Fast low-memory streaming MLS reconstruction of point-sampled surfaces

Gianmauro Cuccuru* Sardegna DISTRICT
Enrico Gobbetti† CRS4
Fabio Marton‡ CRS4
Renato Pajarola§ University of Zurich
Ruggero Pintus¶ CRS4

Abstract

We present a simple and efficient method for reconstructing triangulated surfaces from massive oriented point sample datasets. The method combines streaming and parallelization, moving least-squares (MLS) projection, adaptive space subdivision, and regularized isosurface extraction. Besides presenting the overall design and evaluation of the system, our contributions include methods for keeping in-core data structures complexity purely locally output-sensitive and for exploiting both the explicit and implicit data produced by a MLS projector to produce tightly fitting regularized triangulations using a primal isosurface extractor. Our results show that the system is fast, scalable, and accurate. We are able to process models with several hundred million points in about an hour and outperform current fast streaming reconstructors in terms of geometric accuracy.

Index Terms: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—

1 Introduction

Reconstructing triangulated surfaces from oriented point samples is required in many important applications, ranging from industry, over entertainment, to cultural heritage. The need of managing massive datasets and of requiring certain guarantees on reconstructed surfaces, makes this problem an active subject of research, that has produced a variety of solutions in the last years (see Sec. 2). Our aim is to propose an efficient framework for the streaming implementation of an important class of them.

The method combines a number of ingredients: streaming and parallelization, moving least-squares (MLS) projection, adaptive space subdivision, and regularized isosurface extraction. Although not all the techniques presented here are novel in themselves, their elaboration and combination in a single system is non-trivial and represents a substantial enhancement to the state-of-the-art. In particular, besides presenting the overall design and evaluation of the system, our contributions include methods for keeping in-core data structures complexity purely locally output-sensitive and for exploiting both the explicit and implicit data produced by a MLS projector to produce tightly fitting regularized triangulations using a primal isosurface extractor. Our results show that the resulting system is fast, scalable, accurate and produces good quality meshes. In particular, we are able to process models with several hundred million points in about an hour (see Fig. 1) and outperform current state-of-the-art fast streaming reconstructors in terms of geometric accuracy.

*e-mail:gmauro@crs4.it
†e-mail:gobbetti@crs4.it
‡e-mail:marton@crs4.it
§e-mail:pajarola@ifi.unizh.ch
¶e-mail:ruggero@crs4.it

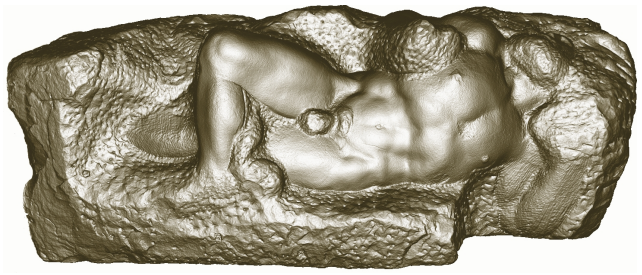


Figure 1: **Surface reconstruction of Michelangelo's "Awakening Slave" from laser scans containing over 380M samples.** This high quality reconstruction was completed in less than 85 minutes on a quad core machine using 588MB/thread. The regularized manifold triangulation contains about 220M triangles, corresponding to over 345M triangles before clustering. The model has been rendered with a highly reflective Minnaert BRDF to emphasize small-scale details.

2 Related work

Surface reconstruction has been extensively studied and literally hundreds of different contributions have appeared in the last decade. We provide here only a brief survey of the subject, emphasizing the most closely related approaches.

Explicit reconstruction. Explicit methods, e.g., [5, 13, 14], are based on the adaptive creation and maintenance of combinatorial structures and typically create triangulations that interpolate a subset of input points. These methods have strong theoretical guarantees, but their interpolation nature makes them sensitive to noise, often requiring extensive pre- or post-processing to produce smooth surfaces from noisy data [24, 28]. Moreover, the need to track and maintain neighboring information to produce triangulations makes them hard to apply to large datasets because of the associated time- and memory-overheads. Even though some of them can be implemented in a streaming fashion to overcome memory issues, state-of-the art techniques are reported to work only of models with few millions polygons at speeds of 2M points/hour [4].

Local implicit reconstruction. Implicit schemes reconstruct an approximating surface using a level-set of a function, typically sampled on an adaptive Cartesian grid. This approach offers more space to optimization and more robustness with respect to noise. Local implicit schemes are scattered data approximation approaches working on a subset of points at a time. Since access patterns are localized, they are typically amenable to efficient out-of-core implementation. A large variety of schemes have been proposed. Many of them fit the point cloud data with a linear combination of Radial Basis Functions (RBF), using compactly supported supported bases [30] and hierarchical partitioning combined with partition of unity blending methods [29] to deal with large data. Our scheme falls in the family of MLS methods [3, 6, 25]. One of the main strengths of these methods is the intrinsic capability to handle noisy input. A large number of reconstruction techniques based on this framework

have been proposed (e.g., [15, 17, 23, 33]), and our contribution consists in a streaming framework for efficiently implementing them in a low-memory setting. Even though many local schemes are theoretically amenable to a streaming implementation, the most common approaches either maintain an adaptive structure fully in-core [9], or exploit an adaptive random access structure maintained out-of-core [11, 16]. Streaming is advantageous over pure out-of-core approaches because input data is accessed sequentially from disk.

Global implicit reconstruction. Global implicit schemes work on either the surface itself or on the volume's indicator function. In both cases, early global fitting methods (e.g., [10, 20]) required contributions from all input samples at each reconstructed surface point, leading to large memory and processing time overheads. Methods working on the indicator function have proved to be very robust to noise and have recently been improved to work on large datasets. In particular, the Fourier method of [20] was later modified to reconstruct the indicator function by solving a Poisson equation with a streaming multigrid solver [8, 21]. The resulting method proved to be able to process massive data sets of hundreds of millions of points. The method requires, however, a large amount of temporary storage for storing the entire octree before the equation is solved. Moreover, its speed is limited to about 5M samples/hour [8]. These problems have been recently overcome by using wavelets with compact support for reconstructing the indicator function [27]. The resulting system requires a moderate amount of in-core memory and has an impressive speed of several hundred million samples/hour. Our method has similar performance characteristics, but provides different features. Wavelet reconstruction is a fast, robust scheme tuned for producing watertight surfaces, while our scheme is tuned for approximation near sample points of (open) surfaces with attributes.

