High Quality Interactive Rendering of Massive Point Models using Multi-way kd-Trees

Prashant Goswami
University of Zurich
Switzerland
goswami@ifi.uzh.ch

Yanci Zhang
Sichuan University
China
yczhang7@gmail.com

Renato Pajarola
University of Zurich
Switzerland
pajarola@acm.org

Enrico Gobbetti
CRS4
Italy
gobbetti@crs4.it

***Abstract*—We present a simple and efficient technique for out-of-core multiresolution construction and high quality visualization of large point datasets. The method introduces a novel hierarchical LOD data organization based on *multi-way kd-trees* that simplifies memory management and allows controlling the LOD tree's height. The technique is incorporated in a full end-to-end system, which is evaluated on complex models made of hundreds of millions of points.**

***Keywords*-point based rendering; gpu-based; multi-way kd-tree;**

I. Introduction

Modern 3D scanning systems can generate massive data sets with hundreds of millions of points. Despite the advances in point-based modeling and rendering, these models can still be a challenge to be processed and in particular rendered with high quality at interactive frame rates. Massive models are tackled with level-of-detail (LOD) and out-of-core techniques. However, since modern GPUs sustain very high primitive rendering rates, a slow CPU based adaptive data selection process can easily lead to starvation of the graphics pipeline. Hence the CPU-GPU bottleneck becomes the prominent challenge to be addressed as a whole, from the choice of data structures used for preprocessing and rendering the models, to the design of the display algorithm. In recent years, this consideration has lead to the emergence of a number of coarse-grained multiresolution approaches based on a hierarchical partitioning of the model into clouds consisting of thousands of points. These methods successfully amortize data structure traversal overhead over rendering time of large numbers of primitives, and effectively exploit on-board caching and object based rendering APIs.

In this paper, we further improve the state-of-the-art in the domain with a novel end-to-end system for out-of-core multiresolution construction and high quality visualization of large point datasets. We exploit the properties of *multi-way kd-trees* to make high quality rendering more GPU oriented. The level-of-detail (LOD) tree, constructed bottom up using a fast high-quality simplification method, is fully balanced, has controllable depth, and contains uniformly sized nodes. Since all blocks are equal sized, on-board memory management is particularly simple and effective.

The presented approach results in fast pre-processing and high quality rendering of massive point clouds at hundreds of millions of splats/second on modern GPUs, improving the quality vs. performance ratio compared to previous large point data rendering methods.

II. Related Work

While the use of points as rendering primitives has been introduced as early as in [1], [2], only over the last decade they have reached the significance of fully established geometry and graphics primitives, see also [3]–[5]. Several techniques have since been proposed for improving upon the display quality, LOD rendering, as well as for efficient out-of-core rendering of large point models. We concentrate here on discussing only the approaches most closely related to ours. For more details, we refer the reader to the survey literature [5]–[7].

QSplat [8] has for long been the reference system in massive point rendering. It is based on a hierarchy of bounding spheres maintained out-of-core, that is traversed at runtime 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 graphics board is never exploited at its maximum performance. A number of authors have also proposed various ways to push the rendering performance limits, mostly through coarse grained structures and efficient usage of retained mode rendering interfaces.

Sequential Point Trees [9] introduced a sequential adaptive high performance GPU oriented structure for points limited to models that can fit on the graphics board. XSplat [10] and Instant points [11] extend this approach for out-of-core rendering. XSplat is limited in LOD adaptivity due to its sequential block building constraints, while Instant points mostly focuses on rapid but moderate quality rendering of raw point clouds. Both systems suffer from a non-trivial implementation complexity. Layered point clouds (LPC) [12] and Wand et al.'s out-of core renderer [13] are prominent examples of high performance GPU rendering systems based

on a hierarchical model decomposition into large sized blocks maintained out-of-core. LPC is based on adaptive BSP subdivision, and subsamples the point distribution at each level. In order to refine a LOD, it adds points from the next level at runtime. This composition model and the pure subsampling approach limits the applicability to uniformly sampled models and produces moderate quality simplification at coarse LODs. In [14] these limitations are removed by making all BSP nodes self-contained and using an iterative edge collapse simplification to produce node representations. We propose here a faster high quality simplification method based on adaptive clustering that provides more control over the LOD tree height through the use of N-ary trees. Wand et al.'s approach [13] is based on an out-of-core octree of grids, and deals primarily with grid based hierarchy generation and editing of the point cloud. This method's limitation is in the quality of lower resolutions created by the grid no matter how fine it is.

