Eurographics Symposium on Geometry Processing (2007)
Alexander Belyaev, Michael Garland (Editors)

Multilevel Streaming for Out-of-Core Surface Reconstruction

Matthew Bolitho[1], Michael Kazhdan[1], Randal Burns[1] and Hugues Hoppe[2]

[1]Johns Hopkins University, Baltimore MD, USA
[2]Microsoft Research, Redmond WA, USA

Abstract

Reconstruction of surfaces from huge collections of scanned points often requires out-of-core techniques, and most such techniques involve local computations that are not resilient to data errors. We show that a Poisson-based reconstruction scheme, which considers all points in a global analysis, can be performed efficiently in limited memory using a streaming framework. Specifically, we introduce a multilevel streaming representation, which enables efficient traversal of a sparse octree by concurrently advancing through multiple streams, one per octree level. Remarkably, for our reconstruction application, a sufficiently accurate solution to the global linear system is obtained using a single iteration of cascadic multigrid, which can be evaluated within a single multi-stream pass. We demonstrate scalable performance on several large datasets.

1. Introduction

We address the robust reconstruction of surfaces from large noisy oriented point sets. An important application is 3D scanning, in which data are acquired at sub-millimeter resolution over large-scale models, potentially resulting in billions of points [LPC*00]. The resulting complexity often exceeds the available computer memory, thus motivating an out-of-core reconstruction algorithm. Existing approaches generally partition the domain into smaller blocks that can be solved locally. However, such partitioning presents several complications. Ideally, surface complexity should adapt to spatially varying point densities, and this is difficult to achieve consistently across block boundaries. Most importantly, the presence of data noise and misalignment makes it difficult to robustly reconstruct a surface by only considering small localized neighborhoods.

Recent work by Kazhdan et al [Kaz05, KBH06] demonstrates that surface reconstruction from oriented points can be made more resilient to data errors by casting the problem as a global Poisson system (Section 3). Intuitively, the idea is to interpret the oriented points as samples of the gradient of the model's indicator function χ (defined as 1 at points inside the model, and 0 at points outside). Thus the desired indicator function is the one whose Laplacian equals the divergence of a vector field $\vec{V}$ constructed from the oriented points: $\Delta\chi = \nabla \cdot \vec{V}$. By representing χ using bases defined over an adaptive octree, the Poisson equation is discretized into a sparse linear system $Lx = b$ whose size is proportional to the complexity of the reconstructed surface. Then, the desired model is an isosurface of the resulting indicator field.

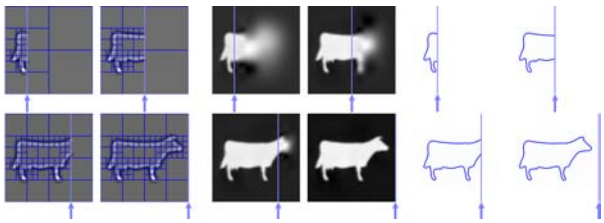

Figure 1: *Example of curve reconstruction as a sequence of three multilevel streaming passes over an adaptive quadtree.*

The fact that Poisson reconstruction has global support would seem to preclude an easy out-of-core solution. Indeed, not only is the matrix L too large to fit in memory, even the vectors b and x are too large. Our contribution is to show that the reconstruction process can be implemented efficiently as a sequence of *streaming* operations over out-of-core data. These operations include the creation of the linear system, its solution, and the final isosurface extraction. The 2D example in Figure 1 helps to illustrate this streaming process.

In general, a streaming approach is advantageous because data is accessed sequentially from disk, and moreover it is

only loaded once. Such sequential access is typically more efficient because it allows for data prefetching. In computer graphics, the concept of streaming computation has been applied to many data types including triangle meshes, point sets, and tetrahedral meshes, as reviewed in Section 2.

A unique aspect of our problem is the requirement for an adaptive multiresolution structure, namely an octree, because a solution over a uniform 3D grid is not scalable [Kaz05]. Interestingly, the operations performed on the octree have different types of inter-level data dependencies, and consequently no single linear ordering of the octree nodes is adequate. To overcome these dependencies, we introduce a *multilevel streaming* representation, in which each resolution level is stored as a separate stream. Thus, a processing pass sweeps over the octree by concurrently advancing through the multiple streams, of course iterating at a faster rate through the finer nodes than the coarser ones. Depending on the operation, information flows up and/or down the tree, and computations on coarser levels precede or succeed those on finer levels.

A surprising result is that we are able to solve the sparse Poisson system $Lx = b$ with sufficient accuracy for our reconstruction application in a *single* multi-stream pass. Two factors make this possible. First, clever scheduling of the computation across levels lets us realize a cascadic multigrid scheme [BK96], which enables fast convergence using only local updates. Second, the reconstructed indicator function χ has high gradient and therefore requires only limited precision due to the subsequent isosurface discretization process.

We obtain reconstructions of highly complex models (210 million triangles) on a PC with only 1 GB of memory, and demonstrate scalable performance.

2. Related Work

Out-of-core surface reconstruction Several surface reconstruction algorithms lend themselves naturally to out-of-core computation because their access patterns are highly localized. For instance, the range-image volumetric merging scheme of Curless and Levoy [CL96] can easily be computed independently on blocks of the domain space. For each block, one conservatively finds the scanned points that contribute to it. Schemes based on local neighborhood fitting such as [HDD*92, ABCO*01] could be computed in a streaming traversal, for instance using the scheme of Pajarola [Paj05]. The multilevel partition of unity (MPU) approach of [OBA*03] uses an adaptive octree structure to blend together estimated implicit surface patches. Its use of local weights should make it amenable to out-of-core processing. The ball-pivoting algorithm of [BMR*99] is implemented out-of-core by partitioning the domain into slices. Our contribution is to consider a global approach that has been demonstrated to improve resilience to data errors [KBH06], and to enable this solution over an out-of-core adaptive octree using a multi-stream scheme.

Stream processing Much of the streaming work in computer graphics addresses irregular triangle meshes [WK03, ILGS03, IL05, ACSE05, AGL06]. An interesting challenge is to find a traversal order that minimizes the working set (bandwidth) of the resulting computation. In practice though, a simple axis-aligned sweep generally works sufficiently well. Streaming operations include surface smoothing, mesh simplification, remeshing and, normal estimation. Streaming has also been applied to irregular tetrahedral mesh compression [ILGS06] and simplification [VCL*07]. Pajarola [Paj05] describes stream processing on points. His streaming scheme is able to find the k-closest neighborhoods of the points, to enable processing operations such as density computation, normal estimation, and geometric smoothing. Isenburg et al. [ILSS06] stream through a set of points to incrementally construct a Delaunay triangulation. Whereas prior streaming methods operate at a single resolution on the data, we introduce a multiresolution streaming framework.

