Efficiently Distributed Watertight Surface Reconstruction

Laurent Caraffa*, Yanis Marchand*, Mathieu Brédif, Bruno Vallet
LASTIG, Univ Gustave Eiffel, ENSG IGN
F-94160 Saint-Mande, France
{name.surname}@ign.fr

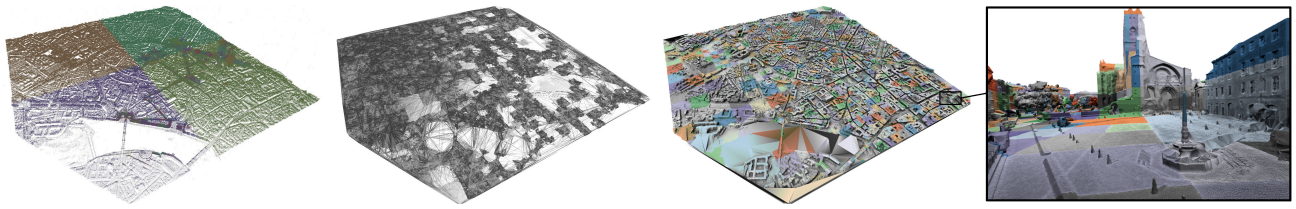


Figure 1: Main steps of the distributed algorithm. From left to right: multiple input point clouds (aerial and terrestrial), partial view of the distributed Delaunay triangulation, large and close-up view of the resulting watertight surface (3D tiles are in different colours).

Abstract

We present an out-of-core and distributed surface reconstruction algorithm which scales efficiently on arbitrarily large point clouds (with optical centres) and produces a 3D watertight triangle mesh representing the surface of the underlying scene. Surface reconstruction from a point cloud is a difficult problem and existing state of the art approaches are usually based on complex pipelines making use of global algorithms (i.e. Delaunay triangulation, graph-cut optimisation). For one of these approaches, we investigate the distribution of all the steps (in particular Delaunay triangulation and graph-cut optimisation) in order to propose a fully scalable method. We show that the problem can be tiled and distributed across a cloud or a cluster of PCs by paying a careful attention to the interactions between tiles and using Spark computing framework. We confirm the efficiency of this approach with an in-depth quantitative evaluation and the successful reconstruction of a surface from a very large data set which combines more than 350 million aerial and terrestrial LiDAR points.

1. Introduction

Surface reconstruction is the task of producing a digital and continuous representation of the surface of individual objects or entire scenes from sensing data (mostly images and LiDAR scans). This problem has been thoroughly studied judging by the number of approaches surveyed by [3] and [20]. This appeal can be explained by the rise in sensor capabilities (in particular number of pixels and pulse rate by cameras and LiDAR scanners) and the diversification of sensing platforms (mobile mapping, drones,). As highlighted in [3], surface reconstruction methods differ by:

Point Cloud Artefacts: The imperfections of the point cloud that the method is able to handle such as non-uniform sampling, noise, outliers, misalignment, missing data/holes (in particular due to occlusions)...

Input Requirements: The type of inputs associated with a point cloud required by the algorithm such as oriented or unoriented normals, sensor information (optical centre position and/or 2D lattice).

Shape Class: The class of shape that the method is capable of reconstructing such as organic, man-made, indoor/outdoor scene, architectural... The shape class is important to define priors that can help handle the aforementioned artefacts.

Reconstruction Output: The representation of the reconstructed surface such as triangle meshes, voxel grids, implicit fields, point sets, deformed models, skeleton curves, primitives...

Watertightness: A surface is watertight if it has no border. In the case of a triangle mesh, this means each edge needs to have exactly two incident faces.

Scalability: The ability to distribute the algorithm on multiple computers/cores in order to handle massive amounts of data and still maintain a reasonable computing time and

* The authors contributed equally to this paper

a memory footprint compatible with the hardware at hand, **without impairing the quality of the reconstruction**. In other terms, assuming that a *reference surface* was reconstructed from the whole data with the *reference* (not distributed) *algorithm* on a computer with sufficient RAM and computing power, we want to ensure that the surface produced by the *distributed algorithm* is close enough to this reference surface. This last point is the main focus of this paper. Some existing approaches scale up by splitting the input point cloud in appropriately sized tiles with some overlaps. This raises two issues:
1) There is no guarantee that the resulting surface matches the reference one, in particular when there are holes/missing data on the overlaps that might be filled very differently in the overlapping tiles.
2) There is no guarantee of surface watertightness as meshes do not even coincide exactly on the overlaps. This might be very problematic for applications requiring watertightness (such as physical simulations, and in particular flooding simulation) but also leads to visual artefacts.

Our major contribution is the first Delaunay-based watertight surface reconstruction algorithm that can handle arbitrarily large point clouds. Poisson surface reconstruction [7] is the only other known approach that scales while ensuring watertightness but it is not suited to urban scenes as it fails to reconstruct sharp edges. Recently, [16] proposed an extension of a segmentation based method to large scale data set but it does not guarantee watertightness which is an important property for many applications. Moreover, [16] only deals with scenes in which a dimension is highly dominated by the two others (e.g. a city scene enables to "partition the scene on the ground plane" as said in the paper). Conversely, our end-to-end distributed out-of-core algorithm guarantees watertightness and it is suited to any kind of large-scale point cloud with arbitrarily complex 3D geometry (overhangs...).

We proceed by first tiling the point cloud with a distributed octree approach which consists in choosing the maximum depth of an octree decomposition and then merging cells for which the total number of points they contain does not exceed a given value (Figure 2a). Then, we compute the exact 3D Delaunay triangulation to discretise space using the algorithm introduced in [11]. Based on a distributed triangulation structure with shared cells between tiles (blue triangles on Figure 2b) associated with the adjacency graph of tiles (red lines on Figure 2b), we propose an extension of a distributed graph-cut algorithm [28] (initially introduced for images). The watertight surface can be extracted on the whole scene (blue line on Figure 2c) without any post-processing thanks to the consistent topology ensured by the global Delaunay triangulation.