All the mentioned pipe-lines for massive model rendering create coarser LOD nodes through a simplification process. Some systems, e.g. [12], [13], are inherently forced to use fast but low-quality methods based on pure subsampling or grid-based clustering. Others, e.g. [10], [14], can use higher quality simplification methods, see also [15]. In this context, we propose a fast high quality technique which combines clustering, greedy selection, and delayed point combination. The method produces high quality simplifications on large non-uniform point clouds.

III. Multiresolution Data Structure

A. Multi-way kd-Tree

Our goal is to produce a system which is at the same time efficient, both in rendering and pre-processing, and simple to implement. We follow the approach of organizing the system around a hierarchical coarse grained structure maintained out-of-core. Most LOD point rendering approaches use either a kd-tree or a regular octree for basic hierarchical data organization.

The main benefits of octrees are that they subdivide the 3D data uniformly in space and are simple to implement. However, octrees also have a number of drawbacks, especially when considering GPU rendering constraints. First of all, octrees are inflexible, because of their fixed fanout factor and space partitioning. Due to this, no direct control over the number of internal nodes is possible. Octrees can thus be highly imbalanced and therefore, deeper than necessary. Moreover, due to the strict spatial subdivision strategy, the number of points in a (leaf) node and hence the number of nodes itself cannot be controlled directly. This leads to suboptimal distribution of points per octree node given a target VBO size, which in turn leads to increased number of context switches at runtime. Most on-board caching schemes make use of similar sized cache blocks and thus a non-uniform data distribution also implies wasted cache memory due to irregular sizes of nodes.

Although kd-trees can be constructed so as to be balanced and symmetric, they too have their limitations. Since the fanout factor is two, leaf nodes of the hierarchy cannot be made to all contain the desired number of points. Using a fixed split strategy at the middle of the bounding box produces wildly different VBO sizes, leading to the same problems encountered with octrees. By adaptively moving the split plane so as to always produce equal sized children can make the situation more controlled (e.g., see [12]). However, even in this case, VBO sizes can vary by as much as 50%.

The challenge, thus, is how a simple to implement hierarchical multiresolution data structure can be designed so as to have an even and constant point distribution among all its nodes, while being fully balanced, spatially selective and parametrized by a desired target VBO size. In our work, we propose to reach this goal by exploiting the properties of *multi-way kd-tree*, i.e., kd-trees with a fanout factor of N at each level (see also Figure 1).

In order to maintain symmetry, at each level a node is divided along its longest axis into N children containing equal number of points each. If the original model has n points, s is the target VBO size and m the number of leaf nodes in the tree then $m = n/s$. For a given N, $l = \lceil \log m / \log N \rceil$ is the maximum level or depth of the multi-way kd-tree.

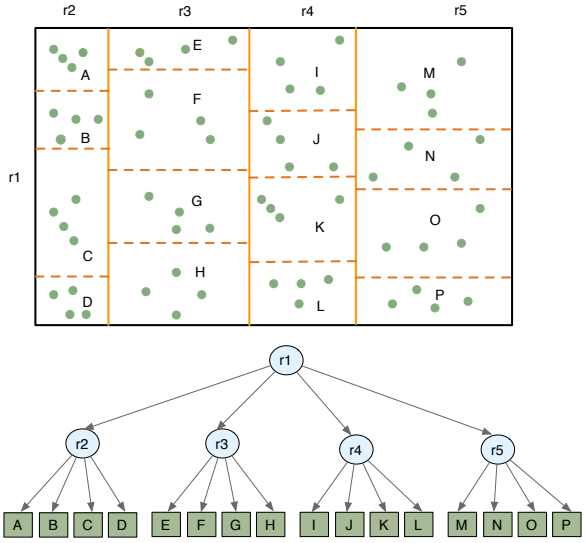


Figure 1. Multi-way kd-tree example for $N = 4$. Each of the leaf regions contains almost the same number of points.

