Infiniband should also be interesting.

Our next solution improvement will be to take camera's distance to the dataset into account to choose density threshold. We expect far camera tests framerate to be greatly improved.

Interactive tree computation is already fast, but with some optimizations, it could be done in real time. The main problem is density computation. One of the solutions could be to use General-Purpose computation on GPUs (GPGPU), like NVidia CUDA technology to accelerate the Monte-Carlo method.

9. ACKNOWLEDGEMENTS

Very special thanks to people from CEA/DIF, and especially Jean-Philippe Nominé for his support and his help, Thierry Carrard for his advices on GPU and parallel programming, Laurent Soulard for his teaching on materials physics. Also thanks to John Biddiscombe and Jean Favre from CSCS for respectively initial work on vtk sphere mapper classes and VTK advices, and Patrick Callet from Ecole Centrale Paris for his tips on sphere illumination, and reviewers for their thoughtful and accurate comments.

10. REFERENCES

[Ahre 05] J. Ahrens, B. Geveci, and C. Law. "ParaView: An End User Tool for Large Data Visualization". In: *Visualization Handbook*, Chap. 7, Academic Press, 2005.

[Baja 04] C. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane. "TexMol: Interactive Visual Exploration of Large Flexible Multi-Component Molecular Complexes". In:

VIS '04: Proceedings of the conference on Visualization '04, pp. 243–250, IEEE Computer Society, Washington, DC, USA, 2004.

[Coor 97] S. Coorg and S. Teller. "Real-time occlusion culling for models with large occluders". In: *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pp. 83–ff., ACM, New York, NY, USA, 1997.

[Ells 04] D. Ellsworth, B. Green, and P. Moran. "Interactive Terascale Particle Visualization". In: *VIS '04: Proceedings of the conference on Visualization '04*, pp. 353–360, IEEE Computer Society, Washington, DC, USA, 2004.

[Fluk 00] P. Flukiger, H. Luthi, S. Portmann, and J. Weber. "MOLEKEL 4.0". 2000.

[Grah 72] R. L. Graham. "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set". *Inf. Process. Lett.*, Vol. 1, No. 4, pp. 132–133, 1972.

[Hump 02] G. Humphreys, M. Houston, Y. Ng, R. Frank, S. Ahern, P. Kirchner, and J. Klosowski. "Chromium: A Stream Processing Framework for Interactive Graphics on Clusters". 2002. presented at SIGGRAPH, San Antonio, Texas.

[Hump 96] W. Humphrey, A. Dalke, and K. Schulten. "VMD - Visual Molecular Dynamics". *Journal of Molecular Graphics*, Vol. 14, pp. 33–38, 1996.

[Kess 06] J. Kessenich. "The OpenGL Shading Language (1.20)". Tech. Rep., opengl.org, 2006.

[Krug 05] J. Kruger, P. Kipfer, P. Kondratieva, and R. Westermann. "A Particle System for Interactive Visualization of 3D Flows". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 11, No. 6, pp. 744–756, 2005.

[Lian 05] K. Liang, P. Monger, and H. Couchman. "Interactive parallel visualization of large particle datasets". *Parallel Comput.*, Vol. 31, No. 2, pp. 243–260, 2005.

[More 01] K. Moreland, B. Wylie, and C. Pavlakos. "Sort-Last Parallel Rendering for Viewing Extremely Large Data Sets on Tile Displays". In: *PVG '01: Proceedings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics*, pp. 85–92, IEEE Press, 2001.

[More 03] K. Moreland and D. Thompson. "From cluster to wall with VTK". *IEEE symposium on Parallel and Large-Data Visualization and Graphics*, pp. 25–31, 2003.

[Nvid 03] Nvidia. "Using vertex buffer objects". Tech. Rep., NVIDIA Corporation, 2003. `http://developer.nvidia.com/object/using_VBOs.html`.

[Phon 75] B. T. Phong. "Illumination for Computer Generated Pictures". *Commun. ACM*, Vol. 18, No. 6, pp. 311–317, 1975.

[Sama 00] R. Samanta, T. Funkhouser, K. Li, and J. Singh. "Hybrid sort-first and sort-last parallel rendering with a cluster of pcs". In: *Eurographics/SIGGRAPH workshop on Graphics hardware*, pp. 99–108, 2000.

[Schr 06] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th edition*. Kitware, Inc. publishers, 2006.

[Shar 02a] A. Sharma. *Techniques and algorithms for immersive and interactive visualization of large datasets*. PhD thesis, Louisiana State University, 2002.

[Shar 02b] A. Sharma, X. Liu, P. Miller, A. Nakano, R. K. Kalia, P. Vashishta, W. Zhao, T. J. Campbell, and A. Haas. "Immersive and Interactive Exploration of Billion-Atom Systems". In: *VR '02: Proceedings of the IEEE Virtual Reality Conference 2002*, p. 217, IEEE Computer Society, Washington, DC, USA, 2002.

[Shar 03] A. Sharma, R. K. Kalia, A. Nakano, and P. Vashishta. "Large Multidimensional Data Visualization for Materials Science". *Computing in Science and Engg.*, Vol. 5, No. 2, pp. 26–33, 2003.

[Stre 05] F. H. Streitz, J. N. Glosli, M. V. Patel, B. Chan, R. K. Yates, B. R. de Supinski, J. Sexton, and J. A. Gunnels. "100+ TFlop solidification simulations on BlueGene/L.". In: *Gordon Bell Prize at IEEE/ACM SC05*, 2005.

[Tari 06] M. Tarini, P. Cignoni, and C. Montani. "Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 5, pp. 1237–1244, 2006.

[Zhan 97] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. "Visibility Culling Using Hierarchical Occlusion Maps". *Computer Graphics*, Vol. 31, No. Annual Conference Series, pp. 77–88, 1997.