In the next section, we review existing works on large scale surface reconstruction with a focus on volumetric segmentation and pioneering contributions to large-scale issues in general. Sections 3 and 4 respectively present the reference surface reconstruction algorithm and its distributed version. We then evaluate our pipeline both in terms of accuracy and speedup and the results of this assessment are to be found in Section 5.

2. Related works

2.1. Watertight surface reconstruction by volumetric segmentation

Volumetric segmentation consists in determining whether each region of a space subdivision can be traversed (empty) or not (occupied) by light at the wavelength to which the sensor used is sensible. Most surface reconstruction approaches, including the one on which this paper focuses, follow this principle and we review them here. [21] and [22] label as *occupied* or *empty* each tetrahedron of the Delaunay triangulation of the point samples thanks to a graph-cut optimisation of an energy function defined based on the lines of sight (from the optical center to the vertex) and the shape of the triangles. A watertight triangulated surface is then extracted from this labelling as the set of triangles separating empty and occupied tetraedra. [17] labels voxels as *empty*, *occupied* or *unknown* based on the lines of sight. An interesting feature is that undesirable moving objects such as humans can be erased in the final surface as they cause inconsistent voxel labels when combining multiple scans. [10] uses the same paradigm but relies on the Dempster Shafer theory to provide in addition a per-triangle confidence measure. This work is the starting point for further improvements such as densification of the point cloud in regions of high uncertainty.

2.2. Big Data-oriented methods

As pointed out in [13], **RAM size** is often a bottleneck when it comes to deal with large amounts of data. Indeed, most surface reconstruction algorithms build the surface all at once by loading the whole point cloud and creating additional data structures (Delaunay triangulations, voxel grids, octrees, etc.) to represent the output surface. This implies a strong constraint on the maximum input data size for a given amount of RAM. **Processing time** inevitably increases with the number of points to process, at best linearly. Even without considering real-time applications, it is important to maintain the processing time within reasonable bounds when the data size becomes very large.

Some authors have pioneered Big Data by **introducing new paradigms** bypassing the typical constraints we discussed. As an example, [18] is an algorithm that can be streamed given the locality of the information that is necessary to build the surface. The *ball-pivoting* algorithm [4] was proposed in 1999 and has come as a forerunner in terms

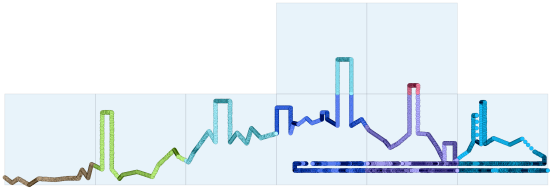
of out-of-core surface reconstruction. It is an incremental interpolating algorithm which basic idea is the following: a ball pivots around an edge of an initial triangle until it touches another point. If no other point is in the ball, the three points form a triangle. The process is repeated for another edge and then from another seed triangle until the whole point cloud has been explored. The interesting property is that only a small neighbourhood is considered. We can therefore process the data incrementally. To achieve this, two axis-aligned planes π_1 and π_2 define the so-called active region of work for pivoting. When all edges within this space have been tested, the two planes are shifted, data that is not going to be used again is dumped and new points are loaded according to the new active region of work.

[25] introduce a *stream-processing* model that enables to process point clouds *out-of-core*. The basic idea is that some operators only need a subset of the whole input data at a given time. Thus, this small number of points is loaded in memory and the function can be performed while the rest of the point cloud stays on *disk*. This has been the theoretical basis for several algorithms including [6] which is still considered as a standard in this domain.

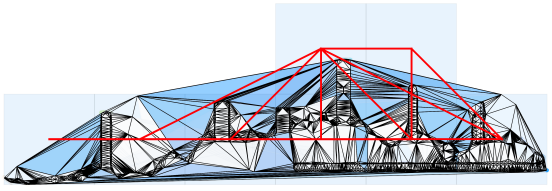
Let us now review **watertight surface reconstruction works addressing distribution and scalability** as proposed in this paper. [2] presents an out-of-core algorithm that enables to interactively process point clouds that do not fit into memory. Their method consists in sub-sampling the initial input point cloud $\mathcal{P}$ to produce a new, smaller one: $\mathcal{P}_{rep}$. The Delaunay triangulation (DT) of $\mathcal{P}_{rep}$ is computed and the *geometric convection* algorithm from [12] allows to reconstruct a simplified version of the surface implied by $\mathcal{P}$. After dividing $\mathcal{P}$ in n regions $\mathcal{P}_{i,\, i=1,\ldots,n}$ of equal size such that: $\bigcup_{i=1,\ldots,n} \mathcal{P}_i = \mathcal{P}$, points of each $\mathcal{P}_i$ are inserted in the triangulation and a surface refinement algorithm processes them in order to update the reconstructed surface.

The methods presented in *Parallel Poisson Surface Reconstruction* [7] and *Streaming MLS* [14] are both out-of-core and parallel. The external memory feature of [7] comes from the fact that it is based on a previous algorithm [6] while being able to run several computations at the same time is the real contribution of this paper. Their approach consists in formulating the surface reconstruction problem as a Poisson equation as introduced in [19]. However, they get rid of the necessity to consider the whole point cloud at once and rather partition space in order to solve the problem locally. Streaming MLS [14] is based on *Moving Least Squares* [23] and has a very straightforward parallel implementation that enables it to be run on a cluster of PCs. Besides, [15] also presented an MLS-based out-of-core algorithm.

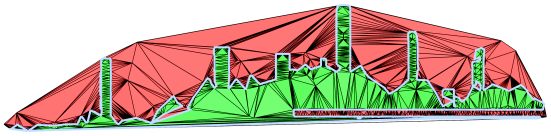
[26] use Non-Uniform Rational B-Spline (NURBS) approximation to compute a surface. The optimisation problem is formulated such that the knot vector T is part of the unknowns. The solution is parallelised using MPI [24] on the *Chemnitzer Linux Cluster* supercomputer [1] and the associated speedup is almost linear.