With the proper choice of parameters s and N (see next section) one can construct an efficient multi-way kd-tree such that each node contains approximately s points. The value of N decides the trade-off between LOD adaptivity and traversal efficiency. For larger N, the tree height will be

lower and hence traversal will be faster. At the same time, an extremely shallow tree will have fewer internal nodes which represents the number of LODs that can be constructed. A multi-way kd-tree has thus the advantages that it is flexible in its fanout factor, giving more control over LOD adaptivity, it is symmetric and balanced, which allows it to be maintained easily in an array, and that it has an almost uniform point distribution close to a desired target VBO size.

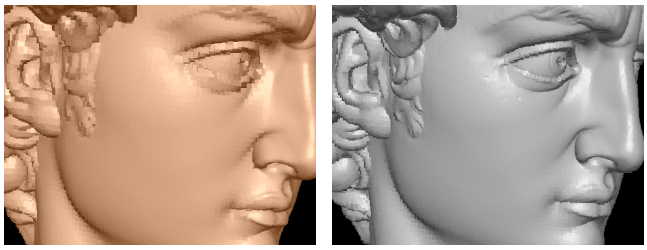
B. Multi-way kd-Tree Construction

Our procedure to construct a multi-way kd-tree is based on a top-down recursive subdivision pattern, followed by bottom-up simplification.

At each subdivision step, if the number of points to be assigned to the current node exceeds the threshold s, the longest axis of the node is determined, the points are split, and equally assigned to N child nodes. In a fully balanced tree, $\frac{n}{s}$ is a power of N and thus N can be chosen such that $\log \frac{n}{s} / \log N$ is as close as possible to an integral value. In that case, the number of points in the leaf nodes will consequently be as close as possible to the desired target VBO size s.

During bottom-up simplification, fixed point-count internal nodes are constructed from their children representation using a simplification procedure. A variety of methods can be used for this goal. Our underlying simplification approach is an extension of grid based clustering. The standard approach, used, e.g., in [13], is to apply grid based quantization to create multiresolution LOD points in each inner node of an octree. For each cell in a K^3 grid of an inner node, a representative point is computed by averaging all points belonging to the same grid cell. However, as shown in Figure 2, this simple approach may fail to produce good multiresolution representations for complex non-uniform models. Grid sampling does not take into account the spatial point density distribution and hence cannot ensure that all the inner nodes have a uniform targeted VBO size. In case of point surfaces, most of the grid cells are unoccupied, creating significantly higher density in only a few cells. Moreover, all points falling within the same cell are used to generate one LOD representative including those which do not belong to the same surface component, thereby generating poor LOD quality. This is because the method cannot identify multiple point clusters within a single grid cell.

Our approach removes the major limitations of simple point clustering. The procedure *Representatives* in Figure 4 outlines the major steps followed to form a reduced multiresolution representation of a node as it is called during tree construction by the parent in the algorithm in Figure 3. To have s lower resolution LOD points in a parent node, this basically means that s/N representative points should be selected in a node to be passed up. Though the procedure *Representatives* can yield as many representative points in a node as desired (i.e. s/N), in practice a strict bound on the node size might be replaced by a lower and upper range of size limits.



(a) (b)

Figure 2. Comparison of LOD quality using (a) a simple grid based sampling approach (e.g. similar to [13]) and (b) our adaptive clustering. Same grid resolution ($K = 45$) and maximum leaf node size (VBO = 100'000 points) used for LOD construction.

