Eurographics Symposium on Point-Based Graphics (2004)
M. Alexa, S. Rusinkiewicz, (Editors)

Layered Point Clouds

Enrico Gobbetti and Fabio Marton

CRS4 Visual Computing Group[†]

Abstract

We present a simple point-based multiresolution structure for interactive visualization of very large point sampled models on consumer graphics platforms. The structure is based on a hierarchy of precomputed object-space point clouds. At rendering time, the clouds are combined coarse-to-fine with a top-down structure traversal to locally adapt sample densities according to the projected size in the image. Since each cloud is made of a few thousands of samples, the multiresolution extraction cost is amortized over many graphics primitives, and host-to-graphics communication effectively exploits on-board caching and object based rendering APIs. The progressive block based refinement nature of the rendering traversal is well suited to hiding out-of-core data access latency, and lends itself well to incorporate backface, view frustum, and occlusion culling, as well as compression and view-dependent progressive transmission. The resulting system allows rendering of complex models at high frame rates (over 60M splat/second), supports network streaming, and is fundamentally simple to implement.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Out-Of-Core Algorithms, Level of Detail

1. Introduction

Current 3D digital photography and 3D scanning systems make it possible to acquire at high resolution both geometry and appearance of complex, real-world objects. At the same time, the size and complexity of isosurfaces generated by numerical simulation is steadily growing, due to the proliferation and performance of affordable scalable high performance computers. Despite the rapid improvement in graphics hardware performance, rendering at interactive speeds the huge datasets generated by the acquisition and simulation pipe-lines remains a very challenging problem, since they still largely overload the performance and memory capacity of state-of-the-art graphics and computational platforms. For such complex and dense models, multiresolution hierarchies of point primitives have recently emerged as a viable alternative to the more traditional mesh refinement [RL00]. These methods are based on the assumption that model sampling rate is so high that triangles are projected to such small screen areas that the advantage of scanline coherence are lost, and appropriately selected point samples are sufficient to accurately reproduce the model. One of the major benefits of this approach is its simplicity, stemming from the fact that there is no need to explicitly manage and maintain mesh connectivity during both preprocessing and rendering.

Unfortunately, current dynamic multiresolution algorithms for large models are very CPU intensive: nowadays, consumer graphics hardware is able to sustain a rendering rate of tens of millions of point primitives per second, but current multiresolution solutions fall short of reaching such performance. This is because the CPU is not able to generate/extract point samples from the out-of-core structure and send them fast enough to the graphics hardware in the correct format and through a preferential data path.

Our contribution is a simple point-based solution for high performance view dependent visualization of very large static point sampled models on consumer graphics platforms. The underlying idea of the method is to reduce the per-primitive structure overhead by moving the grain of the multiresolution model from a hierarchy of point samples to a hierarchy of precomputed object-space point clouds. At rendering time, the clouds are combined coarse-to-fine with a stateless top-down structure traversal to locally adapt sample densities according to the projected size in the image. Since

[†] CRS4, POLARIS Edificio 1, 09010 Pula (CA), Italy – `http://www.crs4.it/vic/` – `first.last@crs4.it`

© The Eurographics Association 2004.

each cloud is made of a few thousands of samples, the multiresolution extraction cost is amortized over many graphics primitives, and host-to-graphics communication can effectively exploit on-board caching and object based rendering APIs. The progressive block based refinement nature of the rendering traversal is well suited to hiding out-of-core data access latency, and lends itself well to incorporate backface, view frustum, and occlusion culling, as well as compression and view-dependent progressive transmission. The resulting system allows rendering of local and remote models of hundreds of millions of samples at high frame rates (over 60M splat/second), supports network streaming and is fundamentally simple to implement.

The rest of the paper is organized as follows. Section 2 reviews related works. The details of the proposed data structure are presented in section 3, while section 4 describes algorithms for view-dependent refinement and rendering, and section 5 propose a simple out-of-core technique for constructing the multiresolution model. The efficiency of the approach is demonstrated with the inspection of a number of very large models, including a massive 234M samples isosurface generated by a compressible turbulence simulation, that exhibits a huge (>100) depth complexity, and a 167M samples model of Michelangelo's St. Matthew (section 6).

2. Related Work

Point-based 3D graphics techniques for processing and rendering of dense models are an old idea [LW85, Gro98], that has found many successful applications, including point-based modeling, high quality and interactive rendering, as well as coding and transmission of point-based models.

