Concomitant Hierarchy Construction and Rendering of Large Point Clouds

Vinícius da Silva*§, Ricardo Guerra Marroquim†, Claudio Esperança‡
dsilva.vinicius@gmail.com, marroquim@cos.ufrj.br, esperanc@cos.ufrj.br
*Institute for Pure and Applied Mathematics (IMPA). VISGRAF Lab.
Estrada Dona Castorina, 110, Jardim Botânico, Rio de Janeiro - RJ, Brazil, CEP: 22460-320
†‡Federal University of Rio de Janeiro (UFRJ). Computer Graphics Lab (LCG).
Cidade Universitria, Technology Center, Block H, Rio de Janeiro - RJ, Brazil, CEP: 21941-972

***Abstract*—Rendering large point clouds ordinarily requires building a hierarchical data structure for accessing the points that best represent the object for a given viewing frustum and level-of-detail. The building of such data structures frequently represents a large portion of the cost of the rendering pipeline both in terms of time and space complexity, especially when rendering is done for inspection purposes only. In this work we present OMiCroN – Oblique Multipass Hierarchy Creation while Navigating – which is the first algorithm capable of immediately displaying partial renders of the geometry, provided the cloud is made available sorted in Morton order. In fact, a pipeline coupling OMiCroN with an incremental sorting algorithm running in parallel can start rendering as soon as the first sorted prefix is produced, making this setup very convenient for streamed viewing.**

I. Introduction

In recent years, improvements in acquisition devices and techniques have led to the creation of huge point cloud datasets. Direct rendering of such datasets must resort to indexing data structures. In many use cases, the cost of building such structures is not critical for the task at hand and need not justify arbitrarily long preprocessing times (e.g. [1], [2]). In other cases, shortening the time to produce the hierarchy is deemed worthwhile, at the expense of achieving slightly worse balance or render quality (e.g. collision detection [3]).

In this paper we introduce OMiCroN (Oblique Multipass Hierarchy Creation while Navigating), a new take on the problem of shortening the delay between point cloud acquisition and its visualization. The technical contributions of this work are the introduction of Hierarchy Oblique Cuts, allowing parallel data sorting, spatial hierarchy construction and rendering; restriction of the preprocessing to a very fast and flexible Morton code based partial sort; on-the-fly Octree construction for large point clouds; full detail rendering of the data from the very beginning, following the Morton Order; immediate visual feedback of the hierarchy creation process.

II. Background

Our work depends on three major concepts: Morton Order; Hierarchical Spatial Data Structures; and Rendering Fronts. The theory behind them is summarized in this section.

§Ph.D. Thesis author.

Morton Order and Hierarchical Spatial Data Structures: Morton [4] proposed a linearization of 2D grids, later generalized to n-dimensional grids. It results in a z-shaped space-filling curve, called the Z-order curve. The order in which the grid cells are visited by following this curve is called Morton order or Z-order. The associated Morton code for each cell can be computed directly from the grid coordinates by interleaving their bits. Morton codes extend naturally to regular spatial subdivision schemes, thus they are usually used in conjunction with Hierarchical Spatial Data Structures such as Octrees and regular Kd-trees (Bintrees). They provide fast data culling and a direct level-of-detail structure, by mapping the n-dimensional structure to a one-dimensional list.

Rendering Front: A Rendering Front, hence called only Front, is a structure to optimize sequential traversals of hierarchies, and has been used in many works [5]–[8]. Instead of starting the traversal at the root node for every new frame, it starts at the nodes where it stopped in the preceding frame. Fronts have two basic operators: *prune* and *branch*. The *prune* operator traverses the hierarchy up and the *branch* operator works in the opposite direction.

III. Related work

While the use of points as rendering primitives was introduced very early in Computer Graphics [9], [10], their widespread adoption only occurred much later, as discussed on extensive survey literature [11]–[16]. Here we focus the discussion on multiresolution and LOD structures, establishing an argument for why a stream-and-feedback-based algorithm such as OMiCroN is a desirable tool for the academy and industry.

QSplat [1] is the seminal reference on large point cloud rendering. It is based on an out-of-core hierarchy of bounding spheres, which is traversed to render the points. Since its main limitation is the extensive CPU usage, QSplat was followed by techniques that load more work onto the GPU. For example, Sequential Point Trees [17] introduced adaptive rendering completely on the graphics card by defining a new octree linearization. Other methods used approaches relying on the out-of-core paradigm, such as XSplat [18] and Instant Points [2]. XSplat proposed a paginated multiresolution point-octree hierarchy with virtual memory mapping, while

Instant Points extended Sequential Point Trees by nesting linearized octrees to define an out-of-core system. Layered Point Clouds [19] proposed a binary tree of precomputed object-space point cloud blocks that is traversed to adapt sample densities according to the projected size in the image. Wand et al. [20] presented an out-of-core octree-based renderer capable of editing large point clouds and Bettio et al. [21] implemented a kd-tree-based system for network distribution, exploration and linkage of multimedia layers in large point clouds. Other works focused on parallelism using multiple machines to speed-up large model processing or to render on wall displays using triangles, points, or both [22]–[26].