(a) Tiled input point cloud with multiple acquisitions.



(b) Tiled Delaunay triangulation. The local cells are shown in white and mixed in blue. The adjacency graph is drawn in red.



(c) Surface extraction. The cells in red (resp. green) are labelled as *empty* (resp. *occupied*), the light blue line denotes the extracted surface.

Figure 2: 2D example of the proposed approach.

More recently, [16] proposed a distributed workflow for urban scene computation, based on local Delaunay triangulation and local graph-cut distribution. However, as mentioned in Section 1, their distribution of the optimisation does not guarantee important properties like watertightness along borders.

3. Surface reconstruction model

In this section, we present the *reference* (undistributed) surface reconstruction algorithm [10] of which we present a distributed version in Section 4.

Local PCA: Several steps of the pipeline rely on a local PCA (Principal Components Analysis) of the inertia matrix of a carefully chosen neighbourhood of each point $\mathbf{p} \in \mathcal{P}$. The resulting normalised eigenvectors define a local frame $f = (\mathbf{p}, \vec{v_1}, \vec{v_2}, \vec{v_3})$ and the corresponding eigenvalues (e_1, e_2, e_3) indicate how much the point cloud locally spreads in each direction. The eigenvector with smallest eigenvalue provides a local (unoriented) surface normal direction. The position of the optical centre can then be used to consistently orient the normals from occupied to empty space.

Sub-sampling: As planar areas can be represented with

large triangles, the input point cloud is sub-sampled to reduce the number of points without impairing the geometric precision with an adaptive algorithm based on PCA. Points $\mathbf{p}$ from the original point cloud $\mathcal{P}$ are randomly sampled and added to the set of sub-sampled points $\mathcal{P}^S$ only if $\forall j \in \{1,2,3\},\ p_j = |(\mathbf{p}-\mathbf{q}).\vec{v_j}| > \sigma e_j$ where p_j are the coordinates of $\mathbf{p}$ in the local frame of its closest point $\mathbf{q} \in \mathcal{P}^S$. The parameter σ controls the density of the resulting point cloud: as it increases, the distance threshold becomes larger so less points are added to $\mathcal{P}^S$ which is less dense.

Delaunay Triangulation (DT): Space is discretised with the DT [5] of $\mathcal{P}^S$. In 3D, cells of this DT are tetrahedra which circumspheres do not contain any point of $\mathcal{P}^S$ other than the 4 points lying on them and which union exactly covers the convex hull of $\mathcal{P}^S$.

Mass computation: Following [10], the empty/occupied/unknown state at any point $\mathbf{y} \in \mathbb{R}^3$ is defined for each input line of sight associated with point $\mathbf{p}$ using Dempster Shafer Theory (DST) masses $\mathbf{m_p}(\mathbf{y}) = (e,o,u) \in \left[0,1\right]^3$ such that $e+o+u=1$. For example, $e=1$ means certainty that space is *empty*, $o=1$ that space is *occupied* and $u=1$ means total *uncertainty* at $\mathbf{y}$. These masses $m_{\mathbf{p}}(\mathbf{y})$ are then combined into an overall mass $m(\mathbf{y}) = (e,o,u)$ and integrated over each tetrahedron t of the DT with a Monte Carlo sampling, yielding a single occupancy value $m_t = o/(1-u)$ that gives the probability of t to be *occupied*.

Volumetric segmentation via graph-cut optimisation: The reconstruction method relies on assigning to each tetrahedron t of the DT the label 0 if t is *empty* or 1 if t is *occupied*. For simplicity we call $\mathcal{T}$ both the DT of $\mathcal{P}^S$ and the set of tetrahedra of this DT. This labelling problem is formulated as a Boolean energy minimisation (1). The energy function is composed of a data term (2) encouraging labels to be in accordance with overall masses and a prior term (3) smoothing the surface, balanced by a parameter α:

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}\in\{0,1\}^{|\mathcal{T}|}}{\arg\min} \Big(E_{data}(\mathbf{x}) + \alpha\, E_{prior}(\mathbf{x}) \Big) \tag{1}$$

where $\mathbf{x} = (x_t)_{t\in\mathcal{T}} \in \{0,1\}^{|\mathcal{T}|}$ is the set of tetrahedra labels and $|\mathcal{T}|$ the number of tetrahedra.

$$E_{data}(\mathbf{x}) = \sum_{t\in\mathcal{T}} V_t.|x_t - m_t| \tag{2}$$

$$E_{prior}(\mathbf{x}) = \sum_{(t,t')\in\mathcal{T}^2} A_{t\cap t'}.|x_t - x_{t'}| \tag{3}$$

where V_t is the volume of tetrahedron t and $A_{t\cap t'}$ is the area of the interface between tetrahedra t and t'.

This energy (1) can be optimised via a graph-cut algorithm. To do so, let us denote by $G=(V,E)$ the dual graph of which the vertices $V = \{s,t\} \cup \mathcal{T}$ are the source s, the sink t and the tetrahedra of $\mathcal{T}$. The edges E are the triangles making the interface between all pairs of adjacent tetrahedra but also those connecting every tetrahedron to both s and t. Each edge is weighted so that the cost of an *s-t* cut in the dual graph has the same value as the energy function (1) after attributing the label values $\mathbf{x}$. To minimise the energy, we then need to minimise the sum of cut edges weights:

$$\sum_{(i,j)\in E} c_{i,j} x_{i,j} \tag{4}$$

where $c_{i,j}$ is the weight of edge (i,j) and $x_{i,j}$ is a Boolean indicating if the edge is cut ($1=$cut, $0=$not cut). Moreover, x_i (the label of tetrahedron i) indicates whether vertex i is linked to s ($x_i=0$) or to t ($x_i=1$) after a given cut. We then have: $x_s=0$, $x_t=1$ and $\forall (i,j)\in E,\ x_{i,j} = |x_i - x_j|$.