Our focus is the development of systems for the distribution and high speed interactive visual inspection of very large models on commodity graphics platforms. QSplat [RL00] is the reference system in this particular area. The system is based on a hierarchy of bounding spheres maintained out-of-core, that is traversed at run-time to generate points. This algorithm is CPU bound, because all the computations are made per point, and CPU/GPU communication requires a direct rendering interface, thus the graphic board is never exploited at his maximum performance. In Streaming QSplat [RL01], the QSplat data structure is subdivided into chunks, that are however only used for streaming objects over networks. The rendering procedure remains a hierarchical traversal done on the CPU, with the additional book-keeping required to check the local availability of data. Kalaiah and Varshney [KV03] have recently proposed to improve the geometry bandwidth bottleneck by working on a compressed point sample geometry model obtained by principal component analysis. Even if they use a large cache of 40M points, the need to regenerate a large number of small point clusters per frame from statistical information leads to a rendering speed which is roughly half the speed of QSplat. We exploit instead a partitioning of the model into clouds to improve the efficiency of CPU/GPU communication through a batched communication protocol and to support conservative occlusion culling for high depth complexity models. This provides at least an order of magnitude improvement in rendering rate on current commodity graphics platforms.

A number of authors have also proposed various ways to push the rendering performance limits in particular situations. The randomized z-buffer [WFP*01] uses a hierarchical traversal of a structure where the leaf nodes contain arrays of random point samples. Stamminger and Drettakis [SD01] dynamically adjusts the point sampling rate for rendering complex procedural geometry at high frame rates. They require a parameterization of the model, while we focus on unstructured point samples. Dachsbacher et al. [DVS03] recently presented a hierarchical LOD structure for points that is adaptively rendered by sequential processing done on the GPU. They report a peak performance of over 50M unfiltered points per second, which is similar to ours, but they are limited in the size of the rendered model, which must fit into the video card memory, while our work focuses instead on very large local and remote models.

There is a large body of work that aims at improving the rendering quality of point-sampled models. For dense models, these include using spheres [RL00], tangential disks [PZvBG00, ZPvBG01], or high degree polynomials [ABCO*01] instead of raw point primitives, as well as improving filtering in image space [ZPvBG01] or object space [RPZ02]. Such work is orthogonal to ours, which focuses on finding simple ways to improve raw rendering performance on very large models by amortizing costs on groups of many graphics primitives. Merging these two directions, possibly by exploiting GPU programming as in [BK03], is a main avenue for future work.

3. Multiresolution model

We assume that the input model is represented by a set of N sample points uniformly distributed over its surface, with an average spacing between samples equal to r. Each sample point is associated with a set of surface attributes, including position, normal, and possibly color information.

Our multiresolution approach creates a hierarchy over the samples of the datasets, simply by reordering and clustering them into point clouds of approximately constant size arranged in a binary tree. In other words, the final multiresolution model has exactly the same points of the input model, but grouped into chunks and organized in a level of detail representation. The root of the level of detail tree represents the entire model with a single cloud of $M_0 = M < N$ uniformly distributed samples. The remaining points are equally subdivided among the two subtrees using a spatial partition, with, again, M uniformly distributed points directly associated to the root of each subtree, and the rest redistributed in the children. The leaves are terminal clusters, which are further indivisible and whose size is smaller than the specified limit M.

Variable resolution representations of the models are obtained by defining a *cut* of the hierarchy and merging all nodes above the cut. This way, each node acts as a *refinement* of a small contiguous region of the parent. The root node is the coarsest available model representation, with a average sample spacing of $r_0 = r\sqrt{\frac{N}{M_0}}$. Each node j, then, locally refines its parent by adding additional M_j samples to the representation. That is, the total number of samples $M_j^{(tot)}$ extracted in the region associated to node j is recursively defined by

$$M_j^{(tot)} = \frac{N_j}{N_{parent(j)} - M_{parent(j)}} M_{parent(j)}^{(tot)} + M_j \qquad (1)$$

where N_k is the total number of samples in the subtree rooted at k, and for the root $M_0^{(tot)} = M_0$. Because of our uniform sampling assumption, the resulting average sample spacing of the region of a non-root node is thus decreased to

$$r_j = r_{parent(j)} \sqrt{\frac{N_j M_{parent(j)}^{(tot)}}{(N_{parent(j)} - M_{parent(j)}) M_j^{(tot)}}} \qquad (2)$$

By storing at each node j the value of r_j along with its point cloud, we can thus rapidly obtain a variable accuracy representation by traversing top down the hierarchy, while accumulating point clouds until the desired density is reached (see figure 1). Since we are interested in view-dependent representations, we also precompute the bounding sphere and bounding cone of normals of each node. These are used for projecting the mean sample distance to the screen, as well as for view-frustum, backfacing, and occlusion culling (see section 4).