Other out-of-core processing Cignoni et al [CMRS03] introduce an octree-based external memory structure to store an irregular mesh out-of-core. They describe how to handle triangles that span octree cell boundaries. Processing a subtree involves loading its adjacent leaf nodes into memory. Maintaining random access to the octree nodes is beneficial for view-dependent rendering, as also shown in [LS01].

Out-of-core linear solvers Toledo [Tol99] provides a nice survey of methods for solving linear systems out-of-core. For sparse systems, most modern methods assume that the system matrix itself can fit in memory. A common approach is to construct a Cholesky factorization out-of-core (e.g. [GR85]). In our problem, even the solution vector itself is too large to lie in-core. We must therefore resort to simple Jacobi iterative updates. However, we show that doing so in a cascadic multigrid setting, with a per-block Gauss-Seidel scheme, is able to produce adequate accuracy for surface reconstruction, in a single multi-stream pass.

3. Review of Poisson Surface Reconstruction

We begin by reviewing the method of [KBH06]. The input is a set of oriented samples S, where each sample has a position $s.p$ and normal $s.\vec{v}$. The basic idea is to reconstruct a surface from S by estimating the indicator function χ of the model. Kazhdan et al show that the (smoothed) gradient of χ corresponds to a vector field $\vec{V}$ formed by an integral over the (unknown) surface, which can be approximated by a summation over the oriented points. To obtain a least-squares solution of $\nabla\chi = \vec{V}$, the divergence operator is applied to both sides, i.e. $\nabla\cdot\nabla\chi = \nabla\cdot\vec{V}$, resulting in a Poisson equation:

$$\Delta\chi = \nabla\cdot\vec{V}. \tag{1}$$

To represent 3D functions efficiently, Kazhdan et al create an octree $\mathcal{O}$ adapted to the distribution of samples, in which each node $o \in \mathcal{O}$ is associated with a tri-quadratic B-spline

blending function $F_o(p)$, shifted and scaled to align with the node's extent (see also [SL96]). Expressed in this basis,

$$\chi(p) = \sum_{o \in \mathscr{O}} x_o F_o(p) \quad \text{and} \quad \vec{V}(p) = \sum_{o \in \mathscr{O}} \vec{v}_o F_o(p), \tag{2}$$

the Poisson equation reduces to the sparse symmetric system

$$L\,x = b, \tag{3}$$

where $x = \{x_o\}$ and $b = \{b_o\}$ are $|\mathscr{O}|$-dimensional vectors of octree coefficients, the matrix entries are the inner products $L_{o,o'} = \langle F_o, \Delta F_{o'} \rangle$, and the divergence coefficients are

$$b_o = \sum_{o' \in \mathscr{O}} \langle F_o, \nabla \cdot (\vec{v}_{o'} F_{o'}) \rangle . \tag{4}$$

The Laplacian matrix L is sparse because the B-spline functions F are locally supported.

Using a cascadic multigrid solver, Equation 3 is transformed into successive linear systems $L^d x^d = b^d$, one per octree depth d. The solutions at finer depths only consider the *residual* divergence not accounted for at coarser depths. More precisely, the divergence is updated as

$$b_o^d \leftarrow b_o^d - \sum_{d' < d} \sum_{o' \in \mathscr{O}^{d'}} L_{o,o'} x_{o'}, \tag{5}$$

where $\mathscr{O}^d$ denotes the set of octree nodes at depth d.

The octree structure $\mathscr{O}$ and vector field $\vec{V}$ must be constructed to account for the nonuniform distribution of the samples S. This involves computing for each sample s an estimate of its associated width $w(s)$, or more precisely its area term $w^2(s)$ in the surface integral defining $\vec{V}$.

Using the blending function F, a family of kernel density estimators K measures the expected number of samples falling into the ball of radius $w/2$ about p, for all $w > 0$:

$$K(w, p) = \sum_{s \in S} F\left(\frac{p - s.p}{w}\right). \tag{6}$$

A discrete set K^d of such estimators is implemented within the octree by associating a density estimator value k_o to each node, defined by having each sample $s \in S$ distribute a unit value into the eight nearest octree nodes at each octree depth, and setting:

$$K(2^{-d}, p) = K^d(p) \equiv \sum_{o \in \mathscr{O}^d} k_o F_o(p). \tag{7}$$

Using these estimators, the width $w(s)$ associated to each sample is found by solving for $K(w(s), s.p) = \kappa$, where the user-specified desired density κ adjusts the average number of point samples per octree node.

The sample width $w(s)$ is used both to scale the contribution of each sample to the surface integral $\vec{V}$, and to define the spatial extent of that contribution (i.e. the octree level in which it is entered). The vector field approximating the gradient of the indicator function

$$\vec{V}(p) = \sum_{s \in S} \frac{w^2(s)}{w^3(s)} F\left(\frac{p - s.p}{w(s)}\right) s.\vec{n} \tag{8}$$

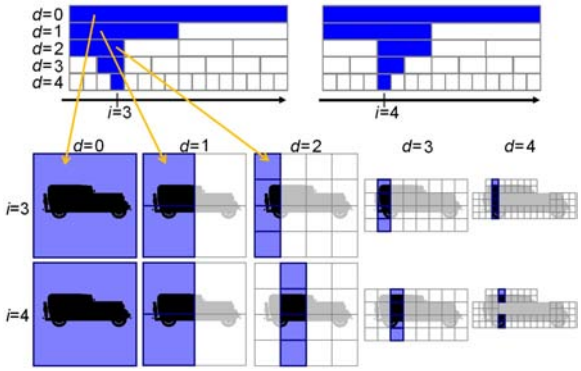

Figure 2: *Illustration of the multilevel stream structure (top row) and the corresponding quadtree nodes (bottom rows) at two moments in time ($i = 3, 4$). In-core blocks and nodes are highlighted in blue.*

where the numerator is the area weight and the denominator normalizes the scaled blending function F such that it has unit integral. To implement this using the octree B-spline basis, the depth of the sample's contribution is defined as $\tilde{d} = \log_2(1/w(s))$ and expressed as the log-based interpolation $\tilde{d} = d_1^{\alpha} \cdot d_2^{1-\alpha}$ of depths $d_1 = \lfloor \tilde{d} \rfloor$ and $d_2 = \lceil \tilde{d} \rceil$. The vector field coefficients $\{\vec{v}_o\}$ are then updated by having the sample splat its normal into the one-ring neighborhood of the nodes $o_1 \in \mathscr{O}^{d_1}$ and $o_2 \in \mathscr{O}^{d_2}$ containing s, weighted by $w^2(s) \cdot (2^{d_1})^3 \alpha$ and $w^2(s) \cdot (2^{d_2})^3 (1 - \alpha)$ respectively.

