Tile & merge: Distributed Delaunay triangulations for cloud computing

Journée d'étude Big Data
et données spatialisées

Mathieu Brédif
Laurent Caraffa

Murat Yirci, Pooran Memari,
Yanis Marchand, Bruno Vallet

LASTIG
UNIV GUSTAVE EIFFEL
IGN-ENSG

29 juin 2023

1

Watertight Surface Reconstruction

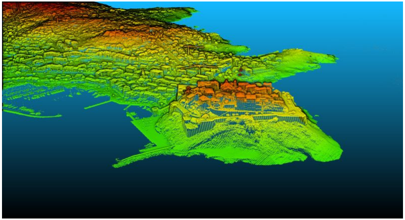
Watertight Surface Reconstruction

- Geospatial point clouds :
  - Aerial LiDAR
  - Mobile mapping LiDAR
  - Photogrammetry
- Desired output
  - Watertight surface
- Example application :
  - Flooding Simulation

Watertight Surface Reconstruction

- Geospatial point clouds :
  - Aerial LiDAR
  - Mobile mapping LiDAR
  - Photogrammetry
- Desired output
  - Watertight surface
- Example application :
  - Flooding Simulation

- Geospatial point clouds :
  - Aerial LiDAR
  - Mobile mapping LiDAR
  - Photogrammetry
- Desired output
  - Watertight surface
- Example application :
  - Flooding Simulation

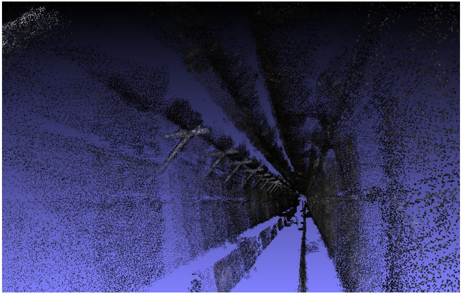


LIDAR HD = High resolution scanning (>30 points/m^2) on the whole French territory : https://geoservices.ign.fr/lidarhd

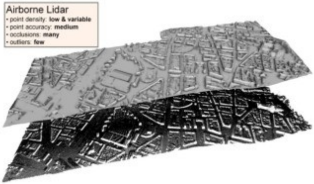
Watertight Surface Reconstruction

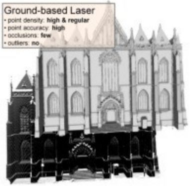
- Geospatial point clouds :
  - Aerial LiDAR
  - Mobile mapping LiDAR
  - Photogrammetry
- Desired output
  - Watertight surface
- Example application :
  - Flooding Simulation

- Geospatial point clouds :
  - Aerial LiDAR
  - Mobile mapping LiDAR
  - Photogrammetry
- Desired output
  - Watertight surface
- Example application :
  - Flooding Simulation



- Geospatial point clouds :
  - Aerial LiDAR
  - Mobile mapping LiDAR
  - Photogrammetry
- Desired output
  - Watertight surface
- Example application :
  - Flooding Simulation

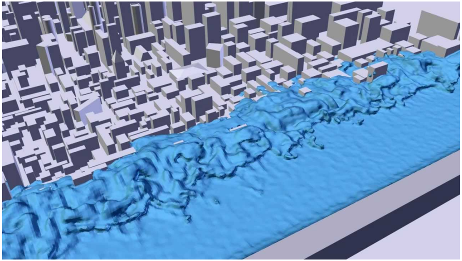




Watertight surface: boundary surface between outside volumes (air) and inside volumes (objects)

Watertight Surface Reconstruction

- Geospatial point clouds :
  - Aerial LiDAR
  - Mobile mapping LiDAR
  - Photogrammetry
- Desired output
  - Watertight surface
- Example application :
  - Flooding Simulation



3D Segmentation

- Binary volume segmentation (Inside / outside)

⇒ Surface is extracted at the interface between inside cells and outside cells

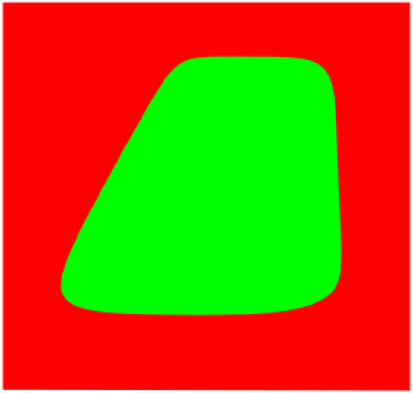
- Volume partitioning
- Graph-cut Formulation
- Graph-cut Optimization
- Surface Extraction

3D Segmentation

- Binary volume segmentation (Inside / outside)

⇒ Surface is extracted at the interface between inside cells and outside cells

- Volume partitioning
- Graph-cut Formulation
- Graph-cut Optimization
- Surface Extraction

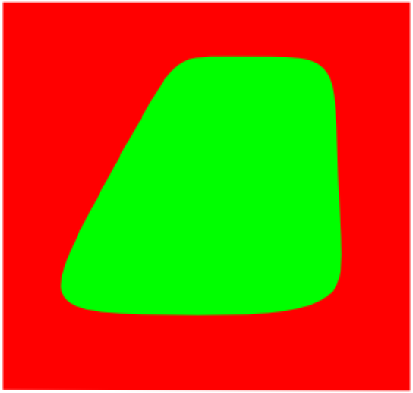


Watertight Surface Reconstruction : algorithm

3D Segmentation

- Binary volume segmentation (Inside / outside)

⇒ Surface is extracted at the interface between inside cells and outside cells



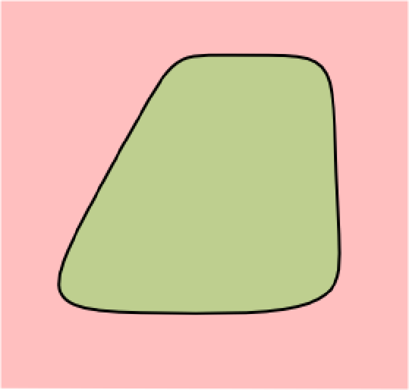
- Volume partitioning
- Graph-cut Formulation
- Graph-cut Optimization
- Surface Extraction

3D Segmentation

- Binary volume segmentation (Inside / outside)

⇒ Surface is extracted at the interface between inside cells and outside cells

- Volume partitioning
- Graph-cut Formulation
- Graph-cut Optimization
- Surface Extraction



3D Segmentation

- Binary volume segmentation (Inside / outside)

⇒ Surface is extracted at the interface between inside cells and outside cells

Algorithm :

- Volume partitioning
- Graph-cut Formulation
- Graph-cut Optimization
- Surface Extraction

Watertight Surface Reconstruction : algorithm

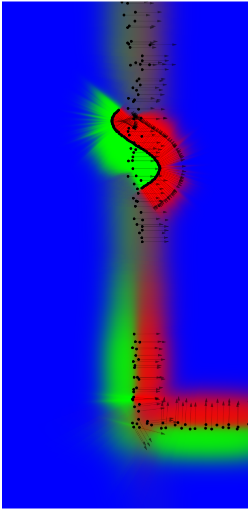
3D Segmentation

- Binary volume segmentation (Inside / outside)

⇒ Surface is extracted at the interface between inside cells and outside cells

Algorithm :

- Volume partitioning
- Graph-cut Formulation
- Graph-cut Optimization
- Surface Extraction



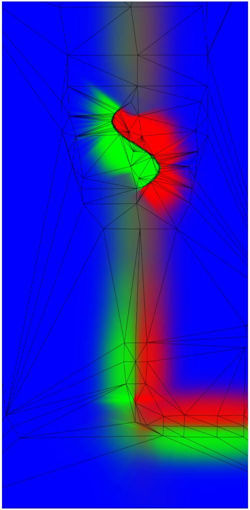
3D Segmentation

- Binary volume segmentation (Inside / outside)

⇒ Surface is extracted at the interface between inside cells and outside cells

Algorithm :