The benefits of this approach are that the workload required for a unit refinement/coarsening step is amortized on a large number of point primitives, and that the small point clusters can be optimized off-line for best performance in host-to-graphics and network communication. By tuning the value of parameter M, we can vary the granularity of the structure from a total multi-resolution model (e.g., QSplat for $M = 1$) to a single-resolution model for point rendering ($M = N$). The choice of parameter M is dictated by performance considerations. In particular, if M is too large, the model becomes less adaptive, and switching from a resolution level to the next leads to a high latency. On the other hand, if M is too small, the model is more adaptive but CPU costs become non negligible. On current graphics platforms, we have empirically determined that the best performance trade-offs are obtained for values of M ranging from 512 to 8192.

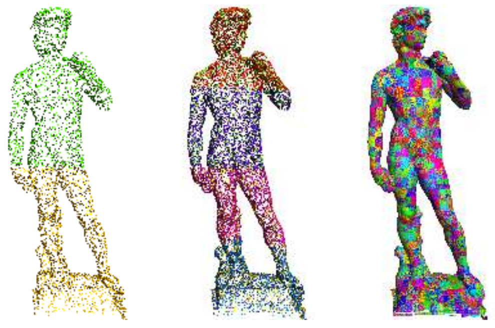

(a) level: 0, clusters: 2, splats: 4K (b) level: 0+1, clusters: 6, splats: 12K (c) pixel tol.: 1, clusters: 1720, splats: 3435K

Figure 1: ***David 1mm different level representations.*** *The original model is made of 28M points. The images illustrate how clusters of finer levels add information to the coarser representations.*

4. The rendering pipe-line

Our adaptive rendering algorithm works on a standard PC, and data is assumed to be either locally stored on a secondary storage unit directly visible to the rendering engine or remotely stored on a network server (see figure 2).

Data layout and data access. The hierarchical data structure is split into an index tree and a point cloud repository. The index tree has a small footprint, since it contains, for each node, just the data required for traversal (sample spacing, bounding sphere, bounding cone of normals, and index of the two children), and refers to the associated point cloud through a 32 bit index that uniquely identifies the cloud in the repository. The repository is organized so that the data storage order reflects traversal order, which is coarse to fine and by physical position in space. We thus sort point clouds in the repository using as a primary key their tree level, and as a secondary key the Morton index of their bounding sphere center [Sam90]. For disk storage and data transmission, each cloud is managed in compressed form. The point cloud is spatially sorted, then each attribute is quantized, delta encoded, and then entropy encoded with the LZO compressor[†]. Access to the point cloud repository is made through a data access layer, that masks to the application whether the repository is local or remote. This layer makes it possible to asynchronously move in-core a point cloud by fetching it from the repository, to test whether a point cloud is immediately available, and to retrieve its representation. We have implemented two versions of this access layer: the

[†] LZO is a data compression library based on a Lempel Ziv variant which is suitable for data decompression in real-time. The library source is available from `http://www.oberhumer.com/opensource/lzo/`

first one provides direct disk access through memory mapping functions and is used for local files as well as remote NFS mounted files. The second one is based on the HTTP 1.1 protocol and, similarly to Streaming QSplat [RL01], fetches data from a standard HTTP server using range requests and permanent connections.

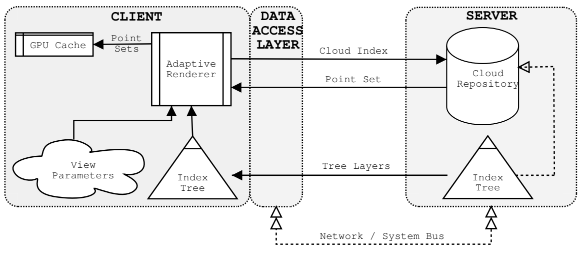


Figure 2: ***The rendering pipeline.*** *The client traverses the index tree coarse-to-fine in a view-dependent manner, requesting point clouds to the server. To maximize rendering performance and minimize traffic, point clouds are cached on board using a LRU strategy.*

Progressive view-dependent refinement. The traversal algorithm, which extracts a view dependent representation of the multiresolution model from the current point of view, is based on a stateless coarse-to-fine refinement of our structure, that exploits the progressive nature and coarse granularity of the multiresolution hierarchy to reduce CPU refinement costs and to improve repository-to-host and host-to-graphics communication. In particular, asynchronous repository requests hide out-of-core data access latency, and communication with the GPU is made exclusively through a retained mode interface, which reduces bus traffic by managing a least-recently-used cache of point clouds maintained on-board as OpenGL *Vertex Buffer Object*.