Finally, to obtain the reconstructed surface, an isovalue is chosen and the corresponding isosurface is extracted using an adaptation of the Marching Cubes algorithm to the octree representation. The isovalue Γ is set to the average of the reconstructed indicator function at the sample positions, weighted by the samples' area. Approximating the contribution of the samples falling into node o by $|\vec{v}_o|$ and evaluating the indicator function at the center of the node, this gives

$$\Gamma = \frac{\sum_{o \in \mathscr{O}} \gamma_o x_o}{\sum_{o \in \mathscr{O}} |\vec{v}_o|} \quad \text{with} \quad \gamma_o = \sum_{o' \in \mathscr{O}} |\vec{v}_{o'}| F_o(o'.\text{center}). \tag{9}$$

4. Our Multi-Stream Octree Representation

In this work, we show that Poisson surface reconstruction can be performed as a sequence of streaming passes over an out-of-core octree representation.

Each streaming pass traverses the octree, sweeping along the x axis. For an octree of height h, each traversal step is associated with a sweep index $0 \le i < 2^{h-1}$ defining the sweep plane $x = (2i + 1)/2^h$. Because streaming computations are local, only the subset of the octree intersecting or near the sweep plane needs to be maintained in main memory. Thus as we advance to sweep index $i + 1$, nodes at the back of the tree (with smaller x coordinates) can be removed from memory, while nodes at the front of the tree need to be loaded in.

To implement a data structure that supports this traversal

pattern, we must address the fact that the in-core persistence of nodes depends on their depth, since coarser nodes are maintained in memory longer than finer ones (see Figure 2). This motivates the construction of a multi-stream octree data structure, consisting of h different streams $\{\mathscr{S}^0,\ldots,\mathscr{S}^{h-1}\}$.

Each stream $\mathscr{S}^d$ contains all nodes $o \in \mathscr{O}^d$, and is partitioned into blocks $\mathscr{S}^d[0],\ldots,\mathscr{S}^d[2^d-1]$ with the nodes in block $\mathscr{S}^d[j]$ all centered on the plane $x = (2j+1)/2^{d+1}$. Thus, at the coarsest depth, $\mathscr{S}^0$ contains only one block $\mathscr{S}^0[0]$ which in turn contains only one node, namely the octree root node. At finer depths, each block $\mathscr{S}^d[j]$ generally contains $O(2^d)$ nodes (out of 2^{2d} nodes in a complete octree) because the surface has co-dimension 1.

Figure 2 shows a visualization of the multi-stream structure for a quadtree representation. Each row marked with a depth $d = 0\ldots4$ corresponds to a separate stream $\mathscr{S}^d$, and within a row the rectangles denote the blocks $\mathscr{S}^d[j]$. In the top left diagram, we see the data structure at sweep index $i = 3$. The in-core blocks are highlighted in blue, corresponding to all the quadtree nodes that intersect the sweep-plane as shown in the middle diagram. Note that as we advance to sweep index $i = 4$ (shown in top right and bottom diagrams), not all streams need to be updated; in this example, it is only the streams at depths $d = 2,3,4$ that are advanced.

At index i, the sweep plane intersects the nodes contained in the blocks $\mathscr{S}^d[\lfloor i/2^{h-d-1}\rfloor]$, which we denote by $\mathscr{S}^d[\phi_d(i)]$, or simply as $\mathscr{S}^d_i$. More generally, stream processing operations may require access to nodes in a small neighborhood of the sweep plane. If the operation needs access to a k-neighborhood at each depth, we maintain an *in-core octree* $\mathscr{O}_{i,k} \subset \mathscr{O}$ defined as the union

$$\mathscr{O}_{i,k} = \bigcup_{d=0}^{h-1} \mathscr{S}^d_{i,k} \quad \text{where} \quad \mathscr{S}^d_{i,k} = \bigcup_{j=-k}^{k} \mathscr{S}^d[\phi_d(i)+j].$$

Thus Figure 2 can be seen to correspond to $\mathscr{O}_{i,0}$ at $i = 3,4$.

An essential property of the in-core octree is that for any node $o \in \mathscr{S}^d_i$ and any depth $d' \le d$, the k-neighborhood of the ancestor of o at depth d', denoted $N^{d'}_k(o)$, is guaranteed to be contained in $\mathscr{O}_{i,k}$, i.e. to be in-core.

As the sweep index is advanced from i to $i+1$, the in-core octree $\mathscr{O}_{i,k}$ is updated. Specifically, we compute the set of depths, D_i, at which the streams need to be advanced:

$$D_i = \{d \mid \phi_d(i) \neq \phi_d(i+1)\}$$

and for each $d \in D_i$ we can unload the block $\mathscr{S}^d[\phi_d(i)-k]$ and load the block $\mathscr{S}^d[\phi_d(i)+k+1]$ into memory.

Implementation We store each stream in a separate file and, using a 64-bit operating system, are able to reserve contiguous blocks of virtual address space large enough to fully span the streams. An advantage of using virtual addressing is that, by simple addition with a base address, a pointer to a node can be represented by the node's offset in the file.

Within each stream, an "active window" between a head pointer and a tail pointer is mapped to physical memory. To efficiently update these pointers during the sweep, we store the offset and extent of all blocks in an index structure, which forms a complete binary tree of height h.

Although we exploit virtual memory addressing, we never rely on the operating system for demand-based paging, as this can be inefficient. Instead we explicitly manage the memory mapping. As the head pointer advances through a stream, the appropriate pages of virtual memory are committed to physical memory and read from disk. And, as the tail pointer advances, dirty data is written to disk and memory pages are uncommitted. Memory management and I/O are performed asynchronously by a background thread, to allow for lazy write-back and anticipatory read-ahead. All I/O is performed at the granularity of 1 MB to maximize disk bandwidth and minimize disk seek overhead.

Additionally, we vertically partition the data for each depth into two separate streamed files that are advanced in lockstep, one containing the octree topology, and the other containing the octree data $(k_o, \vec{v}_o, b_o, \gamma_o, x_o)$. Since the first file becomes read-only after creation, it doesn't need to be written back to disk in subsequent passes, thereby reducing the I/O workload.

5. Streaming Surface Reconstruction

We now describe how Poisson surface reconstruction can be decomposed into a sequence of streaming passes (Figure 3). The focus here is to demonstrate that, thanks to the compact support of the basis functions F_o, each step of the reconstruction process involves local computation, and can therefore be implemented as a streaming pass. In Section 6 we show how these individual steps can be combined more efficiently into just three passes over the out-of-core data.

The discussion in this section is guided by Table 1, which summarizes the extent of the data that needs to be in-core to process block $\mathscr{S}^d_i$ in each step of reconstruction. The key property that enables streaming reconstruction is that this data extent is always bounded by a neighborhood k at each depth, and therefore all the necessary data is available if we maintain an in-core octree $\mathscr{O}_{i,k}$ as we sweep over index i.