Surface extraction: The final surface is defined as the set of triangles separating occupied and empty tetrahedra.

4. Distributed surface reconstruction

In this section, we explain the distribution strategy. Algorithm 1 sums up in detail the different steps and Figure 3 presents the complete distributed workflow.

4.1. Algorithm distribution

Tiling: Given a point set $\mathcal{P}$, we note the partition $\mathcal{P}_K = (\mathcal{P}_k)_{k\in K}$ its decomposition into $|K|$ disjoint subsets $\mathcal{P}_k$, where K denotes a discrete set of tile indices (Figure 2a, Algorithm 1 line 2). We can define the **primary** tile of a point $\mathbf{p} \in \mathcal{P}$ as the unique tile $\mathcal{P}_k$ such that $\mathbf{p} \in \mathcal{P}_k$.

Local PCA: The PCA is distributed by computing it separately on each tile $\mathcal{P}_k$. The tile along with the PCA **I**nformation is denoted $\mathcal{P}_k^I$.

Sub-sampling: Each tile $\mathcal{P}_k$ is sub-sampled separately yielding **S**ub-sampled tiles $\mathcal{P}_k^S$.

Distributed Delaunay triangulation: We use the algorithm proposed in [11] to compute a distributed DT of the union of the sub-sampled point clouds $\mathcal{P}_K^S = \bigcup_{k\in K} \mathcal{P}_k^S$ (Algorithm 1 line 6). The **distributed DT** of a tiled point set $\mathcal{P} = \bigcup_{k\in K} \mathcal{P}_k$ is defined in [11] as a set of DTs: $\mathcal{T}_K = (\mathcal{T}_k)_{k\in K}$ such that $\mathcal{P}_k$ is a subset of the vertices of $\mathcal{T}_k$. The vertices of $\mathcal{T}_k$ are said to be **local** in $\mathcal{T}_k$ if they belong to $\mathcal{P}_k$ and **foreign** in $\mathcal{T}_k$ if they belong to another tile $\mathcal{P}_l$ with $l \neq k$. By extension, cells and facets of $\mathcal{T}_k$ are **local** (resp. **foreign**) in $\mathcal{T}_k$ if all their vertices are local (resp. foreign) in $\mathcal{T}_k$, and **mixed** otherwise. For a mixed cell t, we call K_t the set of indices of tiles containing t. We call a mixed cell *main* in $\mathcal{T}_k$ if $\min(K_t) = k$. The algorithm of [11] guarantees that the union of all local and main mixed cells is exactly the DT of $\mathcal{P}$ and that these cells are disjoint. We define the adjacency graph of the triangulation $G^A = (\mathcal{T}^A, \mathcal{E}^A)$ where $\mathcal{T}^A$ are the tile-triangulations and

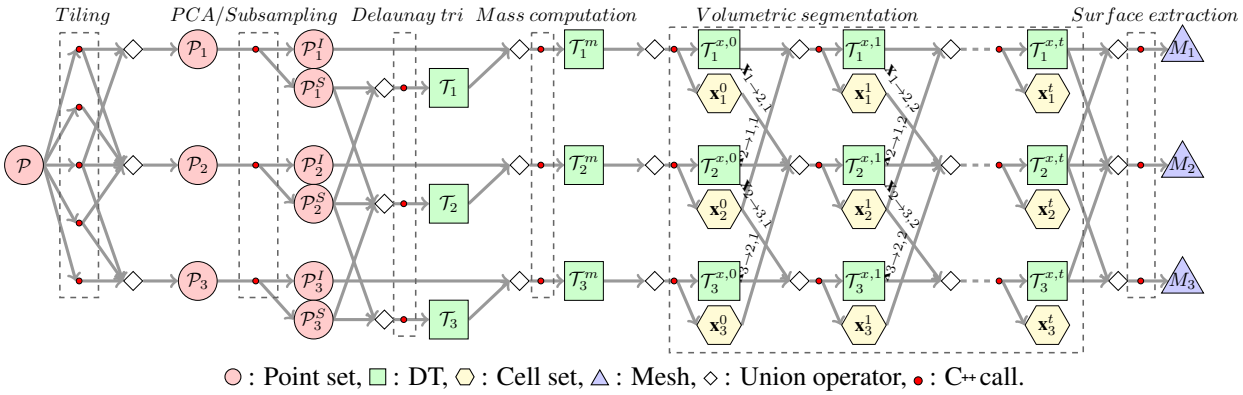


Figure 3: The proposed distributed surface reconstruction workflow. $\mathcal{P}$ denotes the input point set, $\mathcal{P}_k$ the tiled point set, $\mathcal{P}_k^I$ and $\mathcal{P}_k^S$ are the tiled point set with PCA **i**nformation and the **s**implified one, $\mathcal{T}_k$ the tile-triangulations, $\mathcal{T}_k^m$ the tile-triangulation with the mass score, $\mathcal{T}_k^{x,t}$ the labelled tile-triangulation at iteration t, $\mathbf{x}_k$ the updated **mixed** cells during the optimisation and M_k the final mesh.