The basic idea is to cluster points not only according to distance but also normal deviation to avoid clustering of points from different surfaces. To enable fast clustering, points are first quantized to grid cells, and starting with each point being a cluster, clusters are iteratively merged within cells. Our criteria for clustering exploit the normal deviation of points. Since points within a cell are already spatially localized, we can group points differing in their normals by less than a threshold θ into the same cluster. A similar reasoning can be applied to colors. The list of clusters identified in a grid cell is pushed onto a priority queue Q which is ordered by an error metric. In order to make sure that we reach a given target number ($c \approx s/N$) of clusters, the top list of Q is removed and clusters are merged within. This process is repeated until convergence in terms of the obtained size. The LOD representatives are obtained by replacing all points in each cluster by a new aggregate whose attributes are calculated by a size-weighted average of the attributes of all point splats in the cluster and that has a span, radius size that covers the original primitives. The error metric used to prioritize Q can be the maximum normal deviation of a point from its neighbors in l, or alternatively simply the length of l works well in practice too.

```
NdTree(node)  {
  if (number of points in node ≤ s)
     return;

  determine the longest split axis of node;
  split the node data into N child nodes Nc_i;
  for all children i ≤ N
     NdTree(Nc_i);
     get reduced LOD points from child node by
     calling Representatives(Nc_i);
}
```

Figure 3. Multi-way kd-tree construction algorithm.

As stated above, the clusters within a grid cell are obtained by grouping points through normal (and eventually color) deviation. Using a fixed threshold angle θ could lead to

```
Representatives(node)  {
  initialize empty priority queue Q of list of clusters;
  set cluster count c = 0;
  setup an auxiliary K^3 grid within node;
  assign points to grid cells;
  for each grid cell
    initialize list l of clusters to be empty;
    initialize point clusters C_i and add them to l;
    increment c by |l|;
    push l onto Q;

  while (c > s/N)
    pop list l from Q;
    merge clusters in l;
    push l back to Q;
    if necessary relax threshold θ;

  for each cluster in Q
    calculate a representative LOD point;

  return c representative LOD points;
}
```

Figure 4. Representative LOD points selection.

potentially non simplified areas with a number of points higher than the target. In order to achieve a constant output size, if the number of clusters c is more than the targeted s/N, these clusters are merged using an increasingly relaxed threshold θ. This approach guarantees that lower resolutions are not too coarse or ill-formed. While other clustering and merge metrics could be applied to further improve cluster formation, the benefit of the outlined solution is that it is very simple, efficient and provides excellent rendering quality. As long as the grid size K is not extremely large or small, it does not have much impact on the LOD quality, but it may have an impact on the convergence to s/N representatives. It can though speed up the preprocessing by reducing the search space for neighbors.

C. Data Organization

After the multi-way kd-tree construction, all nodes in the tree are rearranged and kept on disk. Vertex positions and splat radii of all the points are contiguously aligned, followed by normals and, if available, color data. Each of these attributes is quantized and compressed using LZO compression. The position values (x, y, z) of a point are quantized with respect to the minimum and maximum coordinates of the bounding box of the node using 16 bits. Normals are quantized using 16 bits with a look up table that corresponds to points on a 104×104 grid on each of the 6 faces of a cube. The radius of a point is quantized with respect to the minimum and maximum splat sizes in its multi-way kd-tree node using 8 bits. In addition to this data, we separately store a small indexing structure, which saves for each node its ID, level in the multi-way kd-tree, bounding box, splats with minimum and maximum radii and size of the node file. Therefore, the multi-way kd-tree itself is small and is loaded at runtime into main memory as an array. With the help of indexing, any node in the multi-way kd-tree array can be accessed in constant time.

IV. Rendering

Our adaptive rendering subsystem loads in memory at runtime the small kd-tree indexing structure, and dynamically loads into memory and GPU the required node data.

In interactive rendering applications, it is often preferable to maintain a constant high frame rate, rather than adhering to a strict LOD requirement, especially during interactive manipulations. To optimize interactivity and LOD quality, rendering can be controlled by a rendering budget B which indicates that no more than the best B LOD points are displayed every frame. At the same time, the selection of LOD is done such that the parts of the 3D scene closer to the user are rendered at a higher resolution. This implicitly captures view dependent refinement while selecting only B points per frame for rendering.