Isosurface extraction and regularization. Our method builds on the primal approach of [19, 22] for extracting crack-free isosurfaces by a combination of cubical marching squares and edge-tree visits for consistently locating isovertices. The method is improved by combining it with a specialized topology preserving vertex clustering technique. In contrast to early isosurface regularization methods based on clustering [35], we support unconstrained octrees and exploit both the projected points and the isovalues to produce tightly fitting regularized triangulations.

3 Method overview

Our system is applied to point clouds that are already registered in 3D. We assume that each point sample $\mathbf{p}_i$ has an associated normal $\mathbf{n}_i$ and influence radius r_i, representing local point spacing, as well as other optional attributes (e.g., a color $\mathbf{c}_i$). During a streaming pass over the input samples, assumed ordered along the samples' major extent axis, a narrow sliding slab of octree cells is refined to a resolution that adaptively matches the local point density. A projection operator is then iteratively applied to the leaf vertices of the current octree region for projecting them on the surface implied by the local point set, and to generate a discretized signed scalar distance field. Both the projected points and the distance field are then exploited by a regularized isosurface extractor working on unconstrained octrees. By splitting the streaming reconstruction process among multiple threads, the method can work in parallel, fully exploiting the power of current multi-core architecture. The following sections provide details on the method's building blocks and on their combination.

3.1 Data preprocessing

Similarly to prior point streaming algorithms [31], we require that the input points are sorted along a direction in Euclidean space (assumed to be the z-axis without loss of generality) and that the bounding box of the entire point cloud is known before streaming. Before sorting, we rotate the point set to align the dominant axis of its covariance matrix with the z-axis. This rotation reduces the maximum complexity encountered during streaming, and therefore peak memory size of the in-core streaming structures.

If normals and radii are missing, it is possible to estimate them from the point cloud using a variety of methods, such as those based on PCA. In most practical cases, however, normals and radii can be derived directly from source data at little cost, e.g., from the range maps provided by laser scanners. In the examples presented in this paper, we assume that normals are already provided, and influence radii are computed with a fast density estimation technique. In this approach, also implemented in a streaming pass, we refine octree cells containing sample points until we reach some maximum depth d_{max} specified by the user. At each visited cell, we maintain the count of contained points. When a octree region has been completed, we locate for each inserted point $\mathbf{P}_i$ the finest level cell that contains at least a prescribed number of samples (16 in our examples). Then, the value of r_i is set to be equal to $2\sqrt{\frac{A}{N}}$, where A is the side area of that cell, and N the number of contained points. This method is much faster than those based on k-neighborhoods [32], since no neighbor searches are required.

3.2 Generating octree corner data by MLS

Our method produces a triangulated surface during a streaming pass over the input samples. We associate an in-core octree to the points' bounding box. During streaming, a sweep plane is advanced along the z-axis in discrete steps to sequentially reconstruct one narrow octree slice at a time.

Since we employ a primal approach for surface extraction, we need to transform the input point cloud into implicit and explicit data sampled at octree cell corners. Per-corner data consists of an approximation of the signed distance to the point set surface, as well as of the location, normal, and other attributes of the closest surface point. We obtain this information by a MLS approach. Each input sample is assumed to have a weight $w_i(x) = q_i \cdot \Phi\left(\frac{\|p_i - x\|}{r_i \cdot H}\right)$, where H is a global smoothing factor allowing to adjust the influence radius of every point, q_i is a sample quality factor, and Φ is a smooth, decreasing weight function for which we use the following compactly supported polynomial: $\Phi(x) = \max(0, (1-x^2))^4$. Since the function has local support, only points within a distance of $R_i = r_i \cdot H$ from x need to be evaluated. The quality factor q_i is used to incorporate in the system knowledge about error distribution in the dataset. In the examples considered in this paper, the factor is set directly proportional to the local sampling density (the higher, the better). More elaborate weighting schemes tuned for 3D scanning are presented elsewhere (e.g., [12]).

At reconstruction time, the octree is refined so that leaf cells have a size proportional to the smallest influence radius $R_{min} = r_{min} \cdot H$ among contributing points, i.e., among points that have a strictly positive contribution. In all examples presented here, refinement is performed up to the finest level l having a cell size greater than $\frac{2}{\sqrt{3}} \cdot R_{min}$, i.e., to the coarsest level with cells containable in a ball of radius R_{min}.

For constructing the triangulation, the corners of the octree cells lying in the current active region are projected to the point-set by applying the almost orthogonal MLS projection procedure starting from the corner's position [2]. The method is iterative. In the linear case used for simplicity in this paper, for a point $\mathbf{q}^{(k)}$ obtained in the k-th iteration, a local average position and normal direction are computed:

$$\mathbf{a}(\mathbf{q}^{(k)}) = \frac{\sum \Phi_i(\|\mathbf{p}_i - \mathbf{q}^{(k)}\|)\mathbf{p}_i}{\sum \Phi_i(\|\mathbf{p}_i - \mathbf{q}^{(k)}\|)} \quad (1)$$

$$\mathbf{n}(\mathbf{q}^{(k)}) = \frac{\sum \Phi_i(\|\mathbf{p}_i - \mathbf{q}^{(k)}\|)\mathbf{n}_i}{\|\sum \Phi_i(\|\mathbf{p}_i - \mathbf{q}^{(k)}\|)\mathbf{n}_i\|} \quad (2)$$

Other attributes, such as colors, are similarly interpolated. Then, the original corner position is projected on the plane defined by $\mathbf{a}(\mathbf{q}^{(k)})$ and $\mathbf{n}(\mathbf{q}^{(k)})$ to determine a new approximation $\mathbf{q}^{(k+1)}$. In a pure weighted least squares (WLS) approach, directly applicable in many cases, the first value q^1 is the final approximation. In a MLS approach, the approximation is improved by iterating the projection (we will see in Sec. 3.4 how this can be done in a limited memory setting). In both the MLS and WLS cases, the end result is an approximation of the closest surface point $\mathbf{q}^\star$, with associated normal and attributes. The signed distance used to generate the scalar field needed by the polygonization algorithm is then computed between the original corner and its projection on the final MLS plane. A similar procedure can be used for more complex local approximations. For instance, a more robust fit can be obtained by locally approximating the point cloud by a fitted algebraic sphere that moves continuously in space [18]. This is orthogonal to the work presented in this paper.