The user selected pixel threshold is the value that drives the refinement of the rendering algorithm: this value represents the required average sample distance between adjacent splats on the screen, and it is used as splat size. The refinement algorithm performs a single pass recursive traversal of the multiresolution structure. For each node, we use its bounding sphere and normal cone to test whether the node is totally outside the view frustum or totally backfacing. In this case, recursion stops, discarding the entire branch of the tree, otherwise we can render the node and, eventually, continue the refinement with its children. Since we are focusing on high speed visualization, our current implementation simply uses OpenGL hardware supported points for point cloud rendering. This fact limits our ability to correctly treat texture and transparency. Using ellipsoidal splats computed on the GPU, as in, e.g, [BK03], would resolve these problems.

At node rendering time, we project the node's hierarchical average sample distance to the screen to obtain its splat size. A consistent upper bound on the projected size is obtained by measuring the apparent size of a sphere with diameter equal to the object space average sample distance and centered at the bounding sphere point closest to the viewpoint. If the projected splat size is less than the threshold, we render the node's point cloud with the prescribed splat size and stop recursion, otherwise a refinement is needed. In that case, to avoid blocking the renderer because of data access latency, especially in the case of rendering data over wide-area networks, we first check whether the node's children data is immediately available, i.e., if it is already in the GPU cache or considered in-core by the data access layer. If so, we continue recursion, otherwise recursion stops and the node is rendered with an increased splat size, equal to its projected mean sample distance, to cover holes left by children unavailability. Fetch requests are then pushed in a priority queue. Similarly to Streaming QSplat [RL01], the request queue is traversed in order of priority at the end of the frame, issuing only as many requests as those allowed by the estimated network bandwidth, and forgetting the remaining ones. Since the repository is sorted coarse to fine and by physical position in space, prioritizing the queue by node's index provides a simple compromise that is both I/O efficient and promises to download the most relevant data as soon as possible while being enough space coherent to minimize visual distraction.

Rendering on a budget. For interactive applications, it is often useful to have direct control on rendering time, instead of the control on rendering quality provided by prescribing a screen error tolerance for the refinement method. In addition to adjusting error tolerance per frame in a feedback loop, we can exploit the fact that our hierarchy is shallow to implement a predictive technique. Given a desired number of points per frame, we perform a binary search of the associated pixel threshold, by repeatedly traversing the index tree with the same refinement logic used for rendering, while only counting the number of generated primitives.

Occlusion culling. A number of complex dense models, such as large isosurfaces deriving from numerical simulation of turbulence (e.g., [GDL*02, MCC*99]) have an important depth complexity. For these models, efficiently culling the invisible portion of the rendered model is of primary importance to avoid uploading, refining, and rendering unnecessary data (see figure 3). Since our structure is coarse grained and provides a spatial partition, we can adapt to a point rendering framework visibility techniques developed for rendering scenes composed of many objects. Similarly to the approach introduced by Toon et al. [YSM03] for complex CAD environments, our rendering algorithm exploits frame-to-frame coherence in occlusion culling, by using the set of visible point clouds from the previous frame as the occluder set for the current frame. At each frame, we render the object in three phases. In the first phase, we perform the usual refinement algorithm, but accumulate the clouds that would be rendered in a list of potentially visible objects, while only rendering the point clouds that were visible in the previous frame. In a second phase, we traverse the entire list of accumulated point sets, generating a hardware occlusion query for the object's bounding sphere (approximated by an icosahedron), using OpenGL ARB_occlusion_query extension to track the number of fragments that pass the depth test. In a third and final pass, we traverse again the list of clouds and query the associated occlusion query object for the number of passed fragments. If this number is above a given threshold, we insert the cloud index in next frame's occluder list and, if the cloud was not among those rendered in the first pass, we proceed render it. With this method, the only additional cost of occlusion culling is the generation and test of occlusion queries. This cost can be further reduced by only checking once every few frames if previous frame occluders are still visible.