More recently, relatively few works have focused on further improving the rendering of large point clouds, such as the method by Lukac et al. [27]. Instead, more effort has been concentrated on using established techniques in domains that require the visualization of large datasets as a tool for other purposes. For example, city visualization using aerial LIDAR [28], [29], sonar data visualization [30] and, more prominently, virtual reality [31]–[34].

While the aforementioned papers present very useful and clever methods to implement or use large point cloud rendering, none of them considers presenting data to the user before the full hierarchy is created.

IV. Overview

Rendering a hierarchy while it is under construction is a non-trivial synchronization problem. Since a rendering front can potentially have access to any node in the hierarchy, the use of locks might lead to prohibitive performance. We propose to synchronize those tasks using specific Morton Curve and Morton Code properties to classify nodes in all curves composing a hierarchy. This classification is based on an Oblique Hierarchy Cut, a novel data-structure to represent hierarchies under construction. Nodes inside an Oblique Cut are guaranteed to be rendered without interference of the construction and vice-versa. An overview of the idea can be seen in Figure 1.

To evaluate if a node is inside an Oblique Cut we need a methodology that is consistent for all curves at different hierarchy levels. One that makes sense is to consider a node inside the cut if all of its descendants are also inside it. Thus, we need a proper way to relate nodes at Morton Curves at different levels of the hierarchy. For that purpose, let $span(x)$ be a function that returns the Morton Code of the right-most descendant of a supposedly full subtree rooted by x. With this definition $span$ has several useful properties. First, it conceptually maps nodes in any hierarchy level with other ones at the deepest level. Thus, it also maps any Morton Curve to the Morton Curve at that level. Not only this, but by definition $span(y) <= span(x)$, for any descendant y of x. Figure 2 shows how $span$ works.

V. Oblique Hierarchy Cuts

In this section we describe the Oblique Cuts in detail. Given a conceptual expected hierarchy H, with depth l_{max}, an Oblique Hierarchy Cut C consists of a delimiting Morton code m_C and a set of lists $L_C = \{L_{C,k}, L_{C,k+1}...L_{C,l_{max}}\}$, where k is the shallowest level of the hierarchy present in the cut. Each node N is uniquely identified by its Morton code m_N and these two concepts are interchangeable from now on. Figure 3 contains a schematic view of the data structure. We now formally define the two operators, *concatenate* and *fix*, as well as the important concept of *Placeholder* nodes.

A. Operator Concatenate

The operator *concatenate* is defined as $C' = concatenate(C, \{x_0, ..., x_n\})$, where $m_C < x_0 < ... < x_n$. This operator incorporates new l_{max} level leaf nodes $\{x_0, ..., x_n\}$ to C, resulting in a new cut C'. The operator itself is simple and consists of concatenating all new nodes into list $L_{C,l_{max}}$. This operator is illustrated in Figure 3.

B. Operator Fix

The operator *fix* definition is $C'' = fix(C')$. Its purpose is to insert the ancestors of $\{x_0, ..., x_n\}$ that should be in subtrees in $L_{C'}$. To achieve this, it suffices to find an ancestor set S where $span(S) > m_C$. To identify S, the lists are processed bottom-up, in Morton order (see Figure 3).

C. Placeholders

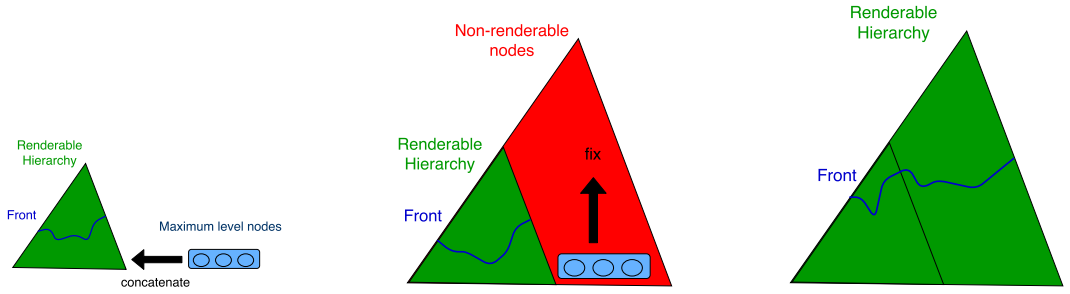
According to the aforementioned definition of Oblique Hierarchy Cut, H can only have leaves at level l_{max}, since the *concatenate* operator only inserts nodes at that level. Leaves could be inserted into other levels directly, but it would make it difficult for *fix* to efficiently maintain morton order. To address this issue, the concept of *placeholder* is defined. A *placeholder* is an empty node at a given level representing a node at a shallower level. More precisely, given a node N at level l, its placeholder $P_{N,l+1}$ at level $l+1$ is defined as the rightmost possible child of N. Note that, with this definition, $P_{N,l_{max}}$ has Morton code $span(m_N)$.