3.3 Fast surface smoothing by cell-to-vertex diffusion

The MLS procedure outlined above generates all the data required for surface reconstruction per cell corner. The H parameter allows users to control the amount of smoothing by modifying the support radii and, therefore, the number of points that are blended together at a given vertex. As noted in [34], another approach for smoothing-out fine details without increasing blending cost is to perform a post-processing smoothing step on the implicit function prior to reconstruction. In our approach, this sort of smoothing step is performed by applying the MLS projection procedure to the cell centers instead of cell corners and then, prior to surface extraction, computing the required information at cell corners by blending together the values associated with all the non-empty incident cells. This blending is obtained by weighted averaging of all values, with weights inversely proportional to cell sizes. Even though averaging slightly decreases accuracy by removing high frequency details, good looking surfaces can be obtained faster for two reasons. First of all, the same degree of smoothing can be obtained with decreased support radii, and therefore decreased MLS projection costs. Second, the method can be fully implemented using cell location rather than corner location operations, leading to increased efficiency for most octree implementations.

3.4 Efficient low-memory streaming implementation

Streaming reconstruction assumes that the octree is adaptively refined near the surface of the model at a level that matches the local point density before processing it to extract a local triangulation. At any given moment, to minimize memory needs, the refined and updated octree must exist only in the smallest possible neighborhood around the current processed slice, and the size of in-memory structures should depend only on the local output size complexity. At the same time, the octree refinement process must be kept in sync with the process that sequentially consumes points extracted from the input stream.

In order to efficiently implement this approach, we assume that there are known bounds on the maximum radius of influence of an input sample, R_{max}, and on the coarsest octree leaf cell level, L_{coarse}, defining a portion of the output surface. By using R_{max}, the streaming process can conservatively determine while streaming if samples further in the stream cannot possibly influence a given octree region because it is out of their influence radius. Furthermore, by aligning processing regions with the grid of level L_{coarse}, and forcing octree subdivision to at least that level at each sample insertion, it is also possible to bound the effects of backward propagation of cell subdivisions, which is required for a streaming implementation. By combining these two facts, it is therefore possible to conservatively decide when a streaming region is completely determined by the samples already extracted from the stream. It should be noted that the bounds on radius and cell size are related but do not have the same effects. The cell size bound limits the maximum size of triangles in the output triangulation. In order to limit memory occupation, we typically force the coarse level to be 0-3 levels less than the finest one dictated by the maximum reconstruction resolution. The maximum radius bound has no effect on the reconstruction, but limits the amount of look-ahead necessary before processing a region, therefore improving performance.

A major problem that needs to be tackled in practice for very large datasets is related to the sheer number of samples that may contribute to a particular region, especially when not reconstructing at the finest resolutions. When the octree is restricted to a shallow depth due to a large blending factor H, it is not uncommon to have neighborhoods of thousands of samples per cell. Per cell point bins, or auxiliary structures for supporting point locations would consume large amount of memory, limiting the scalability of the methods. For instance, [8] noted that their streaming reconstruction system requires radically more memory at coarse resolution than at fine resolution.

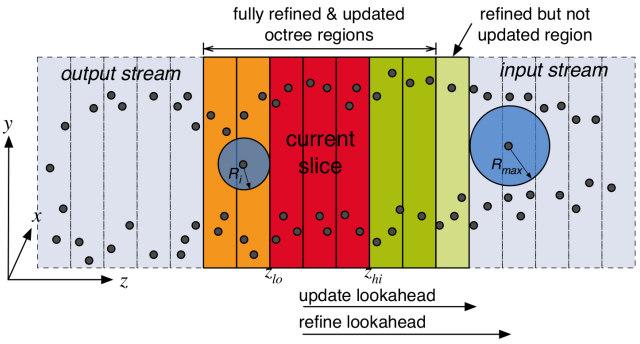


Figure 2: **Streaming regions.** The streaming axis is discretized at the resolution of the coarsest cell size.

We solve this problem by totally avoiding the storage of samples using a local multi-pass approach (see Fig. 2). At each sliding step, we consider that we want to reconstruct the surface in the interval $z_{lo}..z_{hi}$, where z_{lo} and z_{hi} are multiples of the coarsest cell size dz_{coarse} determined by L_{coarse}. Having $z_{hi} > z_{lo} + 1$ enables the system to emit data only

once every few slices, increasing performance at the cost of an increase in in-core memory needs.

Because of sequential streaming, all data before z_{lo} has already been constructed, which means that the triangulation has already been emitted on output, and that this data is already available in the octree if required. For reconstruction, the octree must be fully updated not only in the cells contained within the current z range, but also in the immediate neighborhood of it, depending on the type of operations that are applied. Cell-to-vertex diffusion, for instance, requires data from all cells in a one-ring neighborhood of a cell's corner vertex. As we will see in Sec. 3.5, vertex clustering also requires one-ring neighborhood updates. Since we know that the coarsest cell size containing data will be of a size bounded by dz_{coarse}, we can conservatively assume that the reconstruction of an octree slice would access at most all cells within a range of two layers of width from the slice's boundary dz_{coarse}.