- Volume partitioning
- Graph-cut Formulation
- Graph-cut Optimization
- Surface Extraction



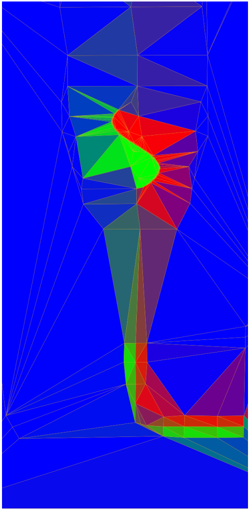
3D Segmentation

- Binary volume segmentation (Inside / outside)

⇒ Surface is extracted at the interface between inside cells and outside cells

Algorithm :

- Volume partitioning
- Graph-cut Formulation
- Graph-cut Optimization
- Surface Extraction



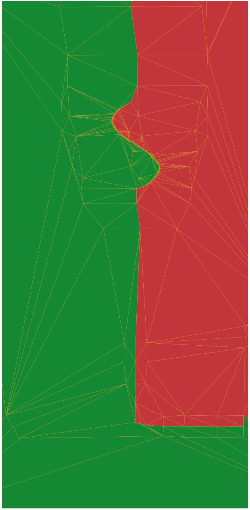
3D Segmentation

- Binary volume segmentation (Inside / outside)

⇒ Surface is extracted at the interface between inside cells and outside cells

Algorithm :

- Volume partitioning
- Graph-cut Formulation
- Graph-cut Optimization
- Surface Extraction



3D Segmentation

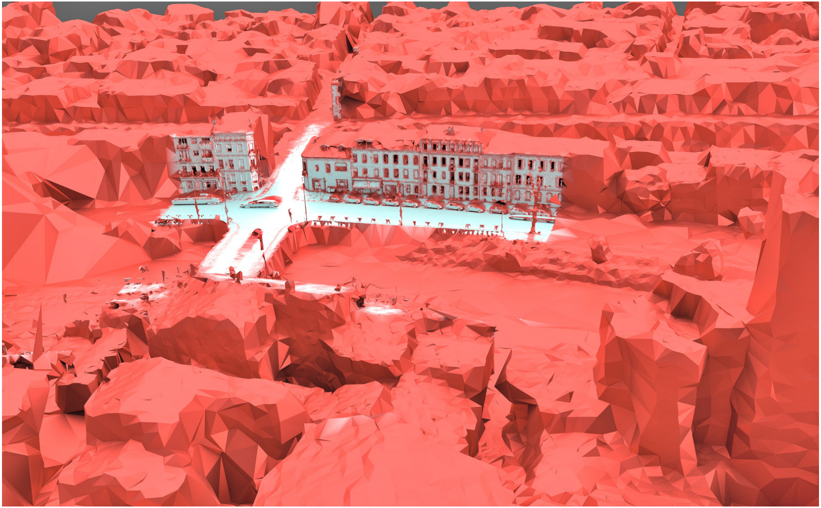
- Binary volume segmentation (Inside / outside)

⇒ Surface is extracted at the interface between inside cells and outside cells

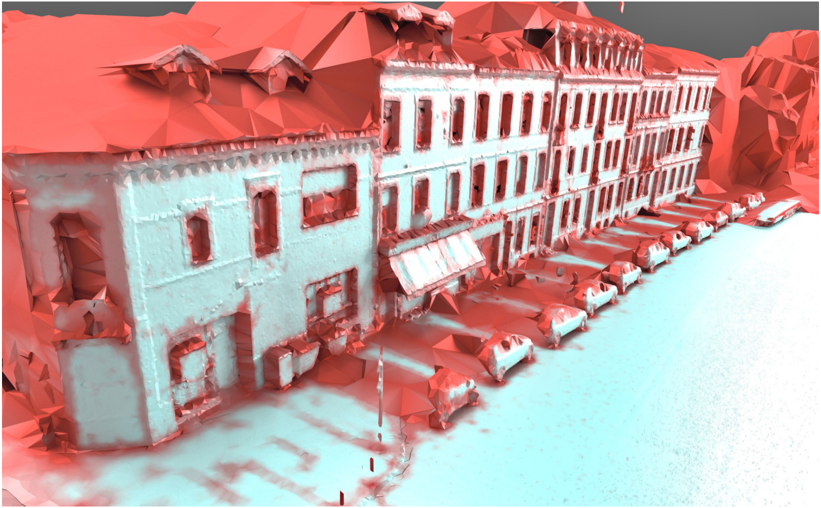
Algorithm :

- Volume partitioning
- Graph-cut Formulation
- Graph-cut Optimization
- Surface Extraction

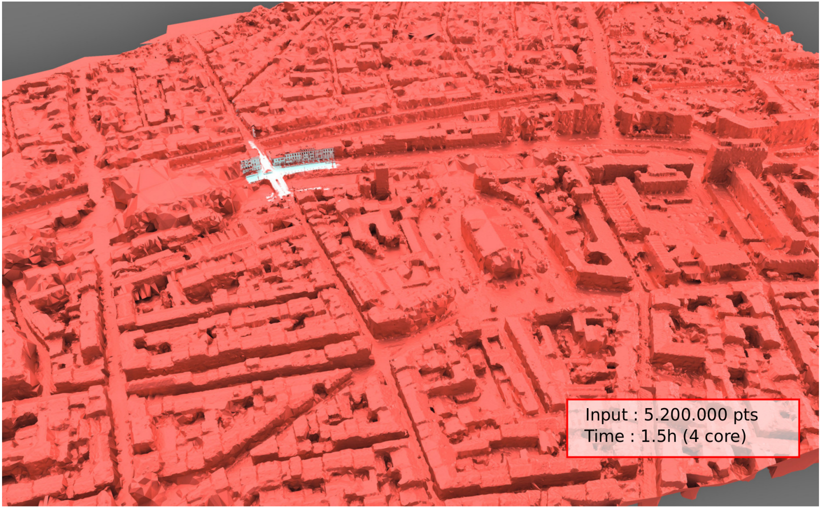
Results : Merging aerial and ground-based point clouds



Results : Merging aerial and ground-based point clouds



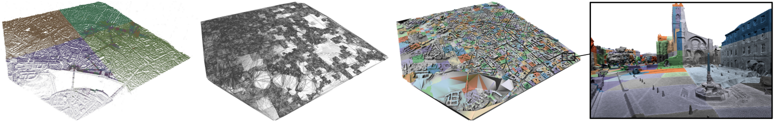
Results : Merging aerial and ground-based point clouds



2 Distributed Watertight Surface Reconstruction

Where are the scalability bottlenecks ?

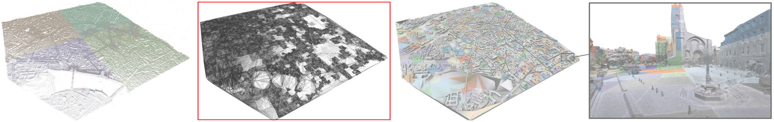
- Delaunay Triangulation Global optimization !
- Graph-cut formulation Embarrassingly Parallel !
- Graph-cut classification Global optimization !
- Surface extraction Embarrassingly Parallel !



4 main steps

Where are the scalability bottlenecks ?

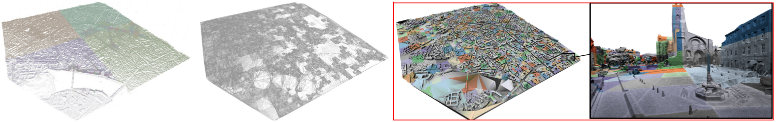
- Delaunay Triangulation Global optimization !
- Graph-cut formulation Embarrassingly Parallel !
- Graph-cut classification Global optimization !
- Surface extraction Embarrassingly Parallel !



4 main steps

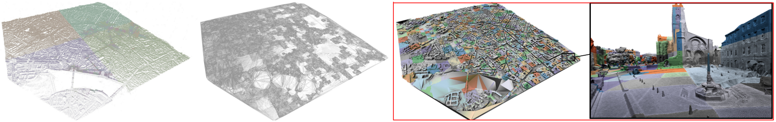
Where are the scalability bottlenecks ?