A leaf X in H with level $l < l_{max}$ is represented by placeholder $P_{X,i}$ such that $l < i \leq l_{max}$ when inserting the subtree of level i at $L_{C'_i}$. Placeholders are used as roots of degenerate subtrees, since there is no purpose for them inside subtrees. Even if not meaningful for H, placeholders ensure morton order in *fix* until level l is reached.

Intuitively, a sequence of Oblique Hierarchy Cuts C_i resulting from sequentially applying operators *concatenate* and *fix* until no more leaf nodes or placeholders are left for insertion results in an oblique sweep of H.

VI. Oblique Hierarchy Cut Front

Concomitantly with the building of H with progressive oblique cuts, a rendering process might be traversing the already processed portions of H with the help of a front (see Figures 1 and 4). Thus, for a given Oblique Hierarchy Cut C, the rendering process will adaptively maintain a front F_C restricted to the renderable part of H. In order to ensure proper independence of F_C with respect to C, the nodes in the front must be in morton order (so siblings are adjacents



(a) Initial (possibly empty) renderable hierarchy and *concatenate* operator.

(b) The *fix* operator: node ancestors are inserted into the hierarchy.

(c) After the *fix* operation the renderable hierarchy is expanded.

Fig. 1. OMiCroN overview. A renderable hierarchy is maintained while inserting incoming nodes in parallel. This cycle is repeated until the whole hierarchy is constructed.

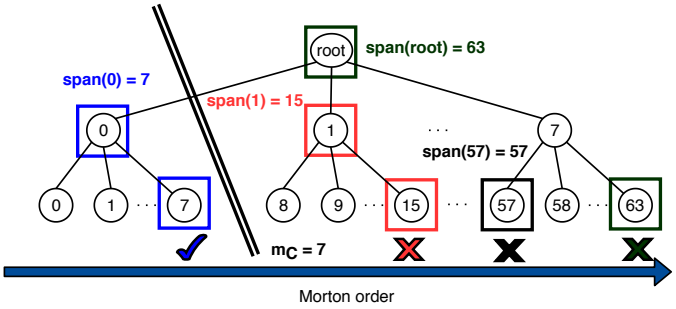


Fig. 2. $span$. In the example, the cut is defined by the delimiting Morton Code $m_C = 7$, defined at the deepest level. Each pair of colored squares shows the input and result of $span$. The blue square case is inside the cut because $span(0) = 7 <= 7$. The other cases (red, green and black) are outside of the cut because $span(x) > 7$. It is important to note that the operation is defined for any level of the hierarchy, even for nodes at the deepest level, where $span(x) = x$.

and prune is trivial) and the roots of subtrees in L_C cannot enter the Front (so roots moved by fix do not interfere in rendering). Similarly, placeholders cannot be pruned either since their parents might not yet be defined. An example of a valid Oblique Hierarchy Cut Front is given in Figure 4.

In summary, the evaluation of an Oblique Hierarchy Cut Front consists of three steps:

1) Concatenate new placeholders into the front.
2) Choose the hierarchy level l where candidates for substituting placeholders in the front are to be sought.
3) Iterate over all front nodes, testing whether they are placeholders that can be substituted, and whether they need to be pruned, branched or rendered.

A. Insertion of new nodes

Since the root of H is only available after all sequential cuts are evaluated, the usual front initialization is not possible for F_C. In order to simplify leaf and placeholder insertion and substitution, all leaves are first inserted in the front as placeholders and saved in a per-level list of leaves to be replaced. One main reason for this duplication is that new nodes are always inserted as roots in $L_{C,l_{max}}$, and cannot enter the front. Thus, placeholders mark their position until the *fix* operator moves them to other subtrees.

B. Substitution of placeholders

Since the leaf lists are organized by level, and the placeholders and leaves are respectively inserted into the front and into the lists in Morton order, a very simple and efficient substitution scheme is proposed. Given a placeholder and a substitution level l, it consists in verifying if the first element in the leaf list of level l is an ancestor of the placeholder. If it is, the leaf is removed from the substitution list and replaces the placeholder in the front. Since comparison of Morton codes is a fast $O(1)$ operation, the entire placeholder substitution algorithm is also $O(1)$. Keeping in mind that for each front evaluation a single level l will be checked for substitution, all leaves at that level are guaranteed to be substituted in a single frame.