The current active window is determined by identifying two active stream ranges: one for refining the octree structure, and one for iteratively updating the fully refined region to implement MLS projection. We first ensure that the region up to $z_{hi} + 2 \cdot dz_{coarse}$ is fully refined, by refining all cells to the level implied by input point samples up to $z_{hi} + 3 \cdot dz_{coarse}$. The extra look-ahead of one more layer of refined cells is required to avoid back-propagation of future octree subdivisions. Once this refinement is performed at the first stream position with $p_z - R_{max} > z_{hi} + 3 \cdot dz_{coarse}$, we restart streaming from the first sample whose region of influence overlaps with the current reconstruction range. We then execute a number of update passes on the appropriate region by repeatedly streaming over all samples in the identified active streaming range. For each pass, the contributions of all samples are accumulated at the cells into running sums to compute the current MLS solution. At the end of the pass, the projection step is performed and all accumulators are cleared in preparation for the next MLS iteration. When the user-determined number of MLS projection steps has been reached, the solution is finalized by eventually executing the cell-to-vertex diffusion step and extracting the triangulation. Streaming then continues by advancing the sliding window and deleting all unneeded data in the past before continuing with the refine-update-extract cycle. The method requires an amount of in-core memory which is proportional to the output complexity, since the only in-core structure is the octree with associated constant-size cell and vertex data. This approach has the drawback of applying the same number of MLS iterations to all points. This drawback is more than offset by the reduced memory needs, which make the system fully scalable. In the rare cases in which different areas of a dataset require radically different iteration counts to reach convergence, it is possible to reduce computational performance by detecting, at the end of each cycle, which corners have already reached a good level of approximation and inserting in future iterations only samples that contribute to them. This requires an additional storage of one bit per active sample for maintaining the bitmask.

3.5 Surface extraction

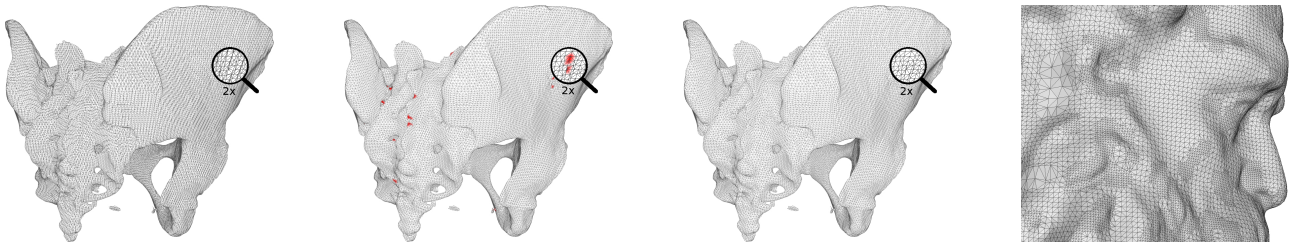
Our method relies on the ability to extract triangulated surfaces from adaptively refined octrees. Hence we build on the primal approach of [19, 22] for extracting crack-free isosurfaces by combining cubical marching squares with edge-tree visits for consistently locating isovertices. The method is improved by a specialized vertex clustering technique that is optionally able to preserve topology. This approach allows us to create regular triangulations that nicely adapt to sample density (see Fig. 3(d)).

Each time an octree slice has been updated completely, all leaf cells within this region are visited, and isovertices and triangles are located and extracted as in [22]. The isovertices are selected on all edges that exhibit a sign change in the implicit value and have both projected points close enough to the weighted average of the input points (see [1] for details).

Without vertex clustering, the triangulation is simply emitted to the output triangle stream, and the process can continue. Such a triangulation, however, is far from optimal, since on average 3 triangles per non-empty octree cell are extracted. In addition, where the cubic lattice just intersects the surface, these triangles can be arbitrarily small or thin (see Fig. 3(a)). Instead of relying on a post-processing beautification step, we regularize the mesh by adopting a specialized constrained vertex clustering approach. During the recursive surface extraction, we do not immediately emit vertices and triangles, but temporarily maintain them in a vertex table and triangle list. Each time an isovertex is generated on a octree cell edge, it is also assigned to the vertex set corresponding to the closest octree corner. Before emitting the triangulation, the vertex sets associated to all active octree corners are visited, and the algorithm determines which of the vertices can be clustered together. Each vertex in a cluster is then replaced by a single representative, with a unique vertex reference and aggregated attributes. Once all corners are processed, the triangles stored during isosurface extraction are emitted by replacing the original vertex references with those resulting from clustering. Any triangles which contain two or more identical vertex references are discarded before output.

Central to this method is the algorithm used to synthesize new vertex values for a given cluster. In the simplest case of unconstrained clustering, all isovertices associated to a given corner need to be replaced by a single representative. In this case, we can take advantage of the fact that during the MLS projection step each corner not only computes an implicit value but also the explicit position, normal, and attributes of its projection on the point set surface. By using the projected MLS values for the representative vertex we can thus produce a surface that has nice triangulations and, at the same time, tightly follows the original point cloud, similar to explicit methods that directly interpolate MLS-projected data [15].

Unconstrained clustering, however, may in some situations create topological and geometrical artifacts, such as non-manifold vertices/faces and holes (see Fig. 3(b)). We overcome this problem by using a topology preserving clustering approach. In this case, we also associate to each corner the set of incident triangles. At corner processing time, we first separate the triangulation associated to it into its separate connected components. For each component, we then check whether collapsing the vertices associated to the current corner would not modify the topology and generate a new representative vertex only if this check is positive. The topological check consists in verifying three conditions: (a) that the triangulation is homeomorphic to a disk before clustering; (b) that it remains disk-like after clustering; (c) that it contains vertices associated to at least three different clusters. These conditions guarantee that the end-result of clustering will have at least three vertices, and thus cannot degenerate to points or segments even if all incident clusters will perform clustering. Since only few triangles are part of a cluster, verification that a cluster is homeomorphic to a disk is performed explicitly, by checking that no edge is shared



(a) (b) (c) (d)

Figure 3: **Triangulation regularization.** (a): Original marching cubes reconstruction of a hip bone. (b): Unconstrained regularization, producing holes and non-manifold vertices (in red) (c): Topology preserving regularization, producing a manifold surface. (d): Coarse reconstruction of the Neptune model: triangles are regularly distributed and at a density that matches the density of the input samples.

by more than one triangle, and that the region is bounded by a single closed loop of boundary edges.