- Delaunay Triangulation Global optimization !
- Graph-cut formulation Embarrassingly Parallel !
- Graph-cut classification Global optimization !
- Surface extraction Embarrassingly Parallel !



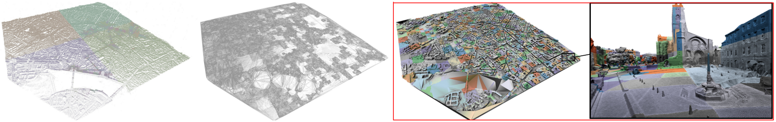
Where are the scalability bottlenecks ?

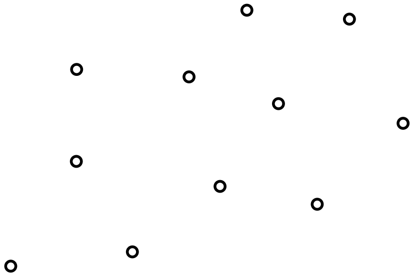
- Delaunay Triangulation Global optimization !
- Graph-cut formulation Embarrassingly Parallel !
- Graph-cut classification Global optimization !
- Surface extraction Embarrassingly Parallel !



Where are the scalability bottlenecks ?

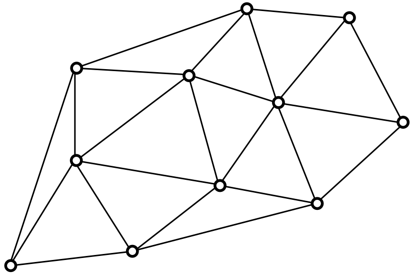
- Delaunay Triangulation Global optimization !
- Graph-cut formulation Embarrassingly Parallel !
- Graph-cut classification Global optimization !
- Surface extraction Embarrassingly Parallel !





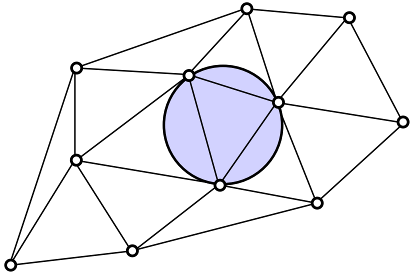
Delaunay condition : empty circumsphere

Delaunay Triangulation (DT)

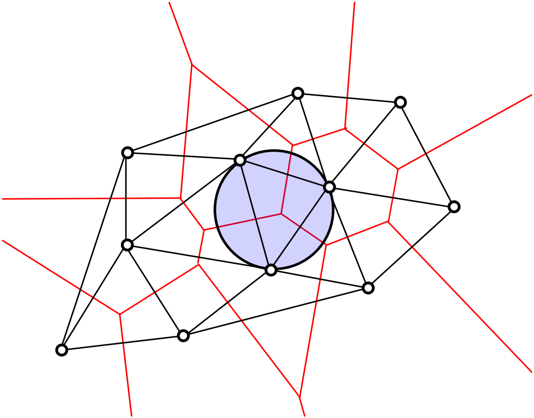


Delaunay condition : empty circumsphere

Delaunay Triangulation (DT)



Delaunay condition : empty circumsphere



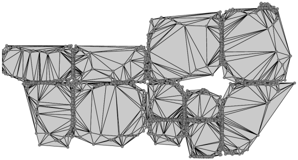
Voronoï diagram : dual of Delaunay triangulation

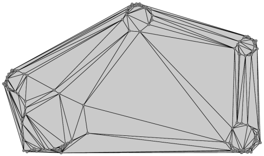
Distributing Delaunay Triangulation

Objectives:

- Scaling to billions or trillions of points using tiling on any computer (from laptop to spark or HPC clusters)
- No hard memory requirements : low memory just takes longer
- Limit communications and synchronizations.

Computing in parallel local DTs (independently within each tile) as an initial triangulation to be repaired to be Delaunay.





Star Splaying

Star Splaying: An Algorithm for Repairing Delaunay Triangulations and Convex Hulls

Jonathan Richard Shewchuk
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, California 94720

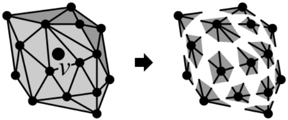
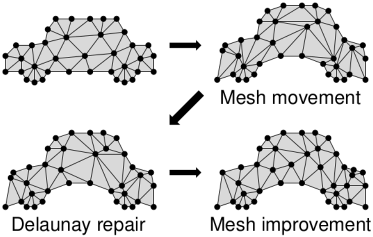


Figure 3. A two-dimensional link triangulation, represented as a collection of two-dimensional stars.



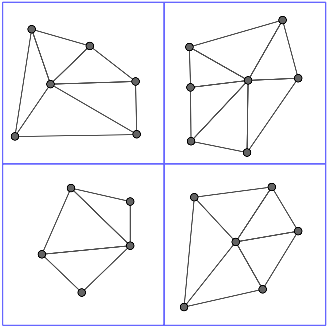
Star Splaying Approach at the tile level (1-rings $\rightarrow$ local DTs)

- Initialize
  - Tile points spatially
  - Compute local DTs
  - Broadcast axis-extreme points
- While points are sent
  - Insert points received by each tile (in batch)
  - Send points with new tile adjacencies
- Local DTs are now local views of the global DT
- Simplify local DTs

Proposal

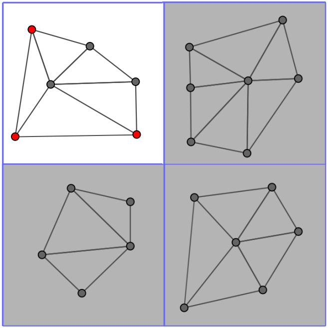
Star Splaying Approach at the tile level (1-rings → local DTs)

- Initialize
  - Tile points spatially
  - Compute local DTs
  - Broadcast axis-extreme points
- While points are sent
  - Insert points received by each tile (in batch)
  - Send points with new tile adjacencies
- Local DTs are now local views of the global DT
- Simplify local DTs



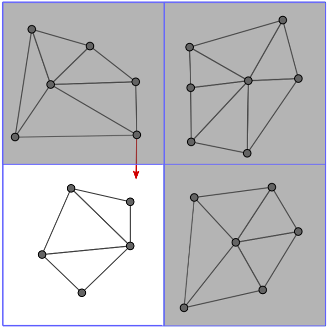
Star Splaying Approach at the tile level (1-rings $\rightarrow$ local DTs)

- Initialize
  - Tile points spatially
  - Compute local DTs
  - Broadcast axis-extreme points
- While points are sent
  - Insert points received by each tile (in batch)
  - Send points with new tile adjacencies
- Local DTs are now local views of the global DT
- Simplify local DTs



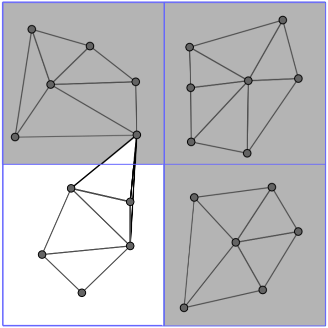
Star Splaying Approach at the tile level (1-rings → local DTs)

- Initialize
  - Tile points spatially
  - Compute local DTs
  - Broadcast axis-extreme points
- While points are sent
  - Insert points received by each tile (in batch)
  - Send points with new tile adjacencies
- Local DTs are now local views of the global DT
- Simplify local DTs



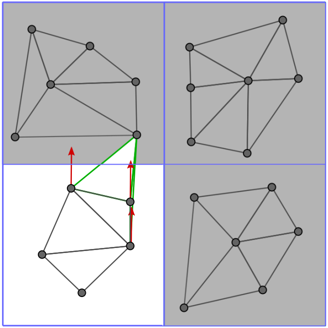
Star Splaying Approach at the tile level (1-rings → local DTs)