We briefly review the individual steps of the reconstruction process, providing the value of the neighborhood k that defines the size of the necessary in-core octree $\mathscr{O}_{i,k}$.

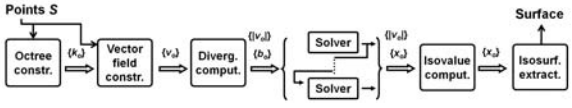

Figure 3: *Sequence of streaming passes through the out-of-core octree data, as described in the naive implementation of Section 5.*

Step	Read	Write
$\mathscr{O}$ constr. (d=h–1)	S_i	k_o: $\mathscr{S}_{i,1}^{d'}$ $d'\leq d$
$\vec{V}$ constr.	k_o: $\mathscr{S}_{i,1}^{d}$, S_j $\phi_d(j)=\phi_d(i)$	$\vec{v}_o$: $\mathscr{S}_{i,1}^{d}$
$\nabla\cdot\vec{V}$ distr.	$\vec{v}_o$: $\mathscr{S}_i^d$	b_o: $\mathscr{S}_{i,2}^{d'}$ $d'\leq d$
Γ distr.	$\lvert\vec{v}_o\rvert$: $\mathscr{S}_i^d$	γ_o: $\mathscr{S}_{i,1}^{d'}$ $d'\leq d$
Γ accum.	$\lvert\vec{v}_o\rvert$: $\mathscr{S}_{i,1}^{d'}$ $d'<d$	γ_o: $\mathscr{S}_i^d$
$\nabla\cdot\vec{V}$ accum.	$\vec{v}_o$: $\mathscr{S}_{i,2}^{d'}$ $d'<d$	b_o: $\mathscr{S}_i^d$
$\nabla\cdot\vec{V}$ update	x_o: $\mathscr{S}_{i,2}^{d'}$ $d'<d$	b_o: $\mathscr{S}_i^d$
Δ solution	b_o: $\mathscr{S}_{i,2}^{d}$	x_o: $\mathscr{S}_i^d$
Γ comput.	$\lvert\vec{v}_o\rvert, x_o, \gamma_o$: $\mathscr{S}_i^d$	isovalue Γ
Isosurface extr.	x_o: $\mathscr{S}_{i,2}^{d}$, Γ	surface mesh

Table 1: *Read and write operations when processing block $\mathscr{S}_i^d$ in the various multilevel streaming computations.*

Preprocessing We first rotate the point set so that the dominant axis of its covariance matrix is aligned with the x-axis. The intent is to reduce the cross-section complexity encountered during the sweep, and hence the peak memory size of the in-core octree $\mathscr{O}_{i,k}$. We then uniformly scale and translate the points so that they fit into the unit cube. Finally, we partition the points into subsets $S_i \subset S$, whose x-coordinates lie in the range $[i/2^{h-1}, (i+1)/2^{h-1}]$. This partitioning process is essentially a binning process, and is implemented efficiently as a single-input, multiple-output streaming operation.

Octree Construction ($k=1$) At index i we read in the subset of points $S_i \subset S$. For each $s \in S_i$ and every depth d, we refine the in-core octree so that the node $o^d(s) \in \mathscr{O}^d$ containing s and its one-ring neighbors are all present in $\mathscr{O}_{i,1}$, adding new nodes as necessary. We also update the density estimator coefficients $\{k_o\}$ by having each sample s splat a unit value into the one-ring neighborhood of $o^d(s)$.

Vector Field Construction ($k=1$) At index i we iterate over all samples $s \in S_i$. For each s, we evaluate the density estimators K^d to determine the sample width $w(s)$, compute the corresponding depths d_1 and d_2, and splat the sample's (weighted) normal into the one-ring neighborhoods $o^{d_1}(s)$ and $o^{d_2}(s)$ to update the vector field coefficients $\{\vec{v}_o\}$.

Because F_o is supported in a one-ring neighborhood of o, $K^d(s.p)$ can be evaluated without access to $k_{o'}$ for $o' \notin \mathscr{O}_{i,1}$.

Divergence Computation ($k=2$) Since processing a node $o \in \mathscr{S}_i^d$, we only have $N_k^{d'}(o) \subset \mathscr{O}_{i,k}$ for $d' \leq d$ in the working set, we decompose the divergence computation into two steps. Following Equation 4, at sweep index i:

We *distribute* divergence to nodes at depths $d' \leq d$ by iterating over $o' \in N_2^{d'}(o)$ and adding $\langle \nabla\cdot(\vec{v}_o F_o), F_{o'}\rangle$ to $b_{o'}$.

We *accumulate* divergence from nodes at depths $d' < d$ by iterating over $o' \in N_2^{d'}(o)$ and adding $\langle \nabla\cdot(\vec{v}_{o'} F_{o'}), F_o\rangle$ to b_o.

Because F_o is supported in a one-ring neighborhood of o, $\langle \nabla\cdot(\vec{v}_{o'} F_{o'}), F_o\rangle \neq 0$ only if $o \in N_2^{d'}(o)$, and both b_o and $b_{o'}$ can be incremented without access to $v_{o'}$ for $o' \notin \mathscr{O}_{i,2}$.

Poisson System Solution ($k=2$) The most straightforward implementation of the cascadic multigrid algorithm performs two streaming passes for each depth $0 \leq d < h$ (from coarsest to finest), first updating b^d in the linear system $L^d x^d = b^d$ using the solution at depths $d' < d$, and then solving the system. We describe such an approach, and later in Section 6 show that it is possible to perform *all* these $2h$ passes in a *single* multilevel streaming pass.

We *update* the divergence coefficients b_o for $o \in \mathscr{S}_i^d$ by iterating over $o' \in N_2^{d'}(o)$ for all $d' < d$ and subtracting the value $x_{o'} L_{o,o'}$ from b_o (following Equation 5).

We *solve* for the values x_o with $o \in \mathscr{S}_i^d$ by performing several iterations over the nodes in $\mathscr{S}_i^d$ and, for each node o, performing the Jacobi update:

$$x_o \leftarrow \frac{b_o - \sum_{o' \in \mathscr{O}^d} L_{o,o'} x_{o'}}{L_{o,o}}.$$

Because F_o is supported in a one-ring neighborhood of o, $L_{o,o'} \neq 0$ only if $o' \in N_2^{d'}(o)$ so updating b_o and solving for x_o can be done without access to $x_{o'}$ for $o' \notin \mathscr{O}_{i,2}$.