Algorithm 1: Distributed surface reconstruction

Input: Point set $\mathcal{P}$

// Tiling

1 **for** $\mathbf{p} \in \mathcal{P}$ **do** in parallel

2 $\quad k \leftarrow TileId(\mathbf{p})$

3 $\quad \mathcal{P}_k \leftarrow \mathcal{P}_k \bigcup \{\mathbf{p}\}$

// PCA/Sub-sampling

4 **for** $k \in K$ **do** in parallel

5 $\quad \mathcal{P}^I{}_k, \mathcal{P}_k^S \leftarrow PCA(\mathcal{P}_k)$

// Delaunay Triangulation (DT)

6 $\mathcal{T}^A, \mathcal{E}^A = DDT(\mathcal{P}^S)$

// Mass computation

7 **for** $k \in K$ **do** in parallel

8 $\quad \mathcal{T}_k^m \leftarrow mass\_computation(\mathcal{T}_k^A \cup \mathcal{P}_k^I)$

// Volumetric segmentation

9 $t \leftarrow 0$

10 **for** $k \in K$ **do** in parallel

11 $\quad \mathcal{T}_{k,t}^x, \mathbf{x}_{k \to l \neq k,t} \leftarrow graph\_cut(\mathcal{T}_k^m)$

12 $t \leftarrow 1$

13 **while** $t < max\_iterations$ **do**

14 $\quad$ **for** $k \in K$ **do** in parallel

15 $\quad\quad \boldsymbol{\lambda}_t^k \leftarrow update\_lagrange\left(\bigcup_l \mathbf{x}_{l \to k,t-1}\right)$

16 $\quad\quad \mathcal{T}_{k,t}^x, \mathbf{x}_{k \to l \neq k,t} \leftarrow graph\_cut(\mathcal{T}_{k,t-1}^x, \boldsymbol{\lambda}_t^k)$

17 $\quad t \leftarrow t + 1$

// Surface extraction

18 **for** $k \in K$ **do** in parallel

19 $\quad \mathcal{T}_k^{'} \leftarrow aggregate\_neigbors(\mathcal{T}_{k,t}, \mathcal{E}^A)$

20 $\quad M_k = extract\_surface(\mathcal{T}_k^{'})$

21 **return** M

$\mathcal{E}^A$ the edges between them. Two triangulations are connected by an edge if and only if these triangulations share mixed points (see Figure 2b).

Mass computation: Given that the effect of an input point $\mathbf{p}$ on a tetrahedron t decreases with the distance, the mass function is only computed locally on each tile-triangulation $\mathcal{T}_k$ using $\mathcal{P}_k^I$ (Algorithm 1 line 8).

Volumetric segmentation: The energy function 1 is optimised in parallel with independent graph-cuts on each tile (Algorithm 1 line 9) where the labels of mixed cells are encouraged to agree with an iterative scheme as explained in Section 4.2.

Surface extraction: To extract the full surface, each tile is loaded with its respective neighbours regarding the adjacency graph's edges E (Algorithm 1 line 18). We denote by $aggregate\_neigbors(\mathcal{T}_k^A, \mathcal{E}^A)$ the function that aggregates the triangulation and its respective neighbours. Because the iterative scheme does not provide guarantee to converge on the same label on **mixed** cells (for which copies exist in multiple tile-triangulations), the label from the main mixed cell is systematically chosen for the surface extraction. Note that the surface is watertight even if the labels do not agree since the optimisation ends with a unique set of labels which inevitably leads to a watertight mesh.

4.2. Distributed graph-cut

The main challenges to distribute a DT based surface reconstruction method is distributing the DT itself, for which we use the implementation of [11], and distributing the graph cut. Graph cuts distribution approaches are highly dependent on the structure of the graph. While many approaches exist to distribute graphs that come from the regular 2D structure of image segmentation [29], we did not find an approach tackling the specific structure of the adjacency

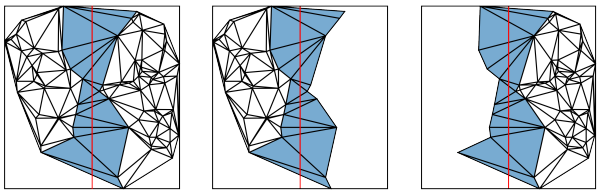


Figure 4: 2D visualisation of the full triangulation (left) being split into 2 tiles (middle and right) with local cells in white and mixed cells in blue. Mixed cells are duplicated in both sub-graphs.

graph G of the cells of a DT (see Section 3) and of its tiling-induced split into $|K|$ sub-graphs $G_k = (V_k, E_k)$. Thus one of the major contributions of this paper is the generalisation to our tiled graph G of the method proposed in [28] to distribute the graph-cut optimisation across our multiple tiles. We use the structure of the tiled DT to split G into $|K|$ subgraphs G_k, one for each tiled-triangulation T_k. In practice, each G_k is defined as the adjacency graph of the set of local and mixed cells of T_k (Figure 4). The main issue of the distributed problem is to ensure that the mixed cells have the same label relative to the tiles that share them. For any $k \in |K|$, we respectively denote by $x^k_{i,j}$ and x^k_i the values of $x_{i,j}$ and x_i in sub-graph k. Besides, we define $c^k_{i,j} = \frac{c_{i,j}}{|\{k:(i,j)\in E_k\}|}$: the weight of edge (i,j) normalised by the number of tiles it appears in. Using this decomposition, the optimised function becomes:

$$f(\mathbf{x}) = \sum_{(i,j)\in E} c_{i,j}x_{i,j} = \sum_{k\in K}\sum_{(i,j)\in E_k} c^k_{i,j}x^k_{i,j} \quad (5)$$

under the condition that labels x^k_i should have the same value amongst all subgraphs they appear into:

$$\forall\, k,l \in K^2, \quad \forall\, i \in V_k \cap V_l, \quad x^k_i - x^l_i = 0 \quad (6)$$

To split this problem in sub-problems using our tiled graph structure, we solve the graph-cut on each sub-graph G_k with variables $x^k_{i,j}$. To guarantee the condition expressed in equation 6, we add a penalty term to the energy using Lagrangian duality:

$$L(\mathbf{x},\boldsymbol{\lambda}) = \sum_{k\in K}\sum_{(i,j)\in E_k} c^k_{i,j}x^k_{i,j} + \sum_{k\in K}\sum_{l>k}\sum_{i\in V_k\cap V_l} \lambda^{k,l}_i(x^k_i - x^l_i) \quad (7)$$

with $\boldsymbol{\lambda} = \{\lambda^{k,l}_i : (k,l) \in K^2,\ i \in V_k \cap V_l\}$. By defining, for any tile index $k \in K$:

$$L_k(\mathbf{x}^k,\boldsymbol{\lambda}) = \sum_{(i,j)\in E_k} c^k_{i,j}x^k_{i,j} + \sum_{l\neq k}\sum_{i\in V_k\cap V_l} \left\{ \begin{array}{ll} 1 & \text{if } k<l \\ -1 & \text{otherwise} \end{array} \right\} \lambda^{k,l}_i x^k_i \quad (8)$$