C. Choice of substitution level

In order to maximize node substitution, l is chosen as the level with most insertions. This is an obvious choice, since the list will be completely emptied after the evaluation, so we are substituting the maximum number of placeholders in one iteration. The nodes not substituted in the current front evaluation are ignored since their corresponding leaves are not in level l. However, the algorithm guarantees that all currently inserted leaves will substitute their placeholders in the next $l_{max} - 1$ front evaluations at max. Thus, the delay to starting rendering a leaf node after insertion is minimal.

D. Leaf collapse

In order to maintain the use of main memory within a given budget, it is also possible to enable a very simple optimization, called *Leaf Collapse*. This optimization removes all leaves at level l_{max} which form a chain structure with their parents, i.e., leaves that do not have siblings.

VII. Experiments

The prototype implementation was tested using four point cloud datasets obtained at the Digital Michelangelo Project page: David (469M points, 11.2GB), Atlas (255M points,

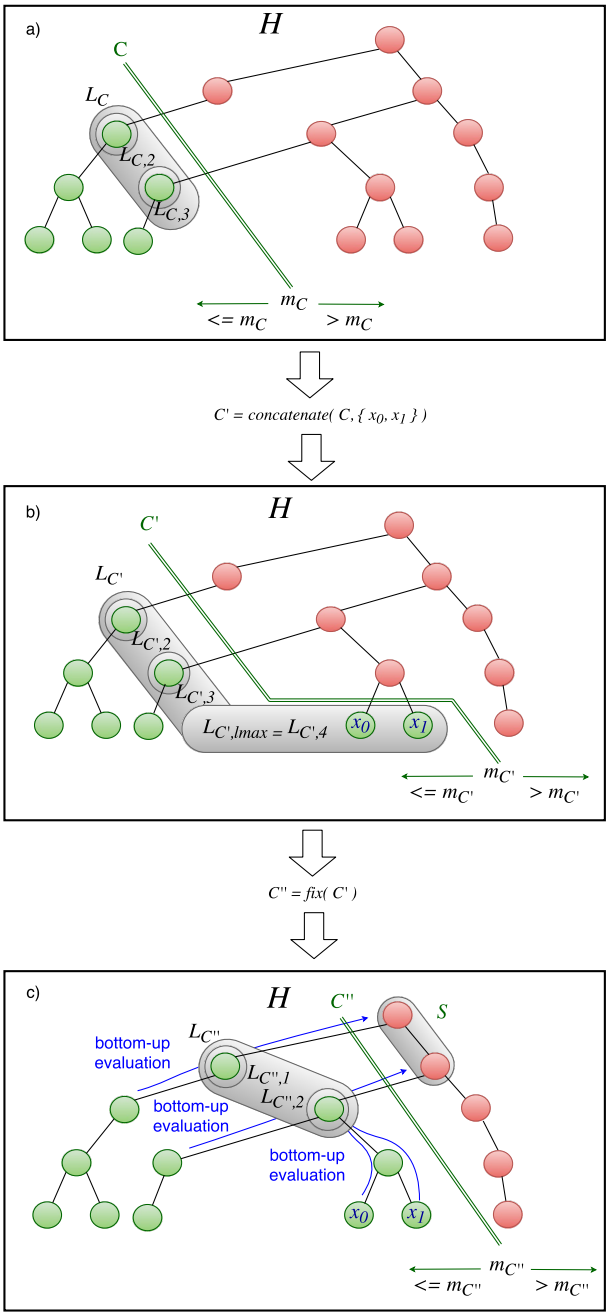


Fig. 3. Oblique Hierarchy Cut and operators *concatenate* and *fix*. A cut C is defined by a delimiting Morton code m_C and a list of roots per level L_C (a). The green color represent nodes already created and inside the cut. The red color indicates nodes not created yet, which exist only in the conceptual expected hierarchy H. The *concatenate* operator inserts new roots x_0 and x_1 at the deepest level l_{max}, resulting in cut C' (b). Then, operator *fix* traverses subtrees bottom-up, creating parents until the boundary S is reached.

6.1GB), St. Mathew (187M points, 4.5GB) and Duomo (100M points, 2.4GB). The maximum hierarchy depth was set to 7 to ensure memory footprints compatible with available memory and swap area. Coordinates in all datasets were normalized to range $[0, 1]$.

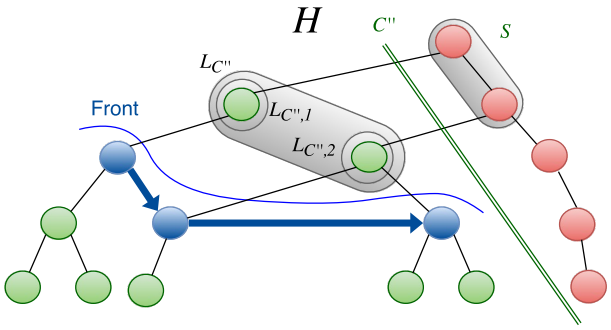


Fig. 4. Example of valid Oblique Hierarchy Cut Front. The direction of the blue arrows indicate the order restriction. The nodes in the front cannot be roots in $L_{C''}$.