- Initialize
    - Tile points spatially
    - Compute local DTs
    - Broadcast axis-extreme points
- While points are sent
    - Insert points received by each tile (in batch)
    - Send points with new tile adjacencies
- Local DTs are now local views of the global DT
- Simplify local DTs



Star Splaying Approach at the tile level (1-rings → local DTs)

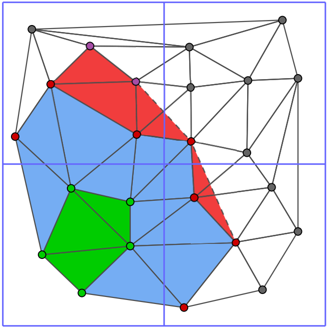
- Initialize
    - Tile points spatially
    - Compute local DTs
    - Broadcast axis-extreme points
- While points are sent
    - Insert points received by each tile (in batch)
    - Send points with new tile adjacencies
- Local DTs are now local views of the global DT
- Simplify local DTs



Proposal

Star Splaying Approach at the tile level (1-rings → local DTs)

- Initialize
  - Tile points spatially
  - Compute local DTs
  - Broadcast axis-extreme points
- While points are sent
  - Insert points received by each tile (in batch)
  - Send points with new tile adjacencies
- Local DTs are now local views of the global DT
- Simplify local DTs



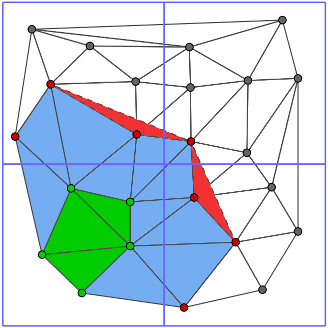
● : Local vertices, ● : Foreign vertices, ● : Redundant foreign vertices,

▲ : Local cells, ▲ : Mixed cells, ▲ : Foreign cells.

Proposal

Star Splaying Approach at the tile level (1-rings → local DTs)

- Initialize
  - Tile points spatially
  - Compute local DTs
  - Broadcast axis-extreme points
- While points are sent
  - Insert points received by each tile (in batch)
  - Send points with new tile adjacencies
- Local DTs are now local views of the global DT
- Simplify local DTs



● : Local vertices, ● : Foreign vertices, ● : Redundant foreign vertices,

▲ : Local cells, ▲ : Mixed cells, ▲ : Foreign cells.

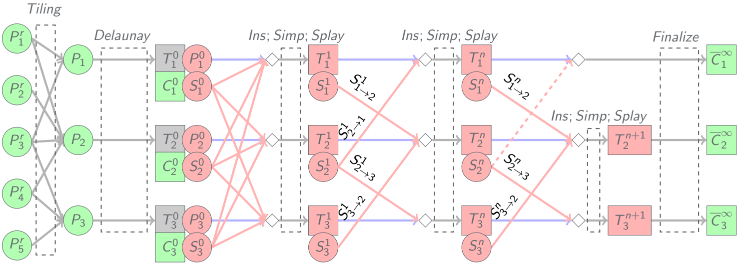
Why does it work ? Does it converge ?

Why does it work ? Does it converge ?

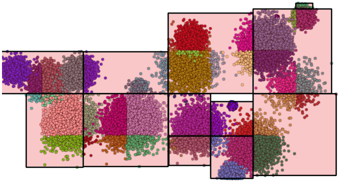
THEOREM 3. *Let V be a generic vertex set in E^{d+1}. Suppose that for every vertex $v \in V$ except the lexicographically minimum vertex, v's starting set W_v contains at least one vertex that lexicographically precedes v. Then star splaying constructs the boundary ∂H of the convex hull $H = \mathrm{conv}(V)$.*

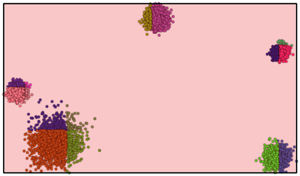
We're good ! (It's even a bit overkill...), because :

- All axis-extreme points are sent to all tiles
- So each tile receives the lexicographically minimum vertex
- Maintaining a local DT is equivalent to maintaining consistent 1-rings for its local points

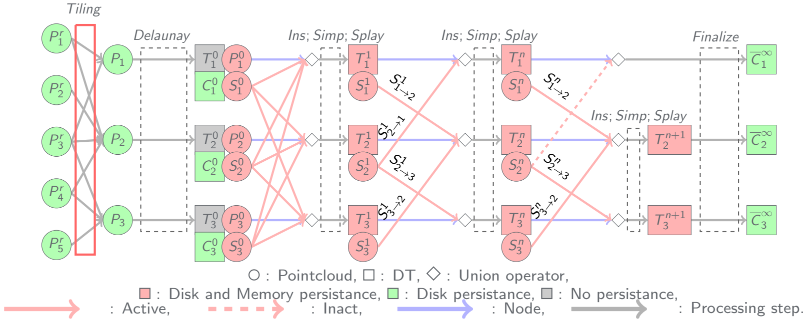


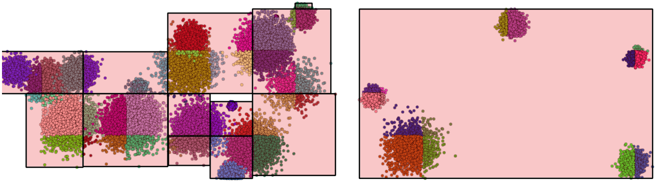
○ : Pointcloud, □ : DT, ◇ : Union operator,
■ : Disk and Memory persistance, ■ : Disk persistance, ■ : No persistance,
: Active, : Inact, : Node, : Processing step.