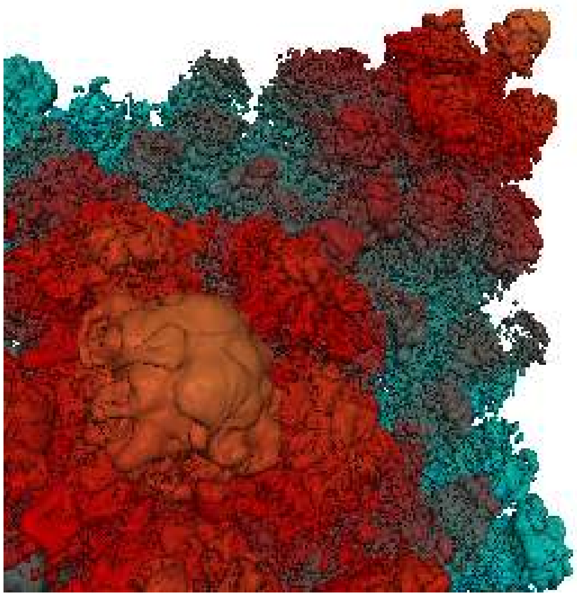


Figure 3: ***Occlusion culling.*** *Closeup view of an isosurface feature in the mixing interface of two gases for a simulation of a Richtmyer-Meshkov instability in a shock tube [MCC*99] rendered at 1 pixel tolerance on a 335x335 window. Without occlusion culling: 12976 patches, 24M splats, 1.7 fps; with occlusion culling: 3490 patches, 6.3M splats, 5.5 fps.*

5. Construction

The multiresolution point-cloud structure is constructed off-line starting from a generic point cloud model. We have implemented a simple I/O efficient recursive clustering method, that is implemented with a single out-of-core component: a standard C++ array (compatible with `std::vector`), that encapsulates a resizable file accessed through system memory mapping functions. The procedure consists of two phases.

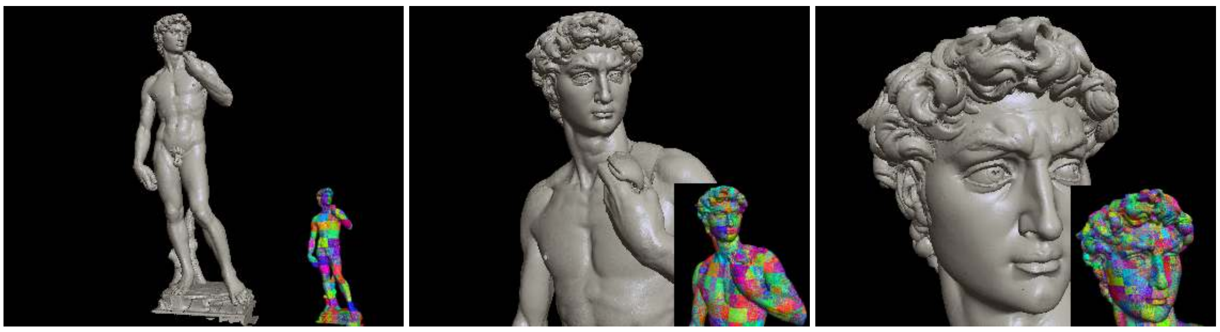
Recursive partitioning. The first phase partitions the input dataset into a tree of point clouds. The partitioning procedure takes as input an external memory array of uniformly distributed point samples, together with its bounding volume. If the number of points N is less than the predefined node sample count M, a leaf node is generated, otherwise M samples are extracted by uniform subsampling, and the remaining samples are distributed to the two children by bisecting the bounding box at the midpoint of its longest axis, and recursively continuing the partitioning procedure. A simple and I/O efficient approach to the subsampling problem is to pick $K*M, K>1$ random samples from the point set using a Russian roulette approach during the linear scan required for the partitioning. Once the $K*M$ samples are in-core, we random shuffle them, select the first M samples, and redistribute the remaining $(K-1)*M$ samples to the two children. The technique does not ensure that splat size is constant per cloud, but maintains it reasonably close to the average value for complex models. Since the screen projection operation is very conservative, we have found this sufficient in practice to produce images without holes with a quality comparable to that of QSplat.

Structure construction. The end result of the partitioning procedure is a tree of nodes, which provides the layout for the index tree, and a set of point clouds, that will be part of the repository. In a second phase, we thus complete the structure by traversing the nodes in the order in which they are stored in the repository (i.e., by level and then by Morton index), computing the index node data (hierarchical sample spacing, bounding sphere, and bounding cone of normals), and converting the point cloud to the final compressed representation.