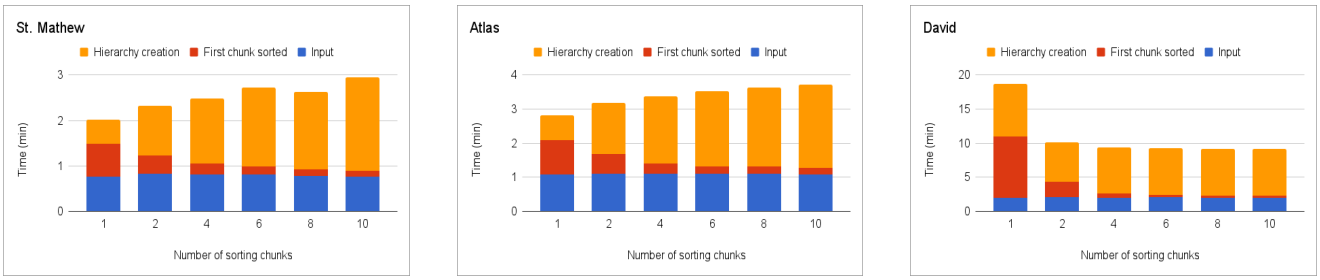
A. Rendering latency tests

In order to assess the actual delay from the moment the raw unsorted collection of points is available, and the moment where rendering actually starts, we must consider the sorting process in some depth. Our testbed consists of a desktop computer with an Intel Core i7-3820 processor with 16GB memory, NVidia GeForce GTX 750 and a SanDisk 120GB SSD. The same SSD is used for swap and I/O.

The first experiment consists of consecutively sorting and streaming chunks of the input to OMiCroN. Parallel rendering and leaf collapse are enabled for these tests. Using more chunks allows rendering to start earlier, as shown in Figure 5. In particular, increasing the number of sorting chunks can improve the time between the moment input finishes and rendering starts from 5 to 31 times, depending on the size of the dataset. For large datasets, the partial sort can diminish the use of swap during sort and hierarchy creation, resulting in better timings in all aspects, as Figure 5c demonstrates.

The second experiment consists of profiling and comparing OMiCroN with the parallel rendering activated and deactivated at hierarchy creation time, also evaluating the system core usage while running the algorithm. The input for this test consists of the datasets already sorted in Morton order and the data is streamed directly from disk. Leaf collapse is disabled. Figure 6 shows the results. The overhead imposed is between 20% (David) and 34% (St.Mathew), which is an evidence that the overhead impact decreases as the dataset size increases. The final observation from this experiment is that OMiCroN maintains the usage of all 8 logical cores near 90% with peaks of 100% for the entire hierarchy creation procedure.

The third experiment's purpose is to generate data for better understanding the hierarchy creation progression over time. It consists of measuring the time needed to achieve percentile milestones of hierarchy creation. For this test, the sorted data is streamed directly from disk, parallel rendering is enabled and leaf collapse is disabled unless pointed otherwise. The results are presented in Figure 7. We can conclude that the hierarchy construction has the expected linear progression.



(a) St. Mathew. (b) Atlas (c) David

Fig. 5. Impact of the number of sort chunks. After a constant time spent reading the input (blue), the first chunk is sorted (red), starting the parallel hierarchy creation and rendering (orange). The first column in all charts corresponds to the case where all input is sorted before the hierarchy creation begins.

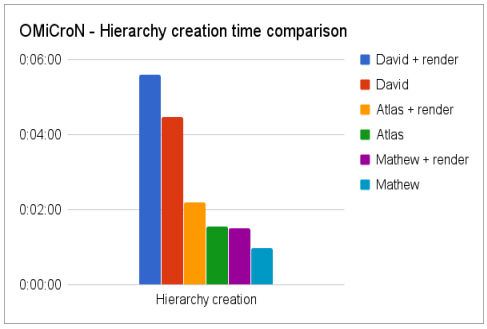


Fig. 6. Comparison of hierarchy creation with and without parallel rendering. Sorted data is streamed directly from disk. The overhead imposed by parallel rendering is between 20% (David) and 34% (St. Mathew).

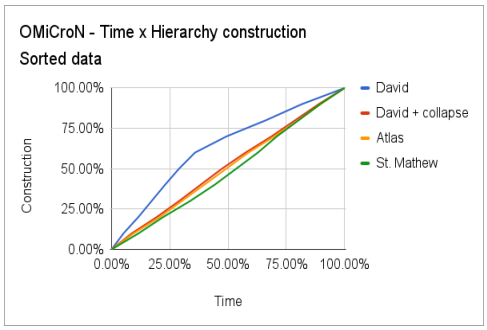


Fig. 7. Hierarchy creation over time. Sorted data is streamed directly from disk, parallel rendering is enabled and leaf collapse is disabled unless pointed otherwise.