Computing the Isovalue ($k=1$) Since processing a node $o \in \mathscr{S}_i^d$, we only have $N_k^{d'}(o) \subset \mathscr{O}_{i,k}$ for $d' \leq d$ in the working set, we decompose the isovalue computation into three steps. Following Equation 9, at sweep index i:

We *accumulate* the isovalue from nodes at depths $d' \leq d$ by iterating over $o' \in N_2^{d'}(o)$ and adding $|\vec{v}_o| F_o(o'.\text{center})$ to γ_o.

We *distribute* the isovalue to nodes at depths $d' < d$ by iterating over $o' \in N_2^{d'}(o)$ and adding $|\vec{v}_{o'}| F_{o'}(o.\text{center})$ to $\gamma_{o'}$.

We *compute* the isovalue by adding $x_o \gamma_o$ to the numerator of Γ and adding $|\vec{v}_o|$ to the denominator.

Extracting the Isosurface ($k=2$) We extract the isosurface by iterating over the leaf nodes, computing the value of χ at the eight cell corners, solving for the positions of Γ-crossings along the edges, and extracting the triangulation.

The challenge in implementing the isosurface extraction is the evaluation of χ at the corners of a leaf node $o \in \mathscr{S}_i^d$. Since the value at a corner can be determined by the values of $x_{o'} \in \mathscr{O}^{d'}$ with $d' > d$, we are not guaranteed to have the necessary information in-core when processing the node o.

To address this challenge we observe that because the functions $F_{o'}$ are supported in the one-ring neighborhood of o', for a corner $c \in o$ we have $F_{o'}(c) \neq 0$ only if either $d' \leq d$ and $o' \in N_1^{d'}(o)$, or $d' > d$ and c is also a corner of o'. Thus, when o is the finest node adjacent to corner c, $\chi(c)$ can be computed using only values $x_{o'}$ for $o' \in N^{d'}(o)$ and $d' \leq d$.

This observation motivates an algorithm for isosurface extraction that iterates over the leaf nodes from finest to coars-

est and stores the evaluation of χ at the corners in a temporary hash table. For a given corner c of a leaf node $o \in \mathscr{O}^d$, we check if there is an entry in the hash table corresponding to c. If there is not, this implies that there are no nodes at depth $d' > d$ containing c as a corner and the value $\chi(c)$ can be computed using only information associated to nodes in the one-ring neighborhood of the ancestors of node o.

In practice, separate hash tables are associated with the corners of the front and back of the leaf nodes at each depth. As the sweep plane is advanced, the front hash table is updated by evaluating the front corners of leaf nodes intersecting the sweep plane and the back corners of leaf nodes immediately in front of the sweep plane. For a corner $c \in o$ that is also a corner of a node $o' \in \mathscr{O}^{d-1}$, we add the value $\chi(c)$ to the front hash table at depth $d-1$. Finally, after extracting the isosurface in the current sweep index, we swap the front and back hash tables and clear the front one. It is also at this point that vertices are finalized shortciteIsenburg:VIS:2005. We write to a block-based streaming mesh format.

6. Optimized Implementation

In the previous section, we showed that the locality of the Poisson reconstruction steps allows for stream processing. In this section, we show how the different streaming passes can be merged into three multilevel streaming passes, with the passes defined as follows:

Pass 1: Octree construction, vector field construction, divergence distribution, and isovalue distribution
Pass 2: Isovalue accumulation, divergence accumulation, divergence update, Poisson system solution and isovalue computation
Pass 3: Isosurface extraction

Our approach is motivated by two observations. First, we can parallelize streaming steps when there are no data dependencies. Second, even when there are dependencies, we may be able to pipeline the steps, resolving the dependencies with only a small increase in the size of the working set.

Due to the data dependencies, three passes are a lower-bound for our reconstruction algorithm: The fine-to-coarse distribution of the divergence field b_o (in pass 1) must be finalized before the coarse-to-fine cascadic multigrid solution (in pass 2), and the computation of the isovalue (in pass 2) must be finalized before the isosurface extraction (in pass 3).

6.1. First Pass ($k = 6$)

To merge the processing steps in the first pass, we must resolve the data dependencies between different steps. We do this by pipelining the steps, delaying execution of later steps to allow earlier steps to finalize the dependent data.

Using the sizes of the read/write neighborhoods described in Table 1, we can resolve the data dependencies in the first pass by iterating over the sweep indices, for each i:

- Constructing the octree for $\mathscr{S}^{h-1}[i+5]$

and for each $d \in D_i$

- Constructing the vector field for $\mathscr{S}^d[\phi_d(i)]$
- Distributing the divergence for $\mathscr{S}^d[\phi_d(i)-3]$
- Distributing the isovalue for $\mathscr{S}^d[\phi_d(i)-3]$

Taking into account the size of the write neighborhoods for octree construction and divergence distribution, the first pass of streaming reconstruction can be implemented by maintaining the octree $\mathscr{O}_{i,6}$ in the working set at sweep index i.

Buffering Samples In addition to maintaining a small working octree, our method must also address the fact that to implement the vector field construction for block $\mathscr{S}_i^d$ the processing step needs access to each sample which lies in the span of $\mathscr{S}_i^d$ and has failed the density test at greater depths.

The exhaustive testing of all samples which lie in the span of $\mathscr{S}_i^d$ can be a computational bottleneck for our system since it requires h passes through the ordered point set. This is unnecessarily expensive since we expect a sample's density estimate to increase by a factor of four as the depth is decremented, so the number of samples processed at depth d but failing the density test should drop by a factor of four, while the number of samples that lie in the span of $\mathscr{S}_i^{d-1}$ should only increase by a factor of two.

We address this concern by associating a sample buffer to each depth and processing the blocks in decreasing depth order. Samples are added into the buffer at depth h during the octree construction step and are promoted to the buffer at depth $d-1$ if they fail the density test at depth d in the vector field construction step. (Points in the depth-d buffer that lie in the span of $\mathscr{S}^d[\phi_d(i)]$ are removed from the buffer at the end of the vector field construction step.)

6.2. Second Pass ($k = 8$)

As in the first pass, we merge the steps in the second pass by pipelining them to resolve data dependencies. However, since the consolidation of these steps into a single pass forces us to iterate over the depths before iterating over sweep indices, the merging of the divergence update with the Poisson system solution poses a challenge. For a fixed sweep index, we can no longer treat the individual steps as atomic because this would result in a circular data dependency: the modification of $\{b_o\}$ in the divergence update requiring access to $\{x_o\}$ set in the Poisson system solver, which in turn requires access to $\{b_o\}$.

We resolve this problem by separately considering the pipelining that needs to be performed to resolve the data dependencies due to sweep index and due to depth.