In order to achieve the above, we maintain a priority queue Q of LOD nodes at runtime which is ordered by the LOD metric for refinement or coarsening. Several metrics such as the projected size of LOD points in pixels on screen could be used. With the goal of rendering on a budget, our LOD selection is based on the difference of the LOD level l of a node and the maximum leaf level l_{max} in the tree and perspective division by the distance d to the viewer, i.e., $\varepsilon_l = \frac{c_0(l_{max}-l+1)}{c_1 d + c_2 d^2}$, with parametrization constants c_i. For rendering on a budget, this LOD selection metric works similar to a projected screen-space LOD point size measure but is more efficient to incrementally adjust the rendering front.

The algorithm in Figure 5 outlines the basic steps to compute the current rendering front given a budget B, the fanout factor N and VBO size s. When a node on level l is added to the priority queue Q, its LOD metric ε_l is computed for prioritization. The priority queue Q holds the currently selected nodes for display which are incrementally refined in the *while* loop according to the budget availability.

```
LODSelection()  {
  initialize empty queue Q prioritized on ε_l;
  push the root node r onto Q;
  count = |r|;
  while (count − |n| + Ns ≲ B and Q is not empty)
    pop node n from Q;
    if (n is not a leaf)
      count = count − |n|;
      foreach child c of n
        push c onto Q, prioritized on ε_l;
        count = count + |c|;
    else
      add n to rendering front;

  if Q is not empty
    add nodes from Q to rendering front;
}
```

Figure 5. LOD selection for rendering on a budget.

Note however, that our rendering system also supports view-dependent LOD selection based on projected screen-space LOD point pixel size. For that purpose, the LOD nodes, e.g. of the past frame's rendering front, are refined or coarsened based on projected pixel size and no budget limit is included.

LOD changes often occur gradually in interactive rendering. In case of unavailability of a new LOD node and to avoid its synchronous fetching, which could lead to jitters, most refinement and coarsening changes can be delayed by one or a very few frames until the new LOD data has been fetched from disk. This is accomplished by running an asynchronous server thread concurrently to the main rendering thread. To support this strategy, we perform incremental frame-to-frame updates as follows, similar to [16]. All nodes currently selected by *LODSelection* constitute the active rendering *front* for the current frame. These nodes can be ordered on the front through the LOD hierarchy and compared to the last one as shown in Figure 6. All node transitions from the last to the current front can be grouped into four types:

1) Old Node $\rightarrow$ No node (a $\rightarrow \phi$)
2) No node $\rightarrow$ New node ($\phi \rightarrow$ l)
3) Ancestor $\rightarrow$ Children (i $\rightarrow$ j, k)
4) Children $\rightarrow$ Ancestor (e,f $\rightarrow$ d)

Out of these, Case 1 is trivial, 3 and 4 can be addressed by asynchronous fetching if necessary, and only Case 2 needs a synchronous load operation. Cases 3 and 4 can hence be treated by rendering the previous frame's respective LOD nodes while the asynchronous load from disk takes place. Thus instead of fetching a new node synchronously, its lower or higher resolution which has been used before is rendered until the new desired LOD is available. The number of forced synchronous fetches in Case 2 can further be reduced by fetching parent nodes corresponding to every group of new sibling nodes while pushing the actual nodes to the asynchronous thread. In the worst case, this might slow down the application by a few frames but guarantees absence of any holes. Further pre-fetching at view-frustum clip boundaries can further reduce synchronous loads.

The basic rendering engine described above is simple to extend, in order to increase performance, with other acceleration techniques. In particular, we have integrated backface culling, using normal cone for each node in tree and occlusion culling, similar to [12] to avoid rendering of invisible points. Unused point budget reclaimed from the occluded or backfacing LOD nodes can be reused to refine some more nodes of a coarser LOD resolution to a finer one. Higher quality rendering than simple one-pass point splatting can also be obtained by smoothly interpolating among points. In particular, our system can employ the deferred blending approach as introduced in [17].