B. Hierarchy creation and rendering

A second set of experiments were conducted to assess OMiCroN's behavior in terms of memory usage and performance. All experiments in this set read a sorted dataset directly from disk. The test system had an Intel Core i7-6700, 16GB memory, NVidia GeForce GTX 1070, and secondary SSD storage with roughly 130 MB/s reading speed. Two main parameters impact OMiCroN's memory footprint: *Leaf Collapse* optimization and parent to children point ratio, as shown in Table I. These also impact the reconstruction quality of the algorithm as can be seen in Figure 8.

TABLE I
RELATIONSHIP BETWEEN THE ALGORITHM RECONSTRUCTION PARAMETERS – LEAF COLLAPSE, PARENT TO CHILDREN RATIO – AND MEMORY FOOTPRINT, TOTAL HIERARCHY CREATION TIMES, AND AVERAGE CPU USAGE PER FRAME.

Model	Coll	Ratio	Mem	Creation	CPU
David	On	0.2	8.5GB	146.3s	7.6ms
David	On	0.25	9.9GB	151.2s	8.8ms
David	Off	0.2	21GB	229.8s	16.7ms
Atlas	On	0.2	2.3GB	77.8s	11.9ms
Atlas	On	0.25	3.0GB	81.9s	11.0ms
Atlas	Off	0.2	11.5GB	120.8s	16.2ms
Mathew	On	0.2	1.7GB	59.6s	13.7ms
Mathew	On	0.25	2.2GB	60.9s	11.6ms
Mathew	Off	0.2	8.4GB	80.6s	25ms
Duomo	On	0.2	0.9GB	31.0s	18.2ms
Duomo	On	0.25	1.2GB	32.6s	23.1ms
Duomo	Off	0.2	4.5GB	40.0s	21.9ms

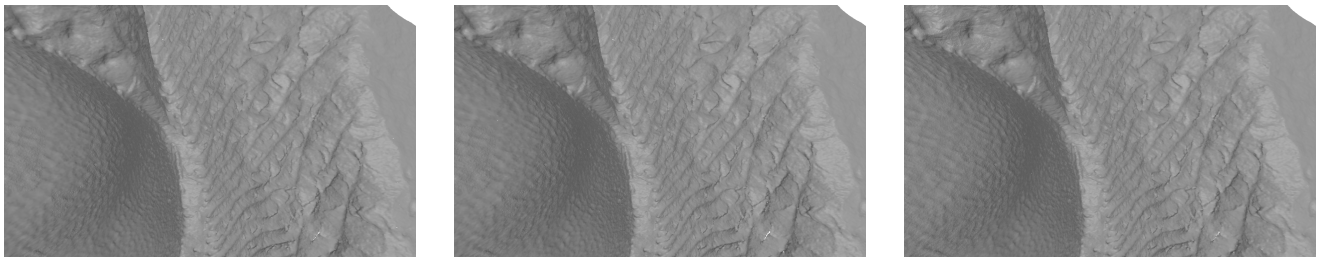
Even though limited to datasets that fit in RAM unless swap space is used, OMiCroN can be set up to fit a broad range of memory budgets maintaining rendering quality.

C. Comparisons

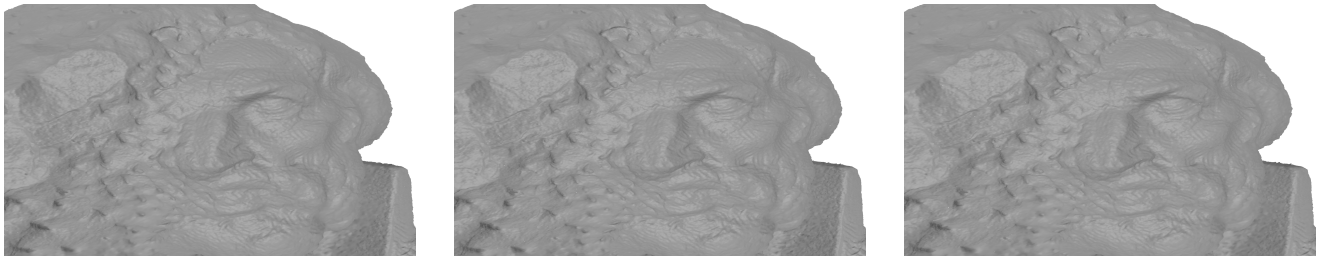
We also found it useful to compare OMiCroN with other algorithms that create hierarchies for large datasets. To this end, we evaluated the hierarchy creation algorithm used in the large point cloud renderer Potree [35]. The methodology was to compare the best cases in Figures 5a, 5b and 5c, which include input, sorting, hierarchy creation and rendering, and the timings reported by Potree, which include input and hierarchy creation. All tests created hierarchies with depth 7.