In most cases (over 95% of all tested practical situations), the topological check results in a single collapsable cluster per corner. In that case, we revert to the unconstrained case and use as representative vertex the MLS projection associated to the corner. In the few remaining cases, we perform a separate quadric collapse per component [26].

The advantages of the resulting method are that the original manifold topology is preserved, and that explicit data generated at octree corners in addition to signed distances is exploited to produce a tightly fitting surface (see Fig. 3(c)). The additional time required for clustering is offset by the reduction in points and triangles generated. On the other hand, it should be noted that in a streaming implementation clustering increases the amount of memory that has to be kept in core at any given moment. First of all, in order to cluster vertices associated to a given corner, all cells around it should already be complete. This imposes a look-ahead of one completed cell layer into the input stream. Moreover, triangles and vertices generated for the slice need to be kept in core for further processing rather than being emitted on the fly. As for the rest of the system, the additional memory costs are strictly locally output sensitive.

3.6 Parallelization

The streaming reconstruction approach outlined above performs a series of sequential operations on the stream that consist of a sequence of octree refinements, octree MLS updates, and surface extractions that are applied only on a small sliding window which advances at discrete steps of size determined by L_{coarse}. Hence the method can easily be parallelized by splitting the data stream among threads into chunks aligned with the grid of level L_{coarse} and perform reconstruction separately for each chunk. Since work is largely proportional to input data complexity, it is sufficient for load balancing to create chunks of approximately the same number of input points. The overall consistency of the triangulation is ensured by assigning to each thread chunks of input points that overlap by the number of processing regions sufficient to ensure boundary conditions. In the overlap region at the beginning of the assigned chunk, each thread executes only the refine and update operations and skips the actual triangulation pass which is completed by the corresponding other thread. In order to reduce synchronization overhead, we let each thread produce separate streaming triangulation files using purely local per-thread vertex indices. This sort of very simple parallelization approach does not require shared memory among threads and thus easily supports execution on distributed memory machines such as PC clusters. We will in the future look at more fine-grained parallelization at the level of each reconstruction thread. A final consolidation pass, executed after all threads have completed their triangulation jobs, joins all files together by re-indexing all vertices in a single streaming pass. This operation is performed efficiently by separately storing for each triangulated chunk the indices of the vertices lying on a chunk boundary, which are the only ones shared among chunks. With this information, triangle mesh merging can be performed using a small table per chunk to re-index shared vertices. All other vertices within a chunk are re-indexed by just offsetting them by a fixed value.

4 Implementation and results

An experimental software library supporting the presented technique has been implemented on Linux using C++ and MPI. Here, we present results obtained from streaming reconstruction from large point clouds.

4.1 Speed and scalability on massive models.

We have applied our algorithm with various parameter settings to the aligned range scans of Michelangelo's "Awakening Slave" unfinished statue consisting of over 380M samples, as well as to the "Pisa Cathedral" dataset consisting of over 154M samples. The points and normals used for reconstruction were taken directly from the original scans and the radii were computed using our fast octree based method, with undersampled areas discarded to filter out clear outliers and misalignment. The Pisa Cathedral dataset also has a color channel associated to part of the scans. In all other scans the color is set by default to average gray. The benchmark results discussed here are relative to a four-thread configuration running on a single quad core PC with a Intel Core 2 Quad CPU 2.4GHz, 4GB RAM, and a single SATA2 disk storing both input point clouds and reconstructed meshes.

Model	Diffus.	Clust.	Time	Mem	Edge Len	Output Triangles
Awakening	No	NO	1h44m	1.6GB	0.28	388.9M
Awakening	No	Unrest.	1h49m	1.8GB	0.35	227.7M
Awakening	No	Constr.	1h58m	2.0GB	0.35	232.8M
Awakening	Yes	NO	1h11m	2.1GB	0.29	345.5M
Awakening	Yes	Unrest.	1h16m	2.3GB	0.35	219.6M
Awakening	Yes	Constr.	1h24m	2.4GB	0.35	219.9M
Cathedral	No	NO	55m	2.3GB	19	142.1M
Cathedral	No	Unrest.	57m	2.6GB	26	68.8M
Cathedral	No	Constr.	1h2m	2.8GB	26	70.9M
Cathedral	Yes	NO	38m	3.0GB	19	141.2M
Cathedral	Yes	Unrest.	40m	3.3GB	26	68.8M
Cathedral	Yes	Constr.	45m	3.4GB	26	69.6M

Table 1: **Statistics for the reconstruction of "Awakening Slave" and "Pisa Cathedral" at H=1 and 1 MLS projection step.** Data gathered on a single quad-core PC running four parallel reconstruction threads.

Reconstruction memory footprint. Table 1 summarizes the results obtained with a blending factor of $H = 1$, which generated the models in Fig. 1 and 4. On the largest model, all the runs were completed with a peak in-core memory between 409MB and 614MB per thread depending on the configuration. The Cathedral, even if smaller, has a peak in-core memory footprint that varies from 588MB to 870MB, due to the larger maximum cross-section. As expected, cell-to-vertex diffusion and clustering both increase in-core memory cost because of the need for larger cell neighborhoods kept in memory. Nevertheless, our results favorably compare in all cases with other state-of-the-art streaming reconstruction results on the same datasets and with similar settings. For instance, streaming wavelet reconstruction of the Awakening requires 574MB when using Haar wavelets and more than 1.6GB when using the smoother D4 wavelets [27]. The Poisson reconstruction code requires for the same dataset 2GB of in-core memory and over 100GB of out-of-core octree memory [8]. The lower memory usage of our approach is particularly important for reconstructions with large blending factors, as illustrated in Table 2, which shows data gathered for reconstructions with varying blending factors H. Corresponding reconstruction details of the Awakening model are presented in Fig. 5. A detail of the facade of the Pisa Cathedral, with color details, is also presented in Fig. 6. Our memory footprint is fully output sensitive, and at $H = 4$ we use only 153MB/thread for the largest model. By contrast, the Poisson reconstruction code tends to use even more memory at coarse than at fine resolutions because of the cost of point sample binning (e.g., 521MB for a level 8 reconstruction of a 200M samples dataset versus 212MB for a level 11 reconstruction) [8]. It must be also noted that our approach also handles colors, not managed by the Poisson and Wavelet reconstruction codes.