We then have:

$$L(\mathbf{x},\boldsymbol{\lambda}) = \sum_{k\in K} L_k(\mathbf{x}^k,\boldsymbol{\lambda}) \quad (9)$$

Since the Lagrange dual function $g(\boldsymbol{\lambda}) = \min_{\mathbf{x}} L(\mathbf{x},\boldsymbol{\lambda})$ is concave [9], we can find an optimal distributed solution to our problem (5+6) by maximising $g(\boldsymbol{\lambda})$ via an ascent method. We denote by t the iteration index and by $\boldsymbol{\tau} = \{\tau^{k,l}_i : (k,l) \in K^2,\ i \in V_k \cap V_l\}$ the vector of step amplitudes by which we increment $\boldsymbol{\lambda}$. We initialise the Lagrange multiplier vector $\boldsymbol{\lambda}(t=0) = \mathbf{0}$ and the vector of steps $\boldsymbol{\tau}(t=0)$ by $\tau^{k,l}_i = \tau_0$. As demonstrated in [28], $(\mathbf{x}^k - \mathbf{x}^l)_{k,l\in K^2}$ is a supergradient to g at $\boldsymbol{\lambda}$. Consequently, we can solve the distributed graph cut by iterating:

- Solve the graph-cut problems $L_k(\mathbf{x}^k,\boldsymbol{\lambda})$ for $\mathbf{x}^k$
- Update $\boldsymbol{\lambda}$ and $\boldsymbol{\tau}$:

  for $k \in K$ and $l < k$:

$$\lambda^{k,l}_{i,t+1} = \lambda^{k,l}_{i,t} + \tau^{k,l}_{i,t}(x^k_{i,t} - x^l_{i,t}) \quad (10)$$

$$\tau^{k,l}_{i,t+1} = \frac{\tau^{k,l}_t}{2} \text{ if } x^k_{i,t} - x^l_{i,t} \neq x^k_{i,t-1} - x^l_{i,t-1} \quad (11)$$

To perform this update at iteration step t, each tile G_l sends to its neighbour tiles G_k the labels of their shared mixed cells (according to G_l) in a vector $\mathbf{x}_{l\to k,t}$.

4.3. Implementation details

The workflow is implemented with the Apache Spark framework [31]. The input of the algorithm is a point cloud separated in several files stored on the Hadoop distributed filesystem (HDFS) [27]. To ensure a reasonable memory footprint in the tiling, we split large files to ensure that each input file contains less than 1 million points. Each such file is serialised into an RDD [30] with a *Base64* encoding in a *String*. The key/value formalism is used: each element of the RDD is represented by a key and a value which is a list of sets (list of point sets, list of tile-triangulations, etc.). A transformation on an RDD (Local graph-cut, mass computation, etc.) is performed with a C++ call in parallel on each RDD chunk by using the *pipe* operator. The union operator ($\cup$ in Algorithm 1 and $\diamond$ in Figure 3) is a union followed by a *ReduceByKey* in Spark.

5. Experimental results

While memory and computational efficiency are the main goals when trying to distribute an algorithm, the most important aspect of evaluation is to verify that the *distributed* algorithm reproduces as closely as possible the output of the *reference* algorithm. For that reason, we will first evaluate the *distributed* algorithm in comparison with

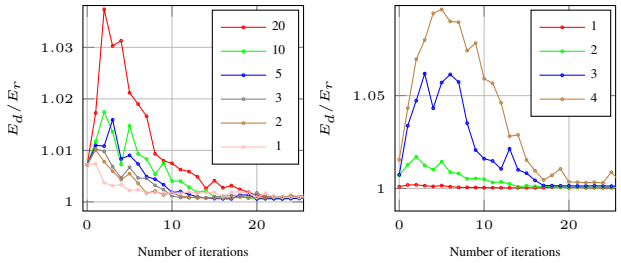


Figure 5: Ratio between the energy of the labelling given by the distributed graph-cut E_d and the reference graph-cut with one thread E_r for different initialisations of the **Lagrangian step** $\tau_0 = 1, 2, 3, 5, 10, 20$ (left) and different **octree depths** $1, 2, 3, 4$ (right).

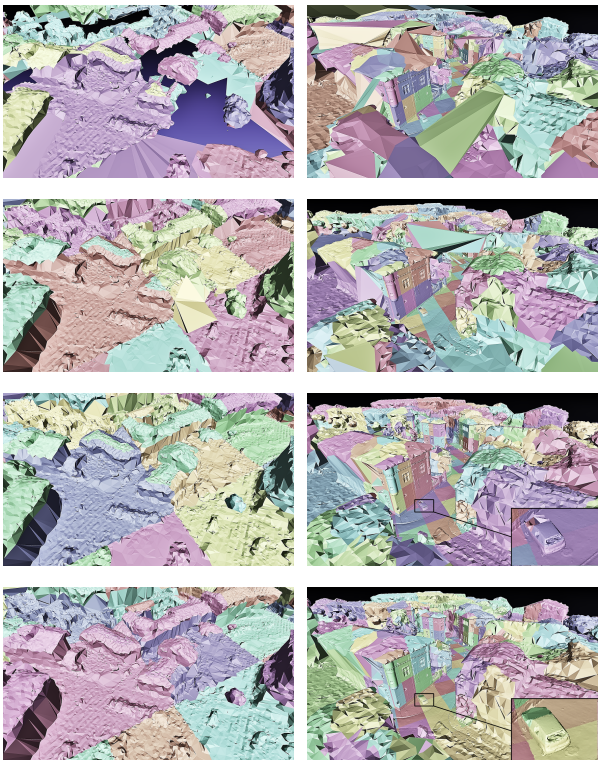


Figure 6: Evolution of the resulting mesh along iterations on a real case data. First line shows the result with the local graph-cut without iterating, second line at iteration 3, third line at iteration 15 and fourth line at iteration 30.