Figure 9 shows the results for St. Matthew, Atlas and David. OMiCroN is more than 2 times faster for David and more than

(a) David, leaf collapse on, 0.2 point ratio. (b) David, leaf collapse on, 0.25 point ratio. (c) David, leaf collapse off, 0.25 point ratio.



(d) Atlas, leaf collapse on, 0.2 point ratio. (e) Atlas, leaf collapse on, 0.25 point ratio. (f) Atlas, leaf collapse off, 0.25 point ratio.



(g) St. Mathew, leaf collapse on, 0.2 point ratio. (h) St. Mathew, leaf collapse on, 0.25 point ratio. (i) St. Mathew, leaf collapse off, 0.25 point ratio.

(j) Duomo, leaf collapse on, 0.2 point ratio. (k) Duomo, leaf collapse on, 0.25 point ratio. (l) Duomo, leaf collapse off, 0.25 point ratio.

Fig. 8. Rendering comparison of hierarchies with different leaf collapse and parent to children point ratio parameters. As can be seen from items (a) to (i), the final reconstructions are very detailed even at close range and the differences when the leaf collapse is turned on are almost imperceptible for the David, Atlas and St. Mathew datasets. The hierarchy for Duomo suffers from lack of density when leaf collapse is turned on because the dataset itself has smaller density in comparison with the others.

4 times faster for St. Matthew and Atlas. An important detail is that Potree reports creating a hierarchy with only 68% of the input points for David, whereas St. Matthew and Atlas result in 100% of input points usage.

VIII. Final remarks

In this work, we presented OMiCroN, a flexible and generic algorithm for rendering large point clouds. We know of no other method that can render incomplete hierarchies with full detail in parallel with its construction and data sorting. We also defined the novel idea of Hierarchy Oblique Cut, a strong concept that can be used to apply sweeps on hierarchies.

Additionally, OMiCroN opens the path for new workflows based on streaming of spatially sorted data. Supposing that large scans could be streamed directly in Morton order, the data could be rendered without any delays at all. Another advantage is that a dataset sorted in a Morton code level can be rendered by OMiCroN using a hierarchy with any level less or equal to the sorting level.

References

[1] S. Rusinkiewicz and M. Levoy, "Qsplat: A multiresolution point rendering system for large meshes," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*,

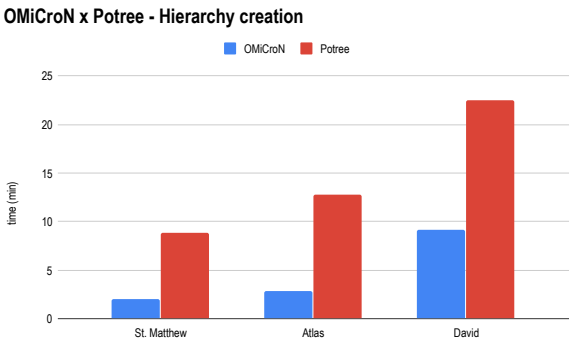


Fig. 9. OMiCroN and Potree [35] comparison. OMiCroN is more than 2 times faster for David and more than 4 times faster for St. Matthew and Atlas.

ser. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 343–352. [Online]. Available: http://dx.doi.org/10.1145/344779.344940

[2] M. Wimmer and C. Scheiblauer, "Instant points: Fast rendering of unprocessed point clouds," in *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, ser. SPBG'06. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006, pp. 129–137. [Online]. Available: http://dx.doi.org/10.2312/SPBG/SPBG06/129-136

[3] J. Klein and G. Zachmann, "Point cloud collision detection," in *Computer Graphics Forum*, vol. 23, no. 3. Wiley Online Library, 2004, pp. 567–576.

[4] Morton, "A computer oriented geodetic data base and a new technique in file sequencing," IBM Ltd., Tech. Rep. Ottawa, Ontario, Canada, 1966.

[5] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-dops," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–36, Jan. 1998. [Online]. Available: http://dx.doi.org/10.1109/2945.675649

[6] S. A. Ehmann and M. C. Lin, "Accurate and fast proximity queries between polyhedra using convex surface decomposition," *Computer Graphics Forum*, vol. 20, no. 3, pp. 500–511, 2001.

[7] C. Lauterbach, Q. Mo, and D. Manocha, "gproximity: Hierarchical gpu-based operations for collision and distance queries." *Comput. Graph. Forum*, vol. 29, no. 2, pp. 419–428, 2010.

[8] O. Argudo, I. Besora, P. Brunet, C. Creus, P. Hermosilla, I. Navazo, and À. Vinacua, "Interactive inspection of complex multi-object industrial assemblies," *Computer-Aided Design*, vol. 79, pp. 48 – 59, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010448516300628

[9] M. Levoy and T. Whitted, *The use of points as a display primitive*. Chapel Hill, NC, USA: University of North Carolina, Department of Computer Science, 1985.