Index Dependencies Fixing a depth d and assuming no cross-depth data dependencies, we define the scheduling as we did in the first pass. Iterating over the (depth-relative) sweep index i^d, with $0 \le i^d < 2^d$, we:

- Accumulate the isovalue for $\mathscr{S}^d[i^d]$
- Accumulate the divergence for $\mathscr{S}^d[i^d]$

Model	#Points	h	#Triangles	Time	Peak mem.	Stream
Lucy Statue	95M	12	26.2M	3.1	138	5,135
David Head	216M	13	210M	32.3	780	62,464
Awakening	391M	13	149M	26.6	990	35,840
Awakening	391M	14	431M	82.4	2120	106,496

Table 2: *Quantitative results for multilevel streaming reconstructions, showing input points, octree height h, output mesh triangles, total execution time (hours), memory use (MB), and total octree stream size (MB).*

Figure 4: *Comparing the results of the in-core algorithm (left; $h=11$; 4,442 MB peak memory) and streaming algorithm (right; $h=13$; 780 MB peak memory).*

- Update the divergence for $\mathscr{S}^d[i^d]$
- Solve the Poisson system for $\mathscr{S}^d[i^d-3]$
- Compute the isovalue for $\mathscr{S}^d[i^d-4]$

Depth Dependencies To resolve the depth-related dependencies we offset the values of i^d so that values required at finer depths are guaranteed to have been set at coarser ones.

Analyzing the size of the read/write neighborhoods shows that the dependencies can be resolved if the indices satisfy the property $i^{d-1} \geq \lfloor i^d/2 \rfloor + 6$. Expressing i^d as an offset from the finest index, $i^d = \phi_d(i^{h-1}) + \delta^d$, and initializing with $\delta^{h-1} = 0$, we obtain a recursive expression for the offsets: $\vec{\delta}^d = \{11,\ldots,11,10,9,6,0\}$. Thus, setting $i^{h-1} = i-3$, the second reconstruction pass can be implemented by maintaining the octree $\mathscr{O}_{i,8}$ in the working set at sweep index i.

In practice, we can further reduce the memory requirements by observing that processing at the finest depths requires a narrower window size. This allows us to maintain a working octree with fewer stream blocks at the finest depths.

Figure 1 shows an example of the three streaming passes for the reconstruction of 2D point set, showing the state of the reconstruction at different sweep indices (indicated by the arrows). As can be seen, the offsetting of the pipeline steps in the second pass forces coarser nodes to be solved ahead of the sweep line, resulting in a lower resolution reconstruction emerging to the right of the sweep index.

7. Results

Large Datasets To evaluate our method, we have reconstructed highly detailed surfaces from large scanned datasets, as summarized in Table 2. All results use a target of $\kappa = 2$ samples per octree node.

Res.	Octree Size		Peak Memory		Running time	
	In-core	Streaming	In-core	Streaming	In-core	Streaming
256	49	48	309	521	0.50	0.53
512	188	168	442	278	0.65	0.68
1024	818	702	1285	213	1.05	1.20
2048	3,695	3,070	4,442	212	2.65	3.33
4096	n/a	13,367	n/a	427	n/a	12.6
8192	n/a	39,452	n/a	780	n/a	32.3

Table 3: *Comparison of the data structure size (MB), peak working set (MB), and running time (hours) for the in-core and streaming reconstruction algorithms over a range of resolutions for the David Head model. Running the in-core algorithm beyond a resolution of 2048 was impossible due to its high memory requirements.*

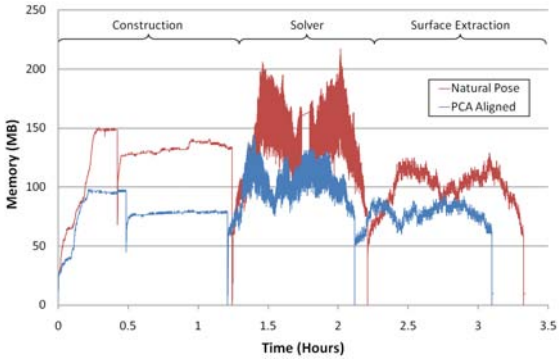

Figure 5: *Memory use over time for a depth 12 reconstruction of the Lucy statue using two different poses of the model.*

Figure 6 shows a surface reconstruction of the Michelangelo's David statue from an input of 216M oriented points from raw scan data. The output surface of 210M triangles was generated at maximum octree depth h=13, and required only 780 MB of memory. In contrast, the in-core Poisson reconstruction of [KBH06] only produced a 20M triangle approximation of this same model (at depth 11), and required 4.4 GB of memory. Figure 4 shows a close-up visual comparison.

As another example of our algorithm's ability to reconstruct large models, Figure 7 presents a reconstruction of Michelangelo's Awakening statue from 391M points from raw scan data. At a maximum depth of h=14, our streaming solution produced a mesh of 431M triangles in 82 hours. Although the storage required for the out-of-core data structure was 104 GB, our reconstruction algorithm never required more than 2.1 GB of working memory. Reconstructions at this resolution allow us to clearly see fine detail such as chisel markings that could not be seen at lower resolutions.

Scalable Memory Use Each of our three multilevel streaming passes only maintains a small window on the entire data structure at any one time. Figure 8 examines how

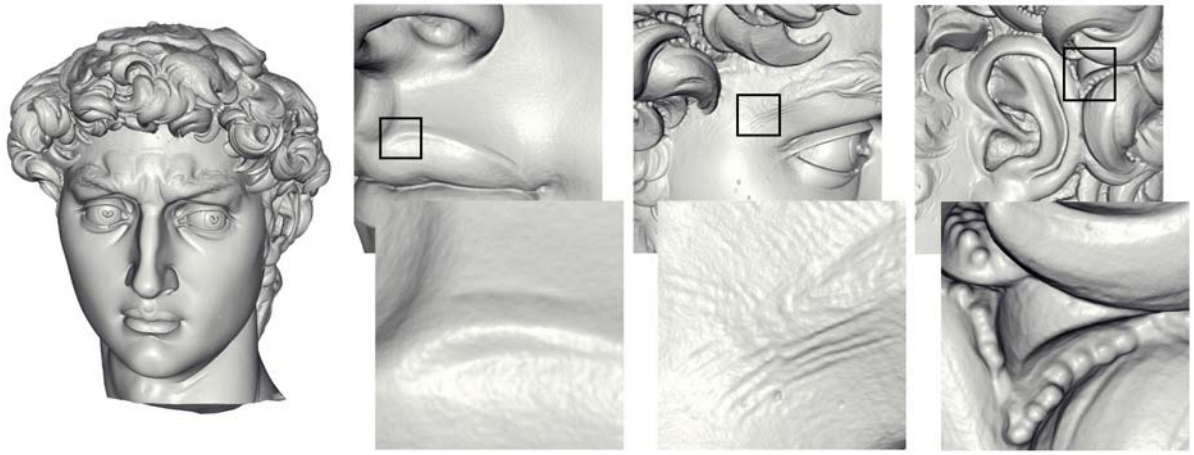

Figure 6: *Views of our reconstruction of the head of Michelangelo's David. Maximum tree depth was 13, with a target of 2 samples per node.*