the *reference*, and then focus on the scalability. Moreover, as PCA and sub-sampling are expected to suffer minor border effects, and the distributed Delaunay triangulation is proved to produce the exact Delaunay Triangulation [11], we mainly focus our evaluation on our major contribution: the distributed graph cut. In all the following

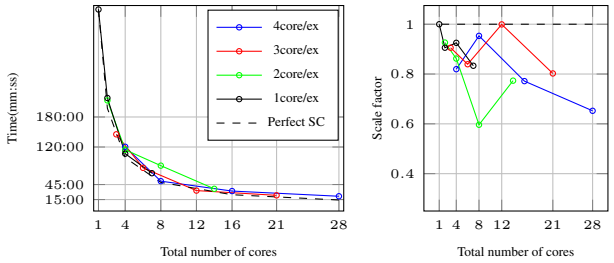


Figure 7: Execution time (left) and strong scaling factor (right) according to the number of cores with 4,3,2 and 1 cores/executors (12Go Ram/executors) on a 10 million points data set.

experiments, unless otherwise stated, we used the values: $\alpha = 0.005$, $\tau_0 = 5$, and 30 iterations.

5.1. Distributed graph-cut convergence

Graph cut implementations provably find the global minimum cut of the graph [8] which equivalently achieves the global minimum of the associated energy (eq 1). Thus we will assess our distributed graph cut implementation by comparing the energy obtained with our distributed implementation with this global minimum energy. In order to evaluate our capacity to handle a large proportion of mixed cells (which makes the distributed problem harder), the algorithm is run with different octree depths for a fixed number of points. A larger depth induces more and smaller tiles, resulting in a higher proportion of mixed cells. As our iterative scheme is mainly influenced by the value of τ_0, we also evaluate its impact on convergence. Figure 5 shows the evolution along iterations t of the ratio between the energy (eq 1) of the labelling computed with the ***d**istributed* graph-cut E_d and the labelling computed with the ***r**eference* graph-cut E_r. It is important to note that even if two surfaces with the same energy are not necessarily the same, they are considered equally good. For all values of τ_0 and octree depths, the energy ratio increases during the first iterations until it reaches a maximum around iteration 5, then decreases until converging significantly under the initialisation value,which shows the robustness of our distributed algorithm to these two factors.

Figure 6 shows the resulting mesh along iterations on a LiDAR data set. Shared cells are visually the most impacted after the first graph-cut. In some areas, tiles are not or badly connected, only triangles between local cells at the centre of a tile are well reconstructed. After 3 iterations, the mesh is better regularised but the quality remains bad at the boundary. After 15 iterations, the main errors are removed and the tiles are globally well connected. At iteration 30, small shared areas like the car hood in the example are well reconstructed.

Figure 8: Reconstruction result on a 350 million points data set. Tiling is visualised through a random color per tile.

5.2. Speedup

The efficiency of the proposed approach is evaluated on a Spark cluster with 28 cores and 100GB of RAM. This configuration is close to the *m5*[1] setup of the Amazon EMR service dedicated to general usage that provides 4GB per core. We evaluate the **strong scaling** speedup which quantifies how the algorithm scales with an increasing number of cores for a fixed number of points. We compute the surface of a 10 million points data set with different cluster configurations: a varying number of executors (7,4,2 and 1) and, for each executor, a varying number of cores (4,3,2,1) for a fixed 12GB of RAM per executor in every case. The total number of cores is the number of executors times the number of cores per executor. The result is shown on Figure 7 where the strong scaling factor is defined by $t_{1\text{ core}}/(n.t_{n\text{ cores}})$ as a function of the number of cores. As our application is fully distributed, we compare it to the perfect scale factor of 1. Results show that every configuration has a scale factor > 0.6. In other words, the execution time with a single thread is at least divided by 0.6 times the number of cores with the distributed implementation.

5.3. Large data set result

We tested our algorithm on a very large data set that combines more than 350 million points acquired with both aerial and terrestrial LiDAR (Figure 8), on which our available hardware could not run the reference implementation by lack of RAM. The point cloud was split into 9520 tiles and it took 17 hours to process using 28 cores. The proposed approach produces a watertight mesh of the whole area with highly detailed and complex 3D structures, composed of around 80M vertices. We empirically set the subsampling parameter $\sigma = 0.3$ to have a good trade-off between accuracy and computing time (a higher σ would reduce computing time at the cost of accuracy).

[1] https://aws.amazon.com/ec2/instance-types/m5/

6. Conclusion and perspectives

In this paper, we have presented a fully distributed and out-of-core surface reconstruction method. The extension of the distributed graph-cut method of [28] to the structure of the tiled 3D Delaunay Triangulation of [11] leads to a fully distributed method that ensures both watertightness of the resulting mesh and preservation of sharp edges. The evaluation confirms that the proposed method produces meshes of quality similar to the reference (non-distributed) implementation. The fully distributed framework allows an efficient speedup while maintaining a reasonable memory footprint even for very large inputs. The implementation on the state-of-the-art Apache-Spark framework shows very good results on managing very large data sets. One of the drawbacks of our approach is that it is still slow compared to a classic optimised algorithm on *small* data sets that fits in memory. Thus the next step is to further optimise the algorithm in order to get processing times comparable to non distributed algorithms on small data set without neglecting the quality. Moreover, we want to improve the quality of the surface reconstruction method by benchmarking recent state of the art DT based surface reconstruction methods (including learning based) that could be easily distributed using our framework as only the mass computation part would change.