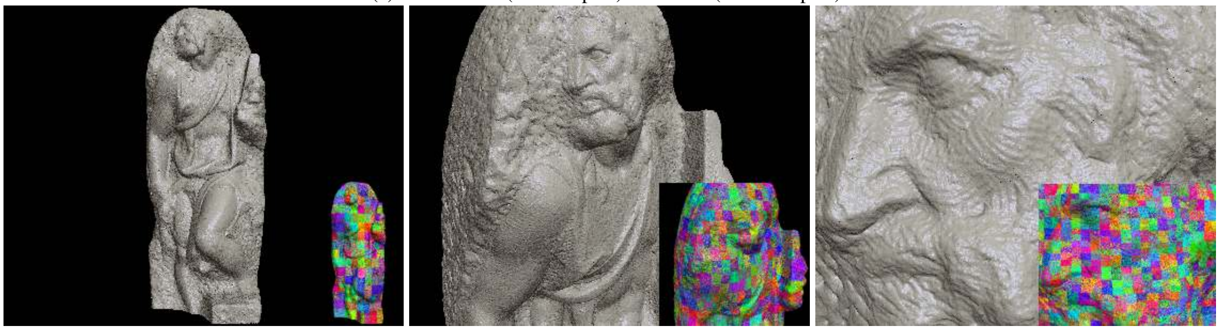
6. Results

The proposed method has been used to develop a C++ application which makes use of OpenGL on a Linux platform. Several tests have been performed on preprocessing and rendering of a number of very large models (see figure 4). The largest model is a full resolution isosurface of the mixing interface from the Gordon Bell Prize winning simulation of a Richtmyer-Meshkov instability in a shock tube experiment [MCC*99], that consists of over 234M sample points extracted from a 2048x2048x1920 8bit grid. This model is convoluted and has a huge depth complexity (>100) from all viewpoints. The other test cases are high resolution scans of the St. Matthew and David statues from The Digital Michelangelo repository.

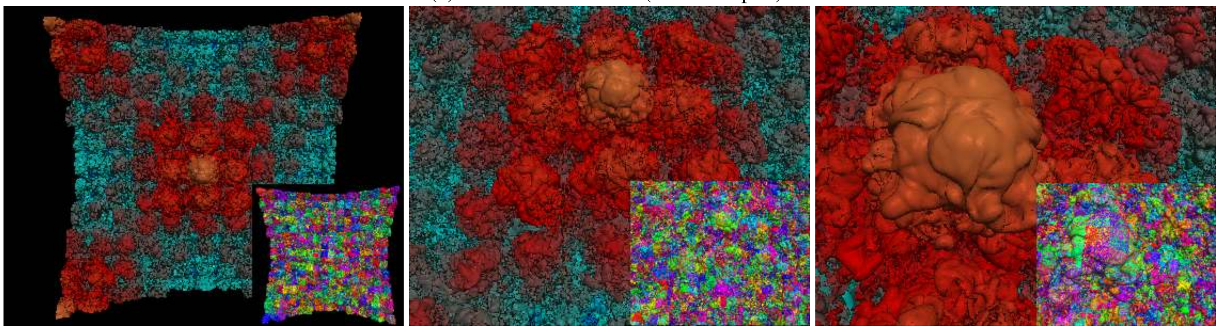
Preprocessing. Table 1 shows numerical results for the out-of-core preprocessing of our algorithm, relative to all test



(a) David 2mm (4M samples) and 1mm (28M samples)



(b) St. Matthew 0.25mm (167M samples)



(c) Mixing interface isosurface (234M samples)

Figure 4: ***Test models.*** *The main images show the models as presented to the user during interactive inspection sessions, while the inset images illustrate the subdivision structure.*

cases. The preprocessing has been evaluated on a PC running Linux 2.4, with two Athlon 2200+ CPUs, 1GB DDR memory, a 70GB ATA 133 hard disk. All the multiresolution models have been constructed with M set to 2K samples/node, using 16 bit/normal quantization and a position quantization ensuring a quantization error inferior to half of the input sampling distance. Overall processing times range from 18K samples/s to 30K samples/s depending on the processor load, and is dominated by disk access times and LZO compression. Compression rates exceed those of QSplat (around 50bits/sample) and other similar systems based on a point hierarchy, but do not match those of state-of-the-art compression systems, since our current implementation has favored ease of coding through the exploitation of general purpose compression libraries. They could be improved by exploiting the locality of each patch, quantizing attributes relative to each cluster's contents.

Model	Samples	Extent (mm)	Pos. Quant. (bits)	Norm Quant. (bits)	Total time (s)	Disk usage (MB)		Output Bit / Sample
						in	out	
David 2mm	4,138,653	5,200	3x13	16	142	95	20	40
David 1mm	28,120,980	5,200	3x14	16	1,233	644	126	37
St. Matthew	167,324,853	2,700	3x15	16	8,820	3,830	777	39
PPM Isosurface	234,717,830	2,048	3x13	16	13,357	5,372	925	33

Table 1: ***Numerical results for out-of-core construction.*** *Tests performed on a single PC. All times are in seconds.*