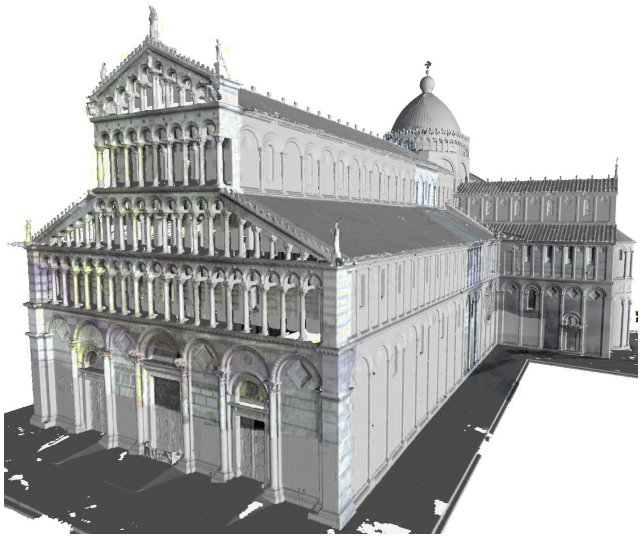


Figure 4: **Reconstruction of "Pisa Cathedral" from laser scans with over 154M samples.** This reconstruction was completed in 45 minutes on a quad core machine using 870MB/thread. The regularized manifold triangulation contains about 70M triangles, corresponding to over 140M triangles before clustering. The color channel, present in only a few scans, is preserved in the reconstruction.

Reconstruction speed. The small memory footprint of our approach makes it possible to run multiple reconstruction processes in parallel on a single PC. As illustrated in Table 1, using four threads, reconstruction times at $H = 1$ and one MLS iteration ranges from 1h11m to 1h58m for the Awakening and from 45m to 1h02 for the Cathedral, depending on reconstruction parameters. Preprocessing times are not included in these times since they can be amortized over many reconstruction runs. Moreover, they are relatively small. Sorting the datasets, which needs to be done by current streaming methods, takes 49m for the Awakening and 23m for the Cathedral using a simple out-of-core quicksort implementation working on a memory-mapped file. Computing influence radii using our density based method streaming method takes 24m for the Awakening and 11m for the Cathedral.

The lowest reconstruction times are for reconstructions made with cell-to-vertex diffusion enabled, which is explained by the fact that octree cell location operations are implemented more efficiently than vertex location operations. The slight reduction in the triangle count of the cell-to-vertex diffusion version is due to the increased amount of smoothing, which reduces the overall output surface size. Enabling or disabling clustering has little effect on speed, but a large effect on output mesh quality and complexity, since the clustered version generates on average 60% of the triangles of the non-clustered version. Increasing the number of MLS iterations increases accuracy as well as reconstruction times, but does not require additional memory. For instance, reconstruction with topology preserving clustering and cell-to-vertex diffusion of the Awakening dataset requires 1h24m

Model	H	Time	Speed Points/s	Mem	Level	Edge Len	Gen. Tri	Out. Tri
Awakening	1	2h49m	35.9K	2.3GB	14	0.35	390.2M	228.3M
Awakening	2	2h16m	44.8K	1.1GB	13	0.71	86.2M	55.8M
Awakening	4	2h1m	50.1K	0.6GB	12	1.41	21.4M	13.9M
Cathedral	1	1h40m	25.0K	3.3GB	13	26	141.0M	68.7M
Cathedral	2	1h25m	29.6K	1.8GB	12	52	34.7M	16.9M
Cathedral	4	49m	50.8K	0.5GB	11	103	4.1M	8.3M

Table 2: **Statistics for the reconstruction of "Awakening Slave" and "Pisa Cathedral" at different resolutions using 4 MLS projection steps.** Data gathered on a single quad-core PC running four parallel reconstruction threads.

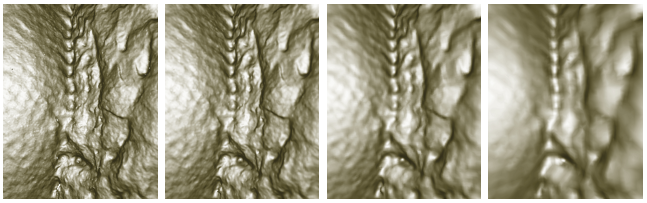


Figure 5: **Details of the reconstruction of "Awakening Slave".** Extreme zooms of the reconstruction with global blending parameter H=1,2,4,8. Note how the blending parameter controls surface smoothness and gap filling. At H=2 the surface is fully continuous and even small chisel marks are reconstructed with high accuracy.

Figure 6: **Detail of the reconstruction of "Pisa Cathedral".** Note how the color channel is preserved.

with one MLS iteration, 2h49m with 4 iterations, and 4h53m with 8 iterations. On the given datasets, there are, however, very little visual differences among these versions.

The reconstruction times obtained qualify our method among the fastest streaming reconstructors, even including pre-processing times. The code is still significantly slower on a single thread with respect to wavelet reconstruction with Haar wavelets, but is on par with wavelet reconstruction with D4 wavelets [27], which provide a visual quality more similar to that attained by our method. Other contemporary streaming frameworks are considerably slower (e.g., two days for a Poisson reconstruction of the same model [8]).

4.2 Reconstruction accuracy evaluation

The visual quality of our reconstructions for the "Awakening Slave" is illustrated in Fig. 1 and 5. Fig. 4 shows a reconstruction of the "Pisa Cathedral", which illustrates one of the advantages of the method, i.e., the ability to seamlessly handle attributes associated to point samples, in this case color. Since it is difficult to measure "accuracy" of reconstructions of a possibly noisy range scan, we use the approach of [27] of sampling points and normals from dense polygonal models and then computing the two-sided Hausdorff distance between reconstructed models and original surfaces. Distance evaluation has been performed using the MESH [7] software for comparing discrete surfaces. It should be noted that this kind of benchmark is biased towards methods that work well on "well behaved" surfaces. For this reason, we have also included results with models artificially distorted by uniform random noise in sample positions. The noise is in the interval $[-\frac{r}{2}, +\frac{r}{2}]$ for each coordinate, where r is the local sample spacing of the undistorted model. In order to make the test more similar to a real-world situation, noise is also included in normals, recomputed by PCA using 8 nearest neighbors on the noisy point sets, as well as in influence radii, recomputed on the noisy datasets using our density based technique.