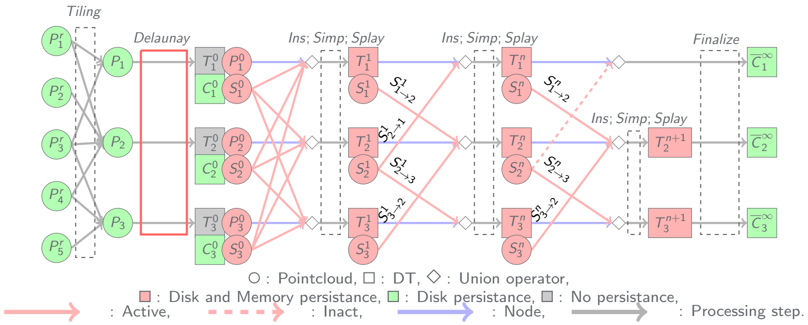


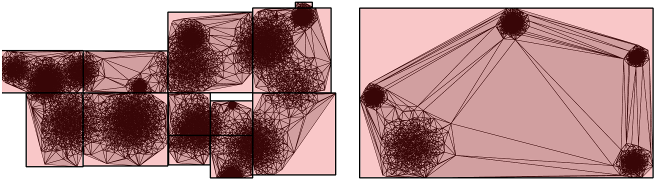
Distributed Delaunay Triangulation

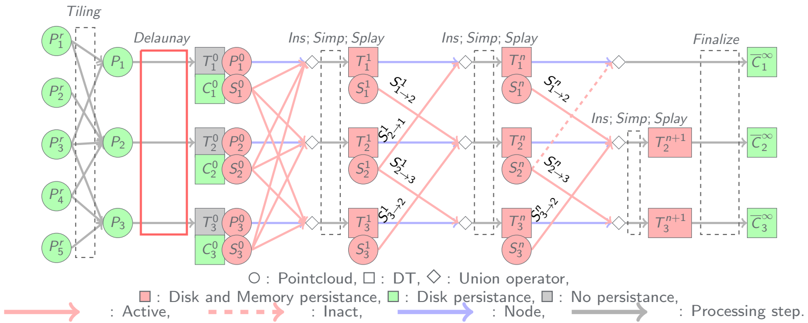


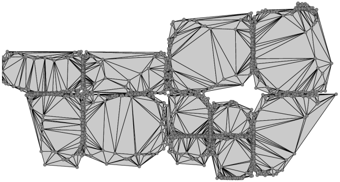


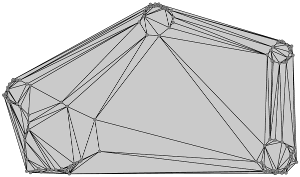
Distributed Delaunay Triangulation

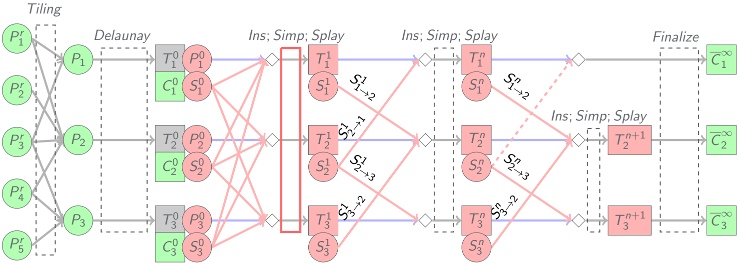




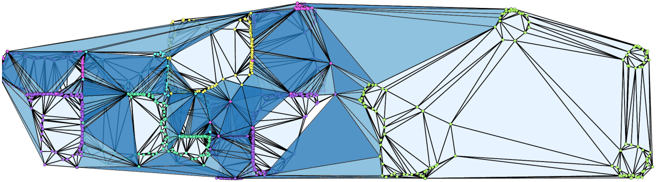

Tiling
Delaunay
Ins; Simp; Splay
Ins; Simp; Splay
Ins; Simp; Splay
Finalize
P_1^r P_2^r P_3^r P_4^r P_5^r
P_1 P_2 P_3
T_1^0 P_1^0 C_1^0 S_1^0
T_2^0 P_2^0 C_2^0 S_2^0
T_3^0 P_3^0 C_3^0 S_3^0
T_1^1 S_1^1 T_2^1 S_2^1 T_3^1 S_3^1
$S_{1\to2}^1$ $S_{2\to1}^1$ $S_{2\to3}^1$ $S_{3\to2}^1$
T_1^n S_1^n T_2^n S_2^n T_3^n S_3^n
$S_{1\to2}^n$ $S_{2\to3}^n$ $S_{3\to2}^n$
T_2^{n+1} T_3^{n+1}
$\overline{C}_1^\infty$ $\overline{C}_2^\infty$ $\overline{C}_3^\infty$
○ : Pointcloud, □ : DT, ◇ : Union operator,
: Disk and Memory persistance, : Disk persistance, : No persistance,
: Active, : Inact, : Node, : Processing step.



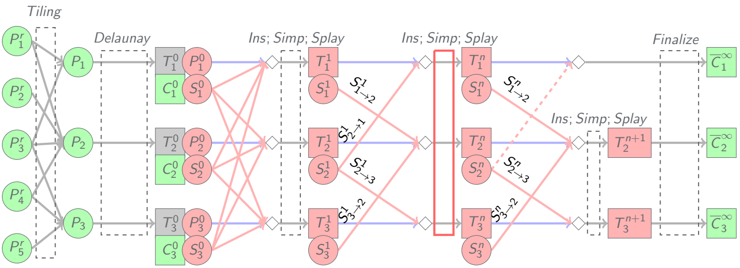




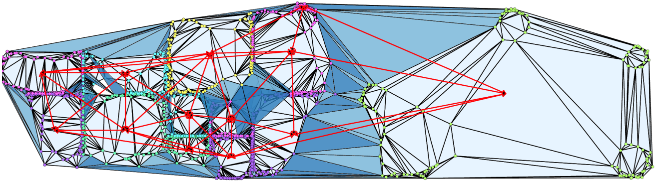
○ : Pointcloud, □ : DT, ◇ : Union operator,
□ : Disk and Memory persistance, □ : Disk persistance, □ : No persistance,
: Active, : Inact, : Node, : Processing step.



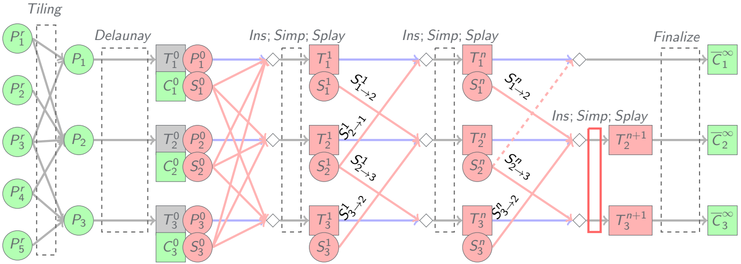
Distributed Delaunay Triangulation



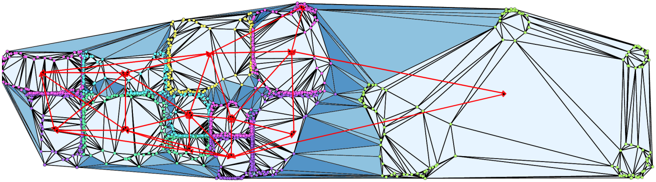
○ : Pointcloud, □ : DT, ◇ : Union operator,
■ : Disk and Memory persistance, ■ : Disk persistance, ■ : No persistance,
: Active, : Inact, : Node, : Processing step.

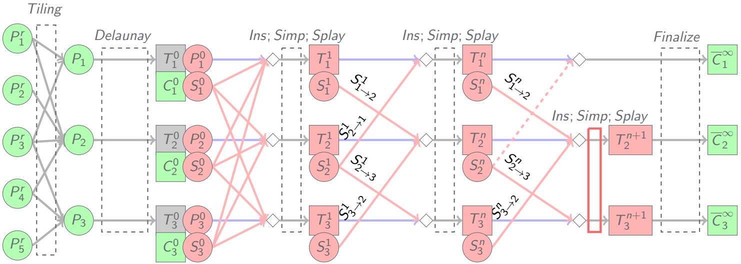


Distributed Delaunay Triangulation

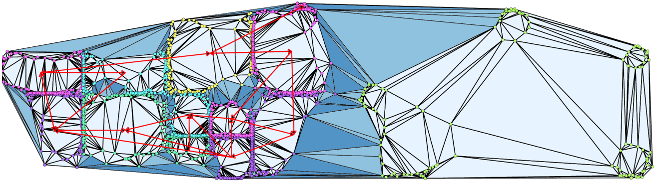


○ : Pointcloud, □ : DT, ◇ : Union operator,
□ : Disk and Memory persistance, □ : Disk persistance, □ : No persistance,
: Active, : Inact, : Node, : Processing step.

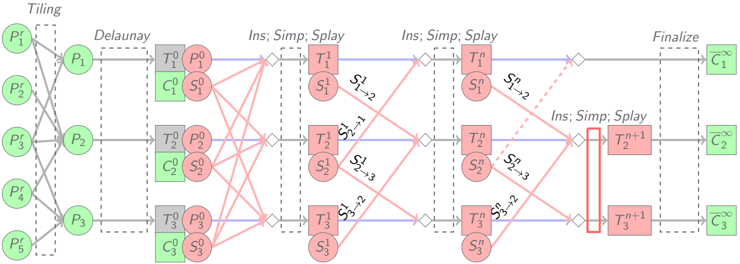




○ : Pointcloud, □ : DT, ◇ : Union operator,
□ : Disk and Memory persistance, □ : Disk persistance, □ : No persistance,
: Active, : Inact, : Node, : Processing step.