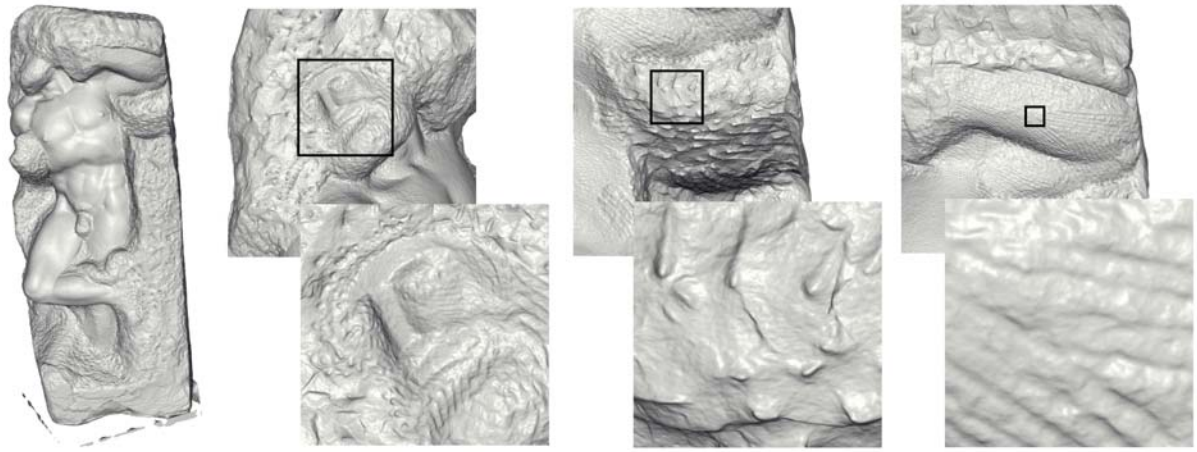

Figure 7: *Views of our reconstruction of Michelangelo's Awakening statue. Maximum tree depth was 14 with a target of 2 samples per node.*

the maximum size of these windows varies with output resolution. By comparison, the curve for the in-core algorithm grows so quickly that it exits the graph on the upper left.

Table 3 shows the octree size and peak memory use as a function of the resolution ($r = 2^h$) of the octree. As expected, the total octree size has complexity $O(r^2)$ since the surface has co-dimension 1. However, using the streaming reconstruction, the size of the in-core window only scales as $O(r)$, allowing the streaming algorithm to process datasets that far exceed a system's main memory capacity.

The unexpectedly large memory use for the coarser resolutions is due to the buffering of points that occurs during octree construction. When the tree is artificially restricted to a small depth, many more points fall into the bins S_i traversed at each sweep step. However, this is an atypical scenario.

Memory use is further highlighted in Figure 5, which plots memory use over time through each of the three multilevel streaming passes during the reconstruction of the Lucy statue. The two different plot curves show how the sweep plane orientation can affect performance. The red curve corresponds to using the x-axis as the sweep direction, with the statue oriented in its original vertical pose; in this orientation, the intersection of the surface with the sweep plane can be large, resulting in a peak memory use of 223 MB. The blue curve corresponds to using the dominant principal direction of the point set as the sweep direction; such orientation reduces the intersection of the sweep plane with the surface, resulting in a peak memory use of only 138 MB.

The graph also shows that the three multilevel streaming passes have similar memory requirements and running times. The graph curves do not include the preprocess operations of orienting, scaling, and binning the points. However,

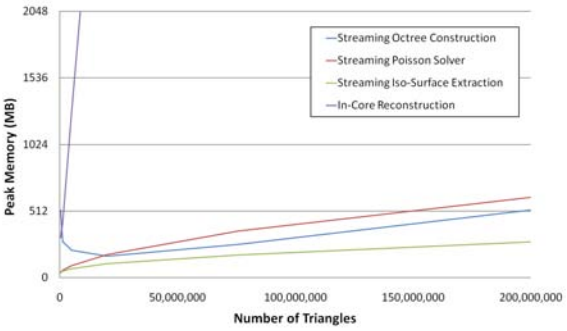

Figure 8: *The peak working set in our 3 multilevel streaming passes, and in the in-core algorithm (far left), for a range of reconstructions of the head of Michelangelo's David.*

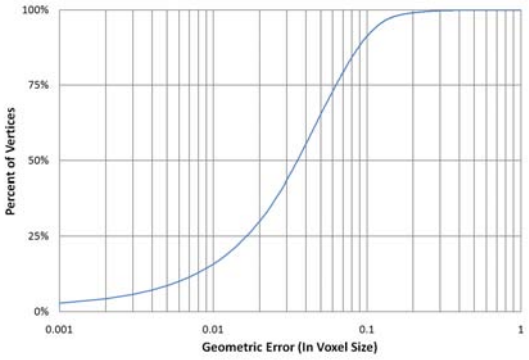

Figure 9: *The cumulative distribution of geometric error for a depth 12 reconstruction of the Lucy statue when compared to the in-core algorithm of [KBH06].*

this preprocess is negligible as it requires only about 1% of the total execution time and uses less memory than the multilevel streaming passes.

Computation Times Table 3 reveals that our streaming algorithm is time-competitive with the in-core algorithm despite the large amount of I/O. The streaming overhead is small because the overall process is compute-bound and the stream read-ahead prevents stalls in computation.

Streaming Solver Accuracy Because our streaming solver computes only an approximate solution to the Poisson equation, the numerical accuracy of the solution could impact the geometric accuracy of the resulting surface mesh. (This topic is further discussed in Section 8.) To test geometric accuracy, we compare the surface mesh generated by our streaming algorithm to that generated by the in-core algorithm of [KBH06]. Figure 9 graphs the cumulative distribution of mesh vertices as a function of their geometric error, measured as the distance in voxel units to the nearest point on the reference surface. Despite the fact that our streaming cascadic multigrid performs only a single sweep at each

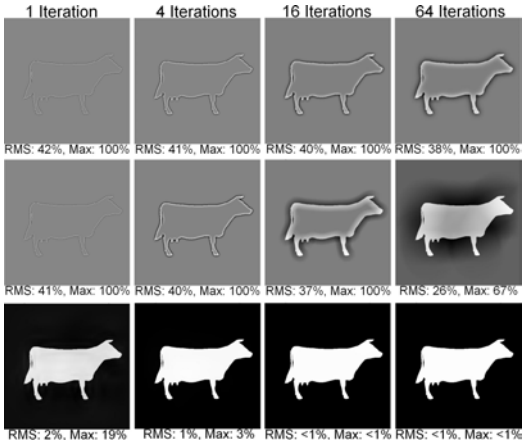

Figure 10: *Comparison of reconstructing the indicator function of a cow silhouette from its Laplacian using a single-resolution streaming solver (top), a traditional conjugate-gradient solver (middle), and a cascadic multigrid solver using multilevel streaming (bottom).*

level, the resulting surface mesh is still very accurate – only 11% of the vertices have an error greater than 0.1 voxels, and the maximum error is 0.842 voxels.