	Arm.	Drag.	Nept.	Noisy Arm.	Noisy Drag.	Noisy Nept.
N1	0.177	0.111	0.123	0.312	0.392	0.236
N4	0.175	0.106	0.119	0.310	0.393	0.240
C,N1	0.182	0.118	0.135	0.315	0.391	0.240
C,N4	0.181	0.110	0.135	0.311	0.392	0.244
S,N1	0.208	0.155	0.189	0.312	0.368	0.218
S,N4	0.203	0.151	0.204	0.311	0.367	0.225
S,C,N1	0.210	0.163	0.208	0.317	0.381	0.229
S,C,N4	0.208	0.154	0.189	0.314	0.376	0.229
SW.Haar	0.358	0.683	0.320	0.441	0.699	0.390
SW.D4	0.491	0.747	0.455	0.538	0.772	0.492
SW.Haar S	0.871	0.841	0.934	0.869	0.803	0.917
SW. D4 S	1.000	1.000	1.000	1.000	1.000	1.000

Table 3: **RMS distance comparison.** Comparison of RMS distance between original and reconstructed surfaces. Each row is normalized with respect to the worst geometric error. Rows represent runs with different parameters: C is clustering (enabled/disabled); N is the number of MLS iterations; S is cell-to-vertex diffusion. Last four rows (SW) are relative to streaming reconstruction using wavelets approach [27] with Haar or D4 wavelets and with/without post-process smoothing (S).

In order to compare the performance of our code with respect to the state-of-the-art, we also reconstructed the same models with the publicly available code of streaming reconstruction using wavelets [27]. The original paper contains comparisons with results obtained by other recent methods (Poisson reconstruction [8] and MPU implicits [29]) that demonstrate that wavelet reconstruction consistently produces better results. We thus limited comparison to only that method, using Haar and D4 wavelets, possibly in conjunction with indicator function smoothing. We tuned algorithm parameters to generate reconstructions with about the same number of triangles for the streaming wavelet code and our code with clustering disabled. Thus, the comparison for the clustered version uses less triangles in our versions than in the the other cases. Table 3 presents the RMS error of the various reconstructions with respect to the original input mesh. The meshes used in this benchmark are the Armadillo and Dragon from the Stanford repository and Neptune from the Aim@Shape repository. All values have been normalized to the worst value, which results to be equal to one in the table. In all the cases our code is able to produce a mesh whose geometric error is significantly smaller than in the streaming wavelet best run. The accuracy improvement is at least 30%. Not surprisingly, the reconstructions with the best accuracy is the one that we obtain without clustering, without diffusion and with the largest amount of MLS projection steps. It is however interesting to analyze the behavior of the system in the other cases. As expected, the number of MLS projection steps increase the quality of the mesh in terms of root mean squared distance. Since we start from vertices which are near the implicit surface, due to the adaptivity of the octree structure, the number of iterations can often be kept small, especially in well-behaved cases such as these ones. It is also important to note that clustering does not sensibly reduce the method accuracy even though the triangle count is drastically reduced (about 60% of the original model). This is because clustered vertices are in general projected onto the implicit surface near the octree corner. In such a way, we are able to have a new mesh with a strongly reduced triangle count while maintaining practically unchanged the distance from the original surface. Clustering also tends to increases visual quality of the shaded output mesh because of the well behaved triangles with good aspect ratio. The smoothness of the output mesh can be increased using the diffusion version of the code: with this approach we are able to produce a smoother version of the mesh. We can see from Table 3 that using the diffuse version slightly reduces the quality of the mesh in terms of distance from the original dataset, removing some details with high frequency information. In the worst case, the error is increased by 20%.

4.3 Limitations

As for all current surface reconstruction techniques, the method has features that are advantageous, but also has limitations. In particular, as the method is based on surfaces, and considers reconstructed values valid only in the neighborhood of the original point cloud, it can seamlessly handle open models. On the other hand, closed models are not guaranteed to lead to watertight reconstructions if the influence radii of the input points are set too small or the input noise level is to high, leading to holes or spurious surfaces. Noise and misalignment can be handled by MLS blending and projection, at the cost of local smoothing and increased projection costs (see Fig. 7). More general gap filling, when required, must be handled by other specialized methods. We should emphasize, however, that the paper does not focus on proposing new techniques for faithful reconstruction, but, rather, on an efficient streaming implementation for an important class of them.

5 Conclusions

We have presented an efficient framework for streaming reconstruction of high quality triangulated surfaces from very large point clouds. The main benefits of the method lie in its performance, quality, and flexibility. Our results show that the resulting system is fast, scalable, accurate and produces good quality meshes. Besides improving the proof-of-concept implementation, we plan to extend the presented

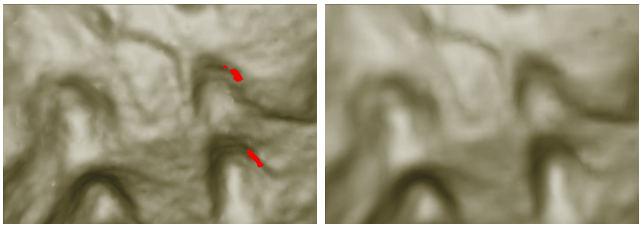


Figure 7: **Extreme details of the reconstruction of "Awakening Slave".** Left: at $H = 1$ the areas in which influence radii have been underestimated produce holes, highlighted in red. Right: at $H = 2$, the surface's appearance is smoother and holes have disappeared.

approach in a number of ways. In particular, it should be clear that the presented method is a framework on top of which more elaborate projection procedures can be implemented. The current system uses a linear least squares with the quasi-orthogonal projection scheme [2]. In the future, we plan to integrate algebraic sphere fitting [18] to improve robustness in areas with high curvature and/or noise and improve convergence speed. Finally, we are working on the integration of the presented reconstruction pipeline within a complete streaming framework for large point clouds.