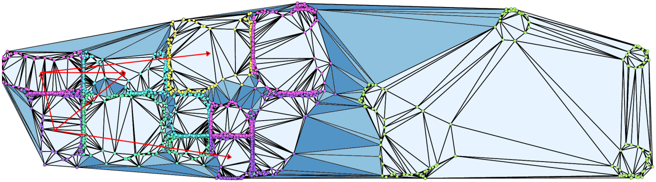
Distributed Delaunay Triangulation



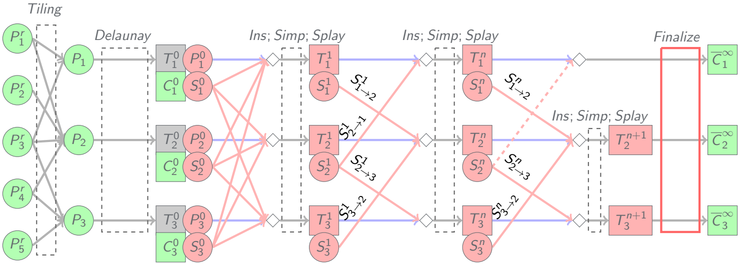
○ : Pointcloud, □ : DT, ◇ : Union operator,

□ : Disk and Memory persistance, □ : Disk persistance, □ : No persistance,

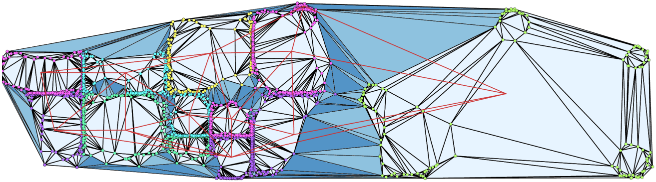
: Active, : Inact, : Node, : Processing step.



Distributed Delaunay Triangulation

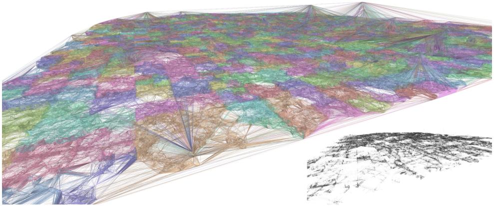


○ : Pointcloud, □ : DT, ◇ : Union operator,
□ : Disk and Memory persistance, □ : Disk persistance, □ : No persistance,
: Active, : Inact, : Node, : Processing step.

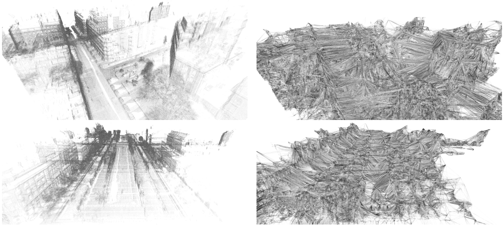


Resultats : Distributed Delaunay Triangulation of 1.9 billion points

Shared cells



Local cells

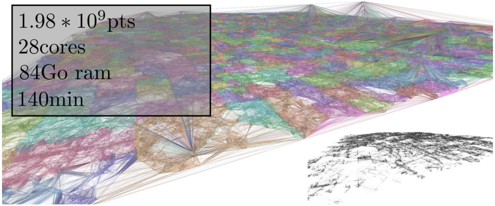


1 2

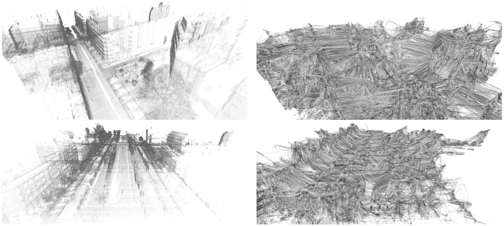
[1] Provably Consistent Distributed Delaunay Triangulation - ISPRS Annals (2020), 195–202

[2] Tile & merge: Distributed Delaunay triangulations for cloud computing - 2019 IEEE International Conference on Big Data (Big Data), 1613-1618

Shared cells



Local cells



[1] [2]

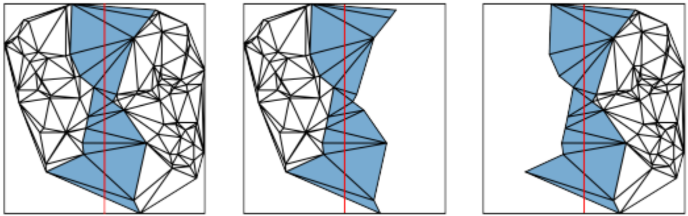
[1] Provably Consistent Distributed Delaunay Triangulation - ISPRS Annals (2020), 195–202

[2] Tile & merge: Distributed Delaunay triangulations for cloud computing - 2019 IEEE International Conference on Big Data (Big Data), 1613-1618

Distributed Graph-cut Formulation

Embarrassingly parallel :

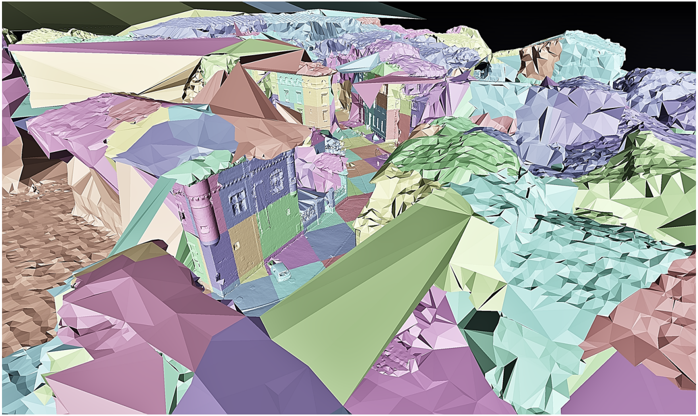
- The graph is the tetrahedron adjacency graph
  - 1 node per tetrahedron
  - 1 edge per triangle
- The Graph-cut energy terms are accumulated on each tetrahedron and each edge for each observation
  - cf undistributed case, many energies exist in the literature.



- The graph is split into unconnected graphs (1 per tile) by considering nodes for local and shared tetrahedra only
- Capacities of edges that are replicated in multiple tiles are divided by the replication count.
- Lagrangian variables are added to enforce consistent labels across replicated nodes.
- Algorithm runs until convergence [3]:
  - In parallel, solve the graph cut sub-problem in each tile
  - Update the Lagrangian variables

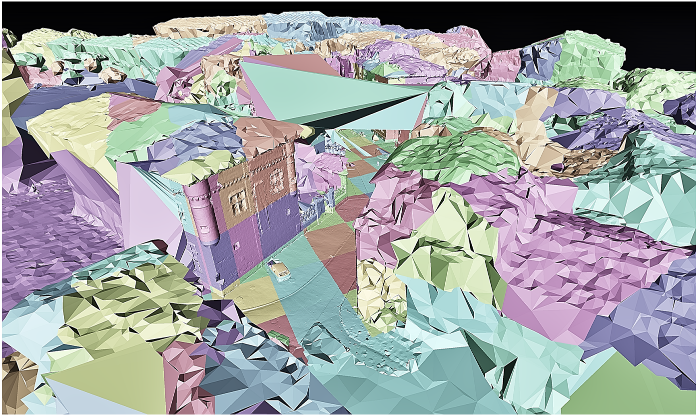
[3] Efficiently distributed watertight surface reconstruction - International Conference on 3D Vision (3DV), 2021