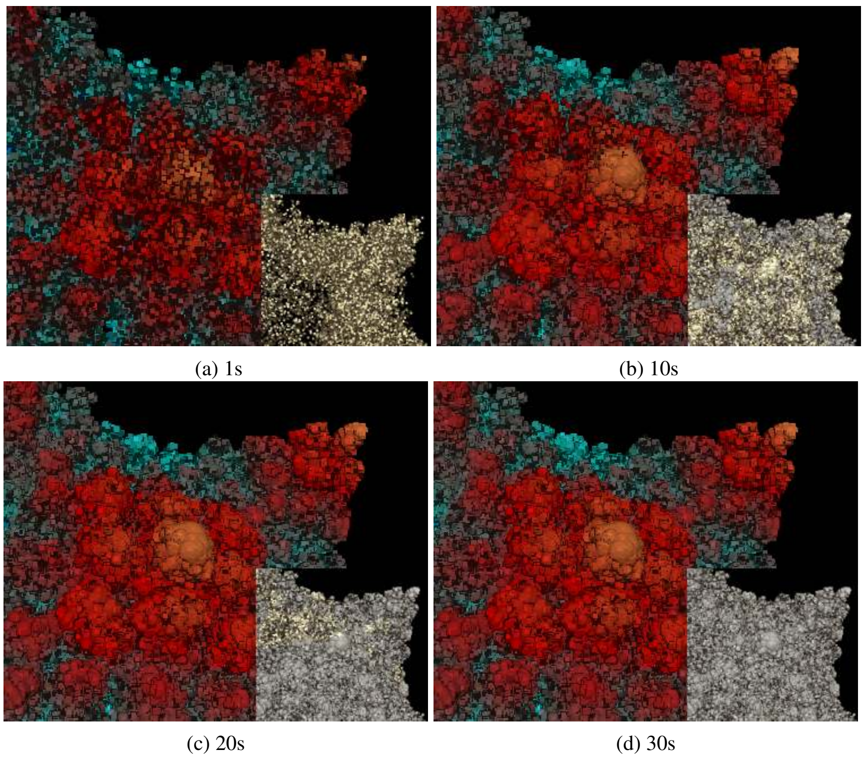


Figure 5: ***Streaming.*** *Progressive refinement of the mixing interface isosurface (234M samples) on a ADSL connection at 1.25M bps. The main images show the model as presented to the user, while the inset images illustrate progressive refinement. Yellow patches indicate areas where refinement stops because of missing data.*

View-dependent refinement. We evaluated the performance of our view-dependent refinement technique on a number of inspection sequences over the test case models. The results were collected on a Linux PC with a Intel Xeon 2.4 GHz, 2GB RAM, two Seagate ST373453LW 70 GB ULTRA SCSI 320 hard drives, AGP 8x and NVIDIA GeForce FX 5800 Ultra graphics. During the entire walkthrough, the resident set size of the application for the largest test case never exceeded 242MB, i.e. less than 27% of the out-of-core data size, demonstrating the effectiveness of out-of-core data management. The qualitative performance of our view-dependent refinement is illustrated in an accompanying video that shows recorded live sequences[‡]. As demonstrated in the videos on the 3D scanning models, that do not employ occlusion culling, we can sustain an average rendering rate of around 40M splat/s, with peaks exceeding 68M splat/s. By comparison, on the same machine, the peak performance of QSplat, was measured at roughly 3.6M splats/s when using the GL_POINTS rendering primitive. For the inspection of the 234M samples isosurface, which has a huge depth complexity, we have enabled occlusion culling. On average 50% of the patches are detected as occluded, strongly diminishing data access and rendering times. The average rendering rate drops in this case to around 30M splat/s, which is still about an order of magnitude faster than that of QSplat. For the same view, and with the same screen space tolerance, we have measured that our method renders up to 10% more points than QSplat. This is because grouping points into clouds for all operations forces us to be more conservative in the projection. The increase in number of points is however compensated by a much larger increase in rendering speed.

Network streaming. Some network tests have been performed on all test models, on a local area network at 100Mbps and on a ADSL at 1.2M bps, using both NFS mounts and HTTP 1.1 connections. As illustrated in the video, rendering rate remains the same as that of the local file version, but updates asynchronously arrive with increased latency. The effect is illustrated in figure 5, which shows the progressive refinement of the largest dataset on a machine connected through ADSL to a moderately loaded Linux box running a Apache web server. Even though the HTTP 1.1 is far from being optimal for the task, the application remains usable even for very large models on consumer-level network connections. The first images in the progressive refine-

[‡] The video is available from: `http://www.crs4.it/vic/multimedia/`

ment sequence also illustrate that a heavy subsampling on coarser scales can lead to strong aliasing artifacts for very comples models, as the average sampling distance is significantly below Nyquist frequency. Our static sample randomization replaces missing data with random information which is stable over time, thus the visual effect of aliasing is less noticeable as it would be for a regular sampling at similarly coarse resolutions. Nevertheless, for some models the occurring aliasing could notably diminish visual quality. Improving this aspect is an important avenue for future work.