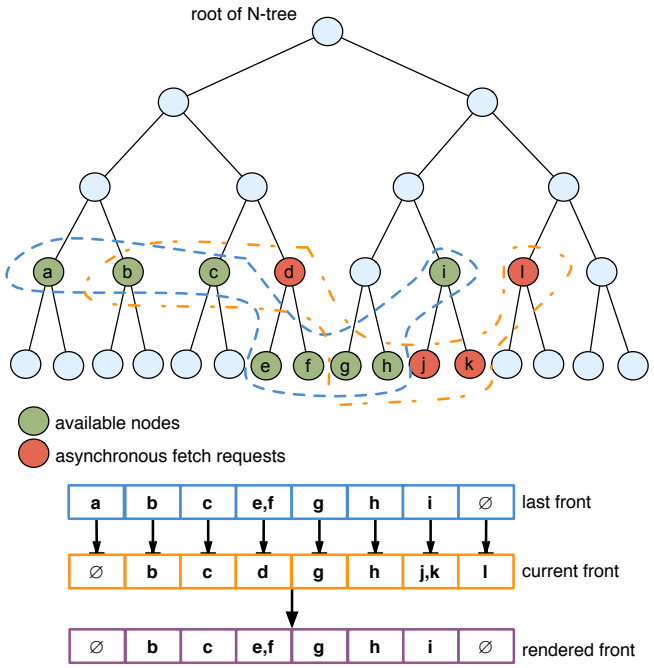


Figure 6. LOD update and asynchronous fetching example for a multi-way kd-tree with $N = 2$.

V. Results

The proposed method has been implemented in C++ using OpenGL and GLUT. Unless otherwise specified, the results have been evaluated on a system with 2x2.66 GHz Dual-Core Intel Xeon processors and NVIDIA GeForce 8800 GT graphics and a display window of 1024×1024 pixels.

A. Preprocessing

The input given to our preprocessing unit is the point cloud containing point attributes like position, splat size, color and normal. In Table I, we report preprocessing statistics for a number of large scale datasets. Statistics include the time required for building the multiresolution model and compressing it, as well as the disk usage (uncompressed and compressed) for the given value of N. Throughout all experiments, the grid size used to support clustering was $K = 35$. As we can see, the preprocessing time depends to some extent on the fanout factor N, and hence number of internal nodes, but in general our approach can preprocess about 40K to 60K points/second.

B. Rendering

Figure 7 shows different views of our large test models, demonstrating the LOD mechanism producing constant quality renderings at varying zoom-in levels by rendering on a budget of $B = 3$M to 5M. Figure 8 shows the comparison of sampling and rendering quality depending on the choice of LOD tree data structure, i.e. using octrees, binary kd-trees or multi-way kd-trees. Our multi-way kd-tree clearly

Model	#Samples	N	Time (s)	Disk Usage (MB)		
				In	Out	Comp.
David 2mm	4129614	3	71	158	229	37
Lucy	14027872	2	310	535	757	143
David 1mm	28184526	5	447	1127	1525	225
St. Matthew	186850683	3	4915	7473	10808	1652
Pisa Cathedral	368585469	4	9044	14743	20937	2973

Table I
PREPROCESS MEASUREMENTS.

outperforms the octree due to better sampling adaptivity and is at least equally as good as a (binary) kd-tree. Speed performance is discussed below.

We benchmarked our implementation for various chunk sizes and thus fanout values N with the St. Matthews model with simple OpenGL points as drawing primitives. Table II shows the tree structure information for various configurations indicating fanout factor N, number of points in VBO, as well as number of levels and nodes in the multi-way kd-Tree. Rendering rates for these configurations are given in Figure 9. This is further verified by comparing our multi-way kd-tree also with octree and kd-tree for point throughput for similar maximal node sizes (see also Table IV).

Fanout (N)	VBO size	Levels	Nodes
6	4009	7	55987
6	24029	6	9331
3	77822	8	3280
7	85437	5	2801
8	364943	4	585

Table II
MULTI-WAY KD-TREE STRUCTURE FOR THE ST. MATTHEW MODEL USING DIFFERENT VBO SIZES AND FANOUT FACTOR N.

As is clear from Table III, rendering efficiency in terms of frames per second and points per second is quite similar for all models despite them varying significantly in size. We achieve overall rendering rates of nearly 290M points per second with peaks exceeding 330M even for the larger datasets. We benchmarked two other state-of-the-art chunk-based systems [12], [14] on the same models and rendering hardware, and obtained very similar results (within $\pm 5\%$). Our new framework offers more flexibility, as it adds the ability to control tree depth, and thus, the number of refinement steps required to achieve the required resolution.