Distributed Graph-cut Optimization : Iterations

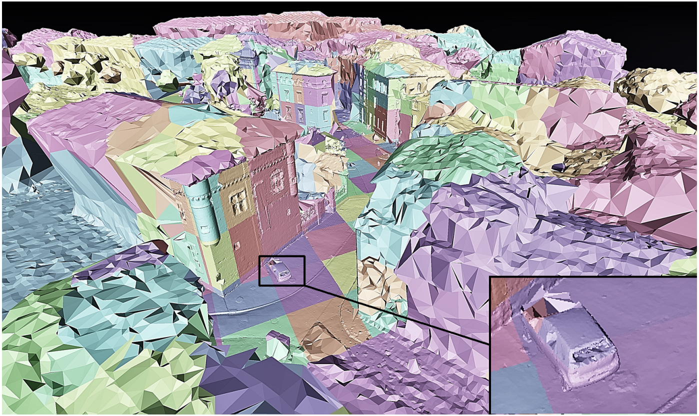


1 iteration

Distributed Graph-cut Optimization : Iterations

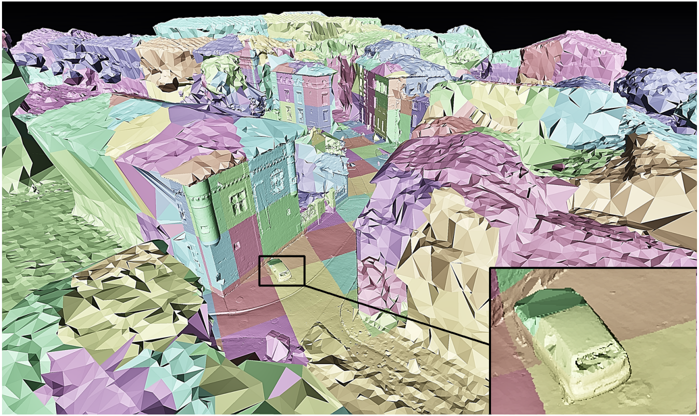


3 iterations



15 iterations

Distributed Graph-cut Optimization : Iterations



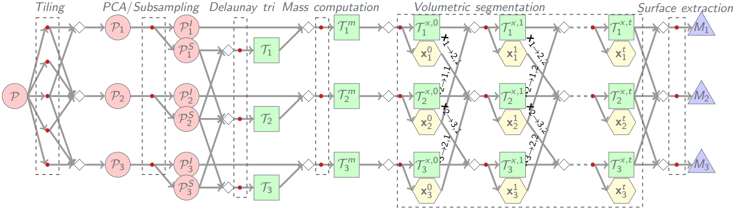
30 iterations

Distributed Surface Extraction

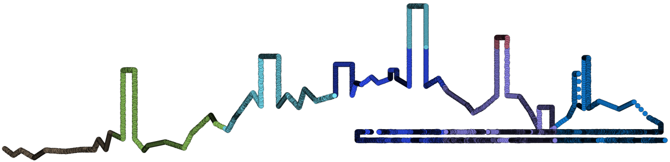
Embarrassingly parallel :

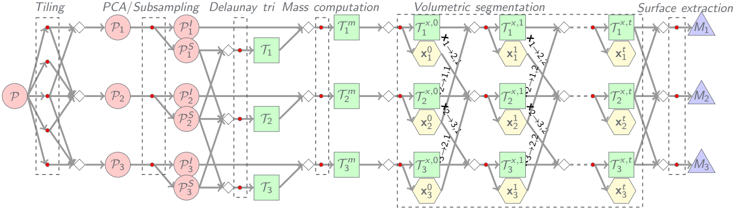
- Each tile yields independently its surface triangles (=between inside and outside tetrahedra) thanks to replicated tetrahedra with consistent labels.

Distributed Watertight Surface Reconstruction

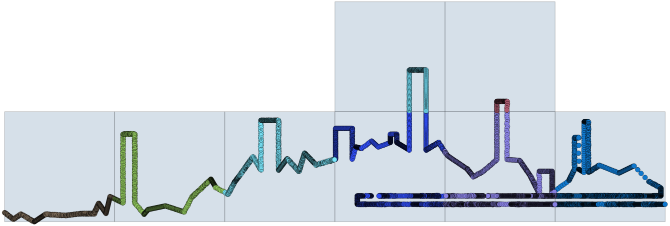


○ : Point set, □ : DT, ⬡ : Cell set, △ : Mesh, ◇ : Union operator,

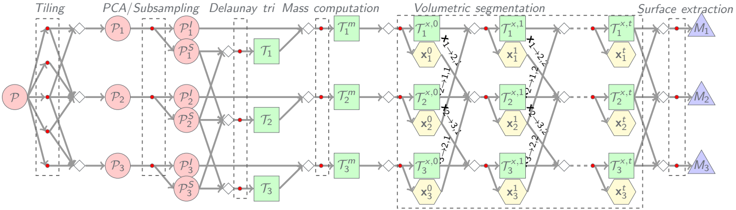




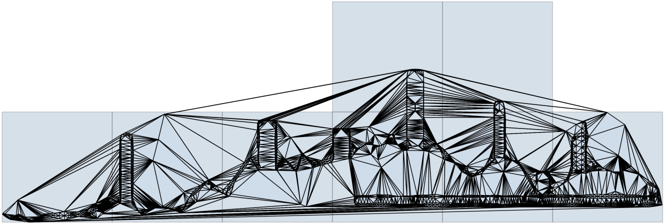
○ : Point set, □ : DT, ⬡ : Cell set, △ : Mesh, ◇ : Union operator,



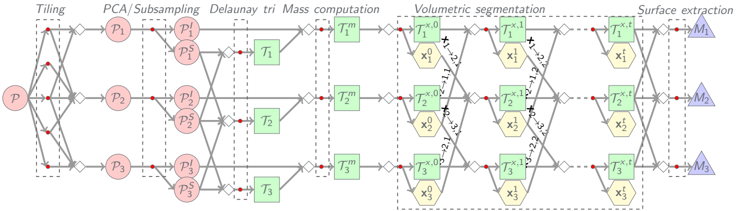
Distributed Watertight Surface Reconstruction



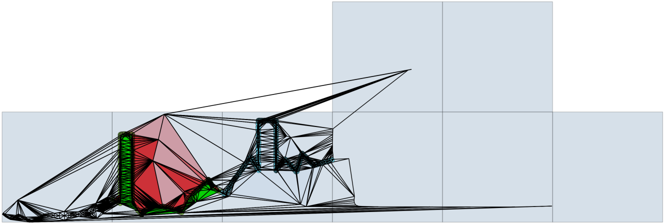
○ : Point set, □ : DT, ⬡ : Cell set, △ : Mesh, ◇ : Union operator,



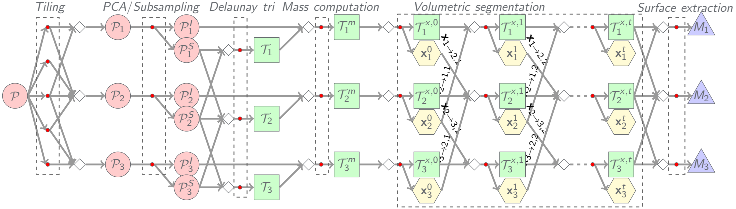
Distributed Watertight Surface Reconstruction



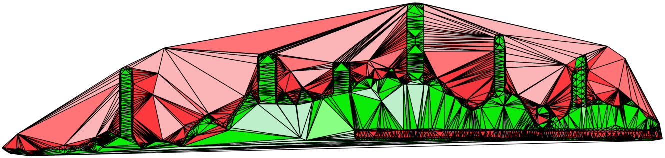
○ : Point set, □ : DT, ⬡ : Cell set, △ : Mesh, ◇ : Union operator,



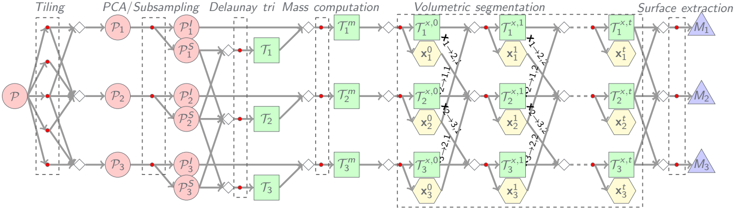
Distributed Watertight Surface Reconstruction



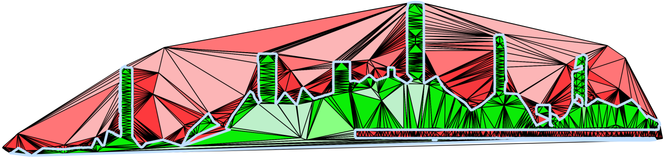
○ : Point set, □ : DT, ⬡ : Cell set, △ : Mesh, ◇ : Union operator,