References

[1] Chemnitzer Linux Cluster (CLiC). Archive Location: Worldwide Last Modified: 2020-06-02 Library Catalog: www.tu-chemnitz.de. 3

[2] Remi Allegre, Raphaelle Chaine, and Samir Akkouche. A Dynamic Surface Reconstruction Framework for Large Unstructured Point Sets. page 11, Jan. 2006. 3

[3] Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Gaël Guennebaud, Joshua A. Levine, Andrei Sharf, and Claudio T. Silva. A Survey of Surface Reconstruction from Point Clouds. *Comput. Graph. Forum*, 36(1):301–329, Jan. 2017. 1

[4] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, Oct. 1999. 2

[5] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic geometry*. Cambridge university press, 1998. 4

[6] Matthew Bolitho, Michael Kazhdan, Randal Burns, and Hugues Hoppe. Multilevel Streaming for Out-of-core Surface Reconstruction. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, pages 69–78, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. event-place: Barcelona, Spain. 3

[7] Matthew Bolitho, Michael Kazhdan, Randal Burns, and Hugues Hoppe. Parallel Poisson Surface Reconstruction. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Yoshinori Kuno, Junxian Wang, Jun-Xuan Wang, Junxian Wang, Renato Pajarola, Peter Lindstrom, André Hinkenjann, Miguel L. Encarnação, Cláudio T. Silva, and Daniel Coming, editors, *Advances in Visual Computing*, Lecture Notes in Computer Science, pages 678–689. Springer Berlin Heidelberg, 2009. 2, 3

[8] Endre Boros and Peter L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1):155–225, 2002. 7

[9] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 6

[10] Laurent Caraffa, Mathieu Brédif, and Bruno Vallet. 3D Watertight Mesh Generation with Uncertainties from Ubiquitous Data. In Shang-Hong Lai, Vincent Lepetit, Ko Nishino, and Yoichi Sato, editors, *Computer Vision – ACCV 2016*, Lecture Notes in Computer Science, pages 377–391. Springer International Publishing, 2016. 2, 3, 4

[11] Laurent Caraffa, Pooran Memari, Murat Yirci, and Mathieu Brédif. Tile merge: Distributed delaunay triangulations for cloud computing. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1613–1618, 2019. 2, 4, 5, 7, 8

[12] Raphaëlle Chaine. A geometric convection approach of 3-D reconstruction. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '03, pages 218–229, Aachen, Germany, June 2003. Eurographics Association. 3

[13] 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, Oct. 2003. 2

[14] Gianmauro Cuccuru, Enrico Gobbetti, Fabio Marton, Renato Pajarola, and Ruggero Pintus. Fast low-memory streaming MLS reconstruction of point-sampled surfaces. In *Proceedings - Graphics Interface*, pages 15–22, 2009. 3

[15] Valentino Fiorin, Paolo Cignoni, and Roberto Scopigno. Out-of-core MLS Reconstruction. In *Proceedings of the Ninth IASTED International Conference on Computer Graphics and Imaging*, CGIM '07, pages 27–34, Anaheim, CA, USA, 2007. ACTA Press. event-place: Innsbruck, Austria. 3

[16] Jiali Han and Shuhan Shen. Distributed surface reconstruction from point cloud for city-scale scenes. In *2019 International Conference on 3D Vision (3DV)*, pages 338–347. IEEE, 2019. 2, 3

[17] Claude Holenstein, Robert Zlot, and Michael Bosse. Watertight surface reconstruction of caves from 3D laser data. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3830–3837. IEEE, 2011. 2

[18] H. Hoppe, T. Derose, T. Duchamp, J. Mcdonald, and Werner Stuetzle. Surface reconstruction from unorganized points. SIGGRAPH Computer Graphics, 26(2), 71-78. *Computer Graphics (Proc. SIGGRAPH 86)*, 20, Jan. 1996. 2

[19] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 61–70. Eurographics Association, 2006. 3

[20] A. Khatamian and Hamid Arabnia. Survey on 3D Surface Reconstruction. *Journal of Information Processing Systems*, 12:338–357, Jan. 2016. 1

[21] Ravikrishna Kolluri, Jonathan Richard Shewchuk, and James F. O'Brien. Spectral Surface Reconstruction from Noisy Point Clouds. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, pages 11–21, New York, NY, USA, 2004. ACM. event-place: Nice, France. 2

[22] Patrick Labatut, J-P Pons, and Renaud Keriven. Robust and efficient surface reconstruction from range data. In *Computer graphics forum*, volume 28, pages 2275–2290. Wiley Online Library, 2009. Issue: 8. 2

[23] Peter Lancaster. Moving Weighted Least-Squares Methods. In Badri N. Sahney, editor, *Polynomial and Spline Approximation: Theory and Applications*, NATO Advanced Study Institutes Series, pages 103–120. Springer Netherlands, Dordrecht, 1979. 3

[24] Peter Pacheco. *Parallel programming with MPI*. Morgan Kaufmann, 1997. 3

[25] R. Pajarola. Stream-processing points. In *VIS 05. IEEE Visualization, 2005.*, pages 239–246, Oct. 2005. ISSN: null. 3

[26] Maharavo Randrianarivony and Guido Brunnett. Parallel Implementation of Surface Reconstruction From Noisy Samples. Sept. 2002. 3

[27] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society. 6

[28] Petter Strandmark and Fredrik Kahl. Parallel and distributed graph cuts by dual decomposition. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2085–2092, June 2010. ISSN: 1063-6919. 2, 6, 8

[29] Petter Strandmark, Fredrik Kahl, and Thomas Schoenemann. Parallel and distributed vision algorithms using dual decomposition. *Computer Vision and Image Understanding*, 115(12):1721–1732, Dec. 2011. 5

[30] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. pages 15–28, 2012. 6

[31] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association. event-place: Boston, MA. 6