[10] J. P. Grossman and W. J. Dally, "Point sample rendering," in *Rendering Techniques '98*, G. Drettakis and N. Max, Eds. Vienna: Springer Vienna, 1998, pp. 181–192.

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

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

[13] M. Alexa, M. Gross, M. Pauly, H. Pfister, M. Stamminger, and M. Zwicker, "Point-based computer graphics," in *ACM SIGGRAPH 2004 Course Notes*. ACM, 2004, p. 7.

[14] M. Gross, "Getting to the point...?" *IEEE computer graphics and applications*, vol. 26, no. 5, pp. 96–99, 2006.

[15] M. Gross and H. Pfister, *Point-Based Graphics*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[16] F. Ramos, J. Huerta, and F. Benitez, "Characterization of multiresolution models for real-time rendering in gpu-limited environments," in *International Conference on Articulated Motion and Deformable Objects*. Springer, 2016, pp. 157–167.

[17] C. Dachsbacher, C. Vogelgsang, and M. Stamminger, "Sequential point trees," in *ACM Transactions on Graphics (TOG)*, vol. 22, no. 3. ACM, 2003, pp. 657–662.

[18] R. Pajarola, M. Sainz, and R. Lario, "Xsplat: External memory multiresolution point visualization," in *Proceedings IASTED Invernational Conference on Visualization, Imaging and Image Processing*, 2005, pp. 628–633.

[19] E. Gobbetti and F. Marton, "Layered point clouds: A simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models," *Comput. Graph.*, vol. 28, no. 6, pp. 815–826, Dec. 2004. [Online]. Available: http://dx.doi.org/10.1016/j.cag.2004.08.010

[20] 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 *SPBG*, 2007, pp. 37–45.

[21] 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 *VAST*, 2009, pp. 25–32.

[22] E. Hubo and P. Bekaert, "A data distribution strategy for parallel point-based rendering," 2005.

[23] W. T. Corrêa, S. Fleishman, and C. T. Silva, "Towards point-based acquisition and rendering of large real-world environments," in *Computer Graphics and Image Processing, 2002. Proceedings. XV Brazilian Symposium on*. IEEE, 2002, pp. 59–66.

[24] W. T. Corrêa, J. T. Klosowski, and C. T. Silva, "Out-of-core sort-first parallel rendering for cluster-based tiled displays," *Parallel Computing*, vol. 29, no. 3, pp. 325–338, 2003.

[25] P. Goswami, M. Makhinya, J. Bösch, and R. Pajarola, "Scalable parallel out-of-core terrain rendering." in *EGPGV*, 2010, pp. 63–71.

[26] P. Goswami, F. Erol, R. Mukhi, R. Pajarola, and E. Gobbetti, "An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees," *The Visual Computer*, vol. 29, no. 1, pp. 69–83, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00371-012-0675-2

[27] N. Lukac *et al.*, "Hybrid visualization of sparse point-based data using gpgpu," in *Computing and Networking (CANDAR), 2014 Second International Symposium on*. IEEE, 2014, pp. 178–184.

[28] Z. Gao, L. Nocera, M. Wang, and U. Neumann, "Visualizing aerial lidar cities with hierarchical hybrid point-polygon structures," in *Proceedings of Graphics Interface 2014*, ser. GI '14. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 2014, pp. 137–144. [Online]. Available: http://dl-acm-org.ez29.capes.proxy.ufrj.br/citation.cfm?id=2619648.2619672

[29] R. Richter, S. Discher, and J. Döllner, "Out-of-core visualization of classified 3d point clouds," in *3D Geoinformation Science*. Springer, 2015, pp. 227–242.

[30] A. Febretti, K. Richmond, P. Doran, and A. Johnson, "Parallel processing and immersive visualization of sonar point clouds," in *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*. IEEE, 2014, pp. 111–112.

[31] M. Potenziani, M. Callieri, M. Dellepiane, M. Corsini, F. Ponchio, and R. Scopigno, "3dhop: 3d heritage online presenter," *Computers & Graphics*, vol. 52, pp. 129–141, 2015.

[32] R. Tredinnick, M. Broecker, and K. Ponto, "Experiencing interior environments: New approaches for the immersive display of large-scale point cloud data," in *Virtual Reality (VR), 2015 IEEE*. IEEE, 2015, pp. 297–298.

[33] H. Okamoto and H. Masuda, "A point-based virtual reality system for supporting product development," in *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2016, pp. V01BT02A052–V01BT02A052.

[34] K. Ponto, R. Tredinnick, and G. Casper, "Simulating the experience of home environments," in *Virtual Rehabilitation (ICVR), 2017 International Conference on*. IEEE, 2017, pp. 1–9.

[35] M. Schütz, "Potree: Rendering large point clouds in web browsers," *Technische Universität Wien, Wiedeń*, 2016.