8. Discussion

Solving the Poisson system in streaming fashion is a challenging task since it involves a global linear system in which Laplacian values at one point affect the solution at points faraway. The key ingredient that enables an effective streaming solution is the use of a cascading multigrid approach.

To demonstrate the importance of multigrid, Figure 10 shows the quality of solutions to a 2D Poisson problem using three different techniques. The first row shows the reconstructions obtained with 1, 4, 16, and 64 iterations of a block-based Gauss-Seidel solver that streams through the column blocks of the image, much like one of the *single-level* streaming passes described in Section 5. As shown in the second row, even if we replace the Gauss-Seidel solver with the more efficient (but non-streaming) conjugate-gradient solver, the convergence is still too slow, requiring at least 64 passes through the data to obtain an approximate solution. In contrast, a cascadic multigrid solver (bottom row) quickly converges to the indicator function.

For general problems, a multigrid solver typically requires several Gauss-Seidel iterations per level, which would involve several streaming passes, but remarkably for our reconstruction problem a single pass is usually sufficient. The intuition is that, in the context of surface reconstruction, the Poisson solution χ approximates an indicator function, and is thus only used to identify the boundary between interior and exterior. Because the indicator function is a binary function whose value is either 0 or 1, and the isovalue is approximately 0.5, the reconstruction is sufficiently accurate if it

never differs by more than 0.5 from the indicator function. As shown in the bottom left reconstruction of Figure 10 (and also earlier in Figure 9), this relaxed error condition can be met with just one iteration per level of the cascadic multigrid solver, allowing us to perform a single streaming pass at each level. And, one of our key algorithmic contributions is to show that all such passes can be combined into a single multilevel streaming pass.

9. Conclusion and Future Work

Streaming computation is an effective tool for processing huge out-of-core datasets. We have shown that such a framework can be extended to multiresolution computation, including global Poisson solution over an adaptive octree structure in the context of surface reconstruction.

Avenues for future work include:

- Application of multilevel streaming to out-of-core processing of multi-gigapixel images.
- Support for multicore parallel processing.
- Generalization to processing of higher-dimensional datasets such as 4D time-varying volumes.

Acknowledgements

We would like to acknowledge the Stanford 3D Scanning Repository for generously distributing their data. The authors would also like to express particular thanks to Szymon Rusinkiewicz and Benedict Brown for providing non-rigid body aligned Awakening and David scan data [BR07].

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C.: Point set surfaces. In *Proceedings of the Conference on Visualization '01* (2001).

[ACSE05] ATTALI D., COHEN-STEINER D., EDELSBRUNNER H.: Extraction and simplification of iso-surfaces in tandem. In *Symposium on Geometry Processing* (2005).

[AGL06] AHN M., GUSKOV I., LEE S.: Out-of-core remeshing of large polygonal meshes. In *Visualization 2006* (2006).

[BK96] BORNEMANN F., KRAUSE R.: Classical and cascadic multigrid – a methodological comparison. In *Proceedings of the 9th International Conference on Domain Decomposition Methods* (1996).

[BMR*99] BERNARDINI F., MITTLEMAN J., RUSHMEIER H., SILVA C., TAUBIN G.: The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics 5* (1999).

[BR07] BROWN B., RUSINKIEWICZ S.: Global non-rigid alignment of 3-D scans. In *Proceedings of SIGGRAPH 2007* (2007).

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. *Computer Graphics (Proceedings of SIGGRAPH 96)* (1996).

[CMRS03] CIGNONI P., MONTANI C., ROCCHINI C., SCOPIGNO R.: External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics 9* (2003).

[GR85] GEORGE A., RASHWAN H.: Auxiliary storage methods for solving finite element systems. *SIAM Journal on Scientific and Statistical Computing 6* (1985).

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. *Computer Graphics 26* (1992).

[IL05] ISENBURG M., LINDSTROM P.: Streaming meshes. In *Proceedings of the Conference on Visualization '05* (2005).

[ILGS03] ISENBURG M., LINDSTROM P., GUMHOLD S., SNOEYINK J.: Large mesh simplification using processing sequences. In *Proceedings of the Conference on Visualization '03* (2003).

[ILGS06] ISENBURG M., LINDSTROM P., GUMHOLD S., SHEWCHUK J.: Streaming compression of tetrahedral volume meshes. In *Proceedings of Graphics Interface 2006* (2006).

[ILSS06] ISENBURG M., LIU Y., SHEWCHUK J., SNOEYINK J.: Streaming computation of Delaunay triangulations. *ACM Transactions on Graphics 25* (2006).

[Kaz05] KAZHDAN M.: Reconstruction of solid models from oriented point sets. *Symposium on Geometry Processing* (2005).

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. *Symposium on Geometry Processing* (2006).

[LPC*00] LEVOY M., PULLI K., CURLESS B., RUSINKIEWICZ S., KOLLER D., PEREIRA L., GINZTON M., ANDERSON S., DAVIS J., GINSBERG J., SHADE J., FULK D.: The digital Michelangelo project: 3D scanning of large statues. *SIGGRAPH* (2000).

[LS01] LINDSTROM P., SILVA C.: A memory insensitive technique for large model simplification. In *Proceedings of the Conference on Visualization '01* (2001).

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.-P.: Multi-level partition of unity implicits. *ACM Transactions on Graphics* (2003).

[Paj05] PAJAROLA R.: Stream-processing points. In *Proceedings of the Conference on Visualization '05* (2005).

[SL96] SZELISKI R., LAVALLEE S.: Matching 3-D anatomical surfaces with non-rigid deformations using octree-splines. *International Journal of Computer Vision 18* (1996).

[Tol99] TOLEDO S.: A survey of out-of-core algorithms in numerical linear algebra. In *External Memory Algorithms and Visualization*, Abello J., Vitter J., (Eds.). American Mathematical Society Press, 1999.

[VCL*07] VO H., CALLAHAN S., LINDSTROM P., PASCUCCI V., SILVA C.: Streaming simplification of tetrahedral meshes. *IEEE Transactions on Visualization and Computer Graphics 13* (2007).

[WK03] WU J., KOBBELT L.: A stream algorithm for the decimation of massive meshes. In *Proceedings of the Conference on Graphics Interface '03* (2003).