Acknowledgements. The models are courtesy of Benedict Brown, Szymon Rusinkiewicz and the Digital Michelangelo Project (Awakening), the ISTI-CNR Visual Computing Group (Pisa Cathedral), the Stanford 3D Scanning Repository (Armadillo and Dragon) and INRIA (Neptune). This work was also partially supported by Sardegna DISTRICT (P.O.R. Sardegna 2000-2006 Misura 3.13) and by the Swiss National Science Foundation Grants 200021-111746 and IZAI20-121072.

References

[1] A. Adamson and M. Alexa. Approximating bounded, non-orientable surfaces from points. In *Shape Modeling International*, pages 243–252, 2004.

[2] M. Alexa and A. Adamson. On normals and projection operators for surfaces defined by point sets. In *Symposium on Point-Based Graphics*, 2004.

[3] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *IEEE Visualization*, pages 21–28, 2001.

[4] R. Allègre, R. Chaine, and S. Akkouche. A streaming algorithm for surface reconstruction. In *Symposium on Geometry Processing*, pages 79–88, 2007.

[5] N. Amenta, S. Choi, and R. Kolluri. The power crust. In *Symposium on Solid Modeling*, pages 249–260, 2001.

[6] N. Amenta and Y. J. Kil. Defining point-set surfaces. *ACM Transactions on Graphics*, 23(3):264–270, 2004.

[7] N. Aspert, D. Santa-cruz, and T. Ebrahimi. M.E.S.H.: Measuring errors between surfaces using the hausdorff distance. In *IEEE International Conference on Multimedia*, pages 705–708, 2002.

[8] M. Bolitho, M. Kazhdan, R. Burns, and H. Hoppe. Multilevel streaming for out-of-core surface reconstruction. In *Symposium on Geometry processing*, pages 69–78, 2007.

[9] T. Boubekeur, W. Heidrich, X. Granier, and C. Schlick. Volume-surface trees. *Computer Graphics Forum*, 25(3):399–406, 2006.

[10] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Proc. ACM SIGGRAPH*, pages 67–76, 2001.

[11] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):525–537, 2003.

[12] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. ACM SIGGRAPH*, pages 303–312, 1996.

[13] T. K. Dey, J. Giesen, and J. Hudson. Delaunay based shape reconstruction from large data. In *IEEE Symposium on parallel and large-data visualization and graphics*, pages 19–27, 2001.

[14] T. K. Dey and S. Goswami. Provable surface reconstruction from noisy samples. *Computational Geometry Theory and Applications*, 35(1):124–141, 2006.

[15] T. K. Dey and J. Sun. An adaptive MLS surface for reconstruction with guarantees. In *Symposium on Geometry processing*, page 43, 2005.

[16] V. Fiorin, P. Cignoni, and R. Scopigno. Out-of-core MLS reconstruction. In *CGIM*, pages 27–34, 2007.

[17] S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. *ACM Transactions on Graphics*, 24(3):544–552, 2005.

[18] G. Guennebaud and M. H. Gross. Algebraic point set surfaces. *ACM Transactions on Graphics*, 26(3):23, 2007.

[19] C.-C. Ho, F.-C. Wu, B.-Y. Chen, Y.-Y. Chuang, and M. Ouhyoung. Cubical marching squares: Adaptive feature preserving surface extraction from volume data. *Computer Graphics Forum*, 24(3):537–546, 2005.

[20] M. Kazhdan. Reconstruction of solid models from oriented point sets. In *Symposium on Geometry Processing*, pages 73–82, 2005.

[21] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Symposium on Geometry Processing*, pages 61–70, 2006.

[22] M. Kazhdan, A. Klein, K. Dalal, and H. Hoppe. Unconstrained isosurface extraction on arbitrary octrees. In *Symposium on Geometry Processing*, pages 125–133, 2007.

[23] R. Kolluri. Provably good moving least squares. *ACM Transactions on Algorithms*, 4(2):1–25, 2008.

[24] R. Kolluri, J. R. Shewchuk, and J. F. O'Brien. Spectral surface reconstruction from noisy point clouds. In *Symposium on Geometry Processing*, pages 11–21, 2004.

[25] D. Levin. *Geometric Modeling for Scientific Visualization*, chapter Mesh-independent surface interpolation. Springer, 2003.

[26] P. Lindstrom. Out-of-core simplification of large polygonal models. In *Proc. ACM SIGGRAPH*, pages 259–262, 2000.

[27] J. Manson, G. Petrova, and S. Schaefer. Streaming surface reconstruction using wavelets. In *Symposium on Geometry processing*, 2008.

[28] B. Mederos, N. Amenta, L. Velho, and L. H. de Figueiredo. Surface reconstruction for noisy point clouds. In *Symposium on Geometry Processing*, pages 53–62, 2005.

[29] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Transactions on Graphics*, 22(3):463–470, 2003.

[30] Y. Ohtake, A. Belyaev, and H.-P. Seidel. A multi-scale approach to 3D scattered data interpolation with compactly supported basis functions. In *Shape Modeling International*, page 153, Washington, DC, USA, 2003. IEEE Computer Society.

[31] R. Pajarola. Stream-processing points. In *IEEE Visualization*, page 31, 2005.

[32] M. Pauly, M. H. Gross, and L. Kobbelt. Efficient simplification of point-sampled surfaces. In *IEEE Visualization*, pages 163–170, 2002.

[33] P. Reuter, P. Joyot, J. Trunzler, T. Boubekeur, and C. Schlick. Surface reconstruction with enriched reproducing kernel particle approximation. In *Symposium on Point-Based Graphics*, pages 79–88, 2005.

[34] S. Schaefer and J. Warren. Dual marching cubes: Primal contouring of dual grids. *Computer Graphics Forum*, 24(2):195–203, 2005.

[35] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: improved iso-surface extraction. *Computers & Graphics*, 23(4):583–598, 1999.