Our compression scheme is conservative in terms of disk space and could be improved by compressing refined point splats relative to their coarser parent points. However, our experiments have shown that performance (frame and point throughput rates) and display quality are hardly affected at all due to the current compression, even though decompression is performed on the CPU. Hence, the simple compression technique used is a good trade-off for storage space versus rendering efficiency.

Model	#Samples	N	VBO Size (K)	Fps	Pps (M)
David 2mm	4129614	3	51	87	264
Lucy	14027872	2	55	85	262
David 1mm	28184526	5	45	86	265
St. Matthew	186850683	3	85	87	265
Pisa Cathedral	368585469	4	90	88	282

Table III
RENDERING PERFORMANCE STATISTICS FOR VARIOUS MODELS AND VBO SIZES, GIVEN A RENDERING BUDGET OF $B = 3$M.

In Table IV, we summarize the rendering performance as a function of different LOD tree construction approaches (see also Figure 8). It shows that an optimum display quality and rendering speed can be obtained with moderate VBO sizes, while highest point throughput is achieved with large VBOs.

VBO Size	Octree		Kd-Tree		MW Kd-Tree	
	Fps	Pps	Fps	Pps	Fps	Pps
24029	115	118	125	203	139	229
77822	122	185	113	224	92	237
85437	119	210	113	224	84	251

Table IV
COMPARISON OF PERFORMANCE BETWEEN OCTREE, KD-TREE AND MULTI-WAY KD-TREE USING THE ST. MATTHEW MODEL AND PIXEL ERROR THRESHOLD OF 3.5.

VI. CONCLUSION AND FUTURE WORK

We have presented a simple and efficient framework for hierarchical multiresolution preprocessing and rendering of massive point cloud datasets. We have demonstrated that our novel point hierarchy definition is flexible in that it can adapt to a desired LOD granularity by adjusting its fanout factor N, that we can target specific rendering-efficient VBO sizes and that our algorithm supports adaptive out-of-core rendering, featuring asynchronous prefetching and loading from disk as well as rendering on a budget. Our future work will concentrate on improving rendering quality through geo-morphing, increasing the compression factor, and implementing networked rendering.

ACKNOWLEDGMENT

We are grateful to the Stanford Graphics Group, The Digital Michelangelo Project, and the ISTI-CNR Visual Computing Group for making benchmark datasets available.

REFERENCES

[1] M. Levoy and T. Whitted, "The use of points as display primitives," Department of Computer Science, University of North Carolina at Chapel Hill, Tech. Rep. TR 85-022, 1985.

[2] J. Grossman and W. J. Dally, "Point sample rendering," in *Proceedings Eurographics Workshop on Rendering*. Eurographics, 1998, pp. 181–192.

Figure 7. Varying zoom views of the David (28M samples), St. Matthews (187M samples) and Pisa Cathedral (368M samples) models.

[3] H. Pfister and M. Gross, "Point-based computer graphics," *IEEE Computer Graphics and Applications*, vol. 24, no. 4, pp. 22–23, July/August 2004.

[4] M. H. Gross, "Getting to the point...?" *IEEE Computer Graphics and Applications*, vol. 26, no. 5, pp. 96–99, September/October 2006.

[5] M. H. Gross and H. Pfister, Eds., *Point-Based Graphics*, ser. Series in Computer Graphics. Morgan Kaufmann Publishers, 2007.

[6] M. Sainz and R. Pajarola, "Point-based rendering techniques," *Computers & Graphics*, vol. 28, no. 6, pp. 869–879, 2004.

[7] L. Kobbelt and M. Botsch, "A survey of point-based techniques in computer graphics," *Computers & Graphics*, vol. 28, no. 6, pp. 801–814, 2004.

[8] S. Rusinkiewicz and M. Levoy, "QSplat: A multiresolution point rendering system for large meshes," in *Proceedings ACM SIGGRAPH*. ACM SIGGRAPH, 2000, pp. 343–352.