7. Conclusions

We have presented a simple point-based multiresolution structure for interactive out-of-core visualization of very large point models on consumer graphics platforms. The system is comparable in both implementation complexity and image quality to (Streaming) QSplat. Despite its simplicity, it is able to handle models of much higher depth complexity and is at least an order of magnitude faster in terms of rendering speed. The current major limitation is in image quality. Since we are focusing on high speed visualization, we simply use OpenGL hardware supported points for point cloud rendering, and do not use a per-sample splat size, which limits our ability to correctly treat texture and transparency. The integration with more advanced filtering techniques implemented on the GPU would resolve these problems without compromising too much rendering speed. Given its simplicity, the method is of immediate practical interest, and inserts itself in the current stream of techniques where the power of current graphics architectures is exploited through coarse grained multiresolution structures.

Acknowledgments. We are grateful to the Stanford Graphics Group and the Lawrence Livermore National Laboratories for making benchmark datasets available. Special thanks also go to Valerio Pascucci (LLNL) and Marco Agus (CRS4).

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *Proceedings of the conference on Visualization '01* (2001), IEEE Computer Society, pp. 21–28. 2

[BK03] BOTSCH M., KOBBELT L.: High-quality point-based rendering on modern GPUs. In *Proc. Pacific Graphics* (2003), pp. 335–343. 2, 4

[DVS03] DACHSBACHER C., VOGELSGANG C., STAMMINGER M.: Sequential point trees. In *Proc. SIGGRAPH* (2003). 2

[GDL*02] GREGORSKI B., DUCHAINEAU M., LINDSTROM P., PASCUCCI V., JOY K. I.: Interactive view-dependent rendering of large IsoSurfaces. In *Proceedings of the 13th IEEE Visualization 2002 Conference (VIS-02)* (Piscataway, NJ, Oct. 27– Nov. 1 2002), Moorhead R., Gross M.,, Joy K. I., (Eds.), IEEE Computer Society, pp. 475–484. 4

[Gro98] GROSSMAN J. P.: *Point Sample Rendering*. Master's thesis, Dept. of Electrical Engineering and Computer Science, MIT, 1998. 2

[KV03] KALAIAH A., VARSHNEY A.: Statistical point geometry. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2003), Eurographics Association, pp. 107–115. 2

[LW85] LEVOY M., WHITTED T.: *The use of points as a display primitive*. Tech. Rep. TR 85-022, University of North Carolina at Chapel Hill, 1985. 2

[MCC*99] MIRIN A. A., COHEN R. H., CURTIS B. C., DANNEVIK W. P., DIMITS A. M., DUCHAINEAU M. A., ELIASON D. E., SCHIKORE D. R., ANDERSON S. E., PORTER D. H., WOODWARD P. R., SHIEH L. J., WHITE S. W.: Very high resolution simulation of compressible turbulence on the IBM-SP system. In *Supercomputing '99* (1999), ACM Press and IEEE Computer Society Press. 4, 5

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *SIGGRAPH 2000, Computer Graphics Proceedings* (2000), Akeley K., (Ed.), Annual Conference Series, ACM Press - Addison Wesley Longman, pp. 335–342. 2

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: A multiresolution point rendering system for large meshes. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 00)* (July 24-28 2000), ACM Press, pp. 343–352. 1, 2

[RL01] RUSINKIEWICZ S., LEVOY M.: Streaming qsplat: A viewer for networked visualization of large, dense models. In *Symposium for Interactive 3D Graphics Proceedings* (2001). 2, 4

[RPZ02] REN L., PFISTER H., ZWICKER M.: Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *EUROGRAPHICS 2002 Proceedings* (2002). 2

[Sam90] SAMET H.: *The design and Analysis of Spatial Data Structures*. Addison Wesley, Reading, MA, 1990. 3

[SD01] STAMMINGER M., DRETTAKIS G.: Interactive sampling and rendering for complex and procedural geometry. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (2001), Springer-Verlag, pp. 151–162. 2

[WFP*01] WAND M., FISCHER M., PETER I., AUF DER HEIDE F. M., STRASSER W.: The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *Siggraph 2001 Proceedings* (2001). 2

[YSM03] YOON S.-E., SALOMON B., MANOCHA D.: Interactive view-dependent rendering with conservative occlusion culling in complex environments. In *Proc. IEEE Visualization* (2003). 4

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Siggraph 2001 Proceedings* (2001). 2