[9] C. Dachsbacher, C. Vogelgsang, and M. Stamminger, "Se-

Figure 8. Comparison of LOD sampling quality depending on the choice of LOD tree data structure, i.e. octree, kd-tree or multi-way kd-tree (f.l.t.r.), using similar maximal node sizes.

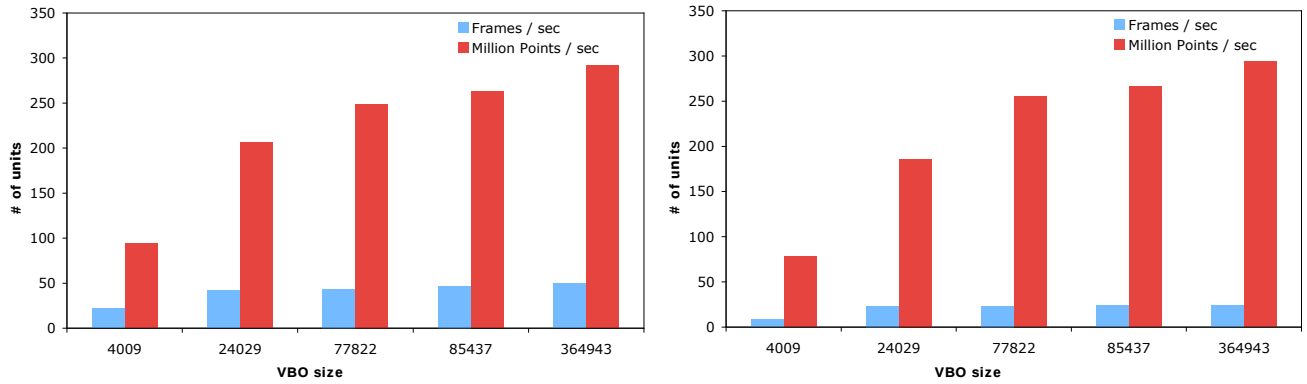


Figure 9. Rendering rates for rendering on a budget B of (a) 6 million and (b) 12 million points.

quential point trees," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 657–662, 2003.

[10] R. Pajarola, M. Sainz, and R. Lario, "XSplat: External memory multiresolution point visualization," in *Proceedings IASTED International Conference on Visualization, Imaging and Image Processing (VIIP)*, 2005, pp. 628–633.

[11] M. Wimmer and C. Scheiblauer, "Instant points : Fast rendering of unprocessed point clouds," in *In Proceedings Symposium on Point-Based Graphics*, July 2006, pp. 129–136.

[12] E. Gobbetti and F. Marton, "Layered point clouds," in *Proceedings Eurographics/IEEE VGTC Symposium on Point-Based Graphics*, 2004, pp. 113–120.

[13] M. Wand, A. Berner, M. Bokeloh, A. Fleck, M. Hoffmann, P. Jenke, B. Maier, D. Staneker, and A. Schilling, "Interactive editing of large point clouds," in *Proceedings Eurographics/IEEE VGTC Symposium on Point-Based Graphics*, 2007, pp. 37–46.

[14] F. Bettio, E. Gobbetti, F. Marton, A. Tinti, E. Merella, and R. Combet, "A point-based system for local and remote exploration of dense 3D scanned models," in *Proceedings Eurographics Symposium on Virtual Reality, Archaeology and Cultural Heritage*, 2009, pp. 25–32.

[15] M. Pauly, M. Gross, and L. P. Kobbelt, "Efficient simplification of point-sampled surfaces," in *Proceedings IEEE Visualization*. Computer Society Press, 2002, pp. 163–170.

[16] J. Boesch, P. Goswami, and R. Pajarola, "Raster : Simple and efficient terrain rendering on gpu," in *Proceedings Eurographics Area Paper, Scientific Visulization*, 2009, pp. 35–42.

[17] Y. Zhang and R. Pajarola, "Deferred blending: Image composition for single-pass point rendering," *Computers and Graphics*, vol. 31, pp. 175–189, 2007.