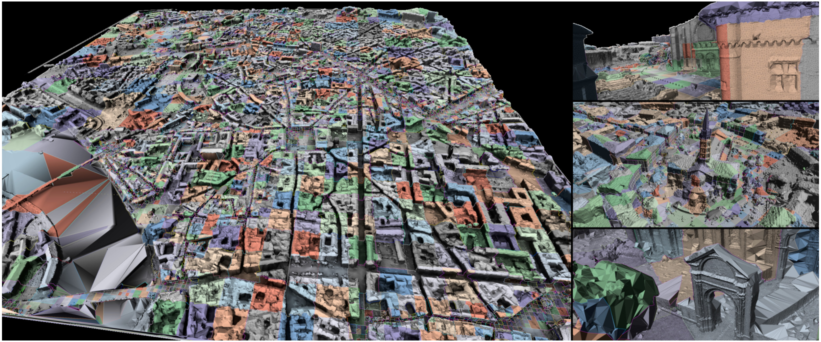


Distributed Watertight Surface Reconstruction



○ : Point set, □ : DT, ⬡ : Cell set, △ : Mesh, ◇ : Union operator,





Results on a scene with 350 million points [4]

- Implementation :
  - C++/CGAL processes
  - Apache Spark scheduling (24 cores)
- Computing time: 2h20

[4] Efficiently distributed watertight surface reconstruction - International Conference on 3D Vision (3DV), 2021

3

Distributed Delaunay Triangulations in CGAL ?

The Computational Geometry Algorithms Library
http://www.cgal.org

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_3.h>
#include <CGAL/IO/read_las_points.h>
typedef CGAL::Exact_predicates_inexact_constructions_kernel        K;
typedef CGAL::Delaunay_triangulation_3<K>                          Triangulation;
typedef typename Triangulation::Point                              Point;

int main(int argc, char*argv[])
{
    char* const* begin = argv + 1; // first filename of a las file
    char* const* end   = argv + argc; // after the last filename of a las file
    Triangulation tri;
    for(char * const* fname = begin; fname !=  end; ++fname) {
        std::ifstream in(*fname, std::ios_base::binary);
        std::vector<Point> points;
        CGAL::IO::read_LAS(in, std::back_inserter (points));
        tri.insert(points.begin(), points.end());
    }
    return EXIT_SUCCESS;
}
```

```cpp
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_3.h>
#include <CGAL/IO/read_las_points.h>

typedef CGAL::Exact_predicates_inexact_constructions_kernel        K;

typedef CGAL::Delaunay_triangulation_3<K>                           Triangulation;

typedef typename Triangulation::Point                               Point;

int main(int argc, char*argv[])
{
    char* const* begin = argv + 1; // first filename of a las file
    char* const* end   = argv + argc; // after the last filename of a las file
    Triangulation tri;
    for(char * const* fname = begin; fname !=  end; ++fname) {
        std::ifstream in(*fname, std::ios_base::binary);
        std::vector<Point> points;
        CGAL::IO::read_LAS(in, std::back_inserter (points));
        tri.insert(points.begin(), points.end());
    }
    return EXIT_SUCCESS;
}
```

Non-distributed CGAL code

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_3.h>
#include <CGAL/IO/read_las_points.h>
#include <CGAL/Triangulation_vertex_base_with_info_3.h>
#include <CGAL/DDT/traits/Vertex_info_property_map.h>






typedef unsigned char Tile_index;
typedef CGAL::Exact_predicates_inexact_constructions_kernel          K;
typedef CGAL::Triangulation_vertex_base_with_info_3<Tile_index, K> Vb;
typedef CGAL::Triangulation_data_structure_3<Vb>                    TDS;
typedef CGAL::Delaunay_triangulation_3<K, TDS>                      Triangulation;
typedef CGAL::DDT::Vertex_info_property_map<Triangulation>          Property;
typedef typename Triangulation::Point                               Point;






int main(int argc, char*argv[])
{
    char* const* begin = argv + 1; // first filename of a las file
    char* const* end   = argv + argc; // after the last filename of a las file
    Triangulation tri;
    for(char * const* fname = begin; fname !=  end; ++fname) {
        std::ifstream in(*fname, std::ios_base::binary);
        std::vector<Point> points;
        CGAL::IO::read_LAS(in, std::back_inserter (points));
        tri.insert(points.begin(), points.end());
    }
    return EXIT_SUCCESS;
}
```

Store the Tile index in the triangulations

```cpp
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_3.h>
#include <CGAL/DDT/tile_points/LAS_tile_points.h>
#include <CGAL/Triangulation_vertex_base_with_info_3.h>
#include <CGAL/DDT/traits/Vertex_info_property_map.h>
#include <CGAL/DDT/traits/Triangulation_traits_3.h>
#include <CGAL/DDT/serializer/File_serializer.h>
#include <CGAL/Distributed_triangulation.h>
#include <CGAL/DDT/scheduler/Multithread_scheduler.h>
#include <CGAL/DDT/IO/write_pvtu.h>

typedef unsigned char Tile_index;
typedef CGAL::Exact_predicates_inexact_constructions_kernel         K;
typedef CGAL::Triangulation_vertex_base_with_info_3<Tile_index, K> Vb;
typedef CGAL::Triangulation_data_structure_3<Vb>                    TDS;
typedef CGAL::Delaunay_triangulation_3<K, TDS>                      Triangulation;
typedef CGAL::DDT::Vertex_info_property_map<Triangulation>          Property;
typedef typename Triangulation::Point                               Point;
typedef CGAL::DDT::LAS_tile_points<Point>                           Points;
typedef CGAL::Distributed_point_set<Point, Tile_index, Points>      DPointset;
typedef CGAL::DDT::Multithread_scheduler                            Scheduler;
typedef CGAL::DDT::File_serializer<Triangulation, Property>         Serializer;
typedef CGAL::Distributed_triangulation<Triangulation, Property, Serializer>
                                                                    DTriangulation;
int main(int argc, char*argv[])
{
    char* const* begin = argv + 1; // first filename of a las file
    char* const* end   = argv + argc; // after the last filename of a las file
    DPointset points(begin, end);

    Scheduler scheduler(12 /* threads */);
    DTriangulation tri(3 /* 3D */, 4 /* tiles in memory */, Serializer("tmp"));
    tri.insert(scheduler, points);
    tri.write(scheduler, CGAL::DDT::PVTU_serializer("out")); // -> paraview

    return EXIT_SUCCESS;
}
```

Distributed Point Set : loads lazily LAS files

```cpp
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_3.h>
#include <CGAL/DDT/tile_points/LAS_tile_points.h>
#include <CGAL/Triangulation_vertex_base_with_info_3.h>
#include <CGAL/DDT/traits/Vertex_info_property_map.h>
#include <CGAL/DDT/traits/Triangulation_traits_3.h>
#include <CGAL/DDT/serializer/File_serializer.h>
#include <CGAL/Distributed_triangulation.h>
#include <CGAL/DDT/scheduler/Multithread_scheduler.h>
#include <CGAL/DDT/IO/write_pvtu.h>

typedef unsigned char Tile_index;
typedef CGAL::Exact_predicates_inexact_constructions_kernel        K;
typedef CGAL::Triangulation_vertex_base_with_info_3<Tile_index, K> Vb;
typedef CGAL::Triangulation_data_structure_3<Vb>                   TDS;
typedef CGAL::Delaunay_triangulation_3<K, TDS>                     Triangulation;
typedef CGAL::DDT::Vertex_info_property_map<Triangulation>         Property;
typedef typename Triangulation::Point                              Point;
typedef CGAL::DDT::LAS_tile_points<Point>                          Points;
typedef CGAL::Distributed_point_set<Point, Tile_index, Points>     DPointset;
typedef CGAL::DDT::Multithread_scheduler                           Scheduler;
typedef CGAL::DDT::File_serializer<Triangulation, Property>        Serializer;
typedef CGAL::Distributed_triangulation<Triangulation, Property, Serializer>
                                                                   DTriangulation;
int main(int argc, char*argv[])
{
    char* const* begin = argv + 1; // first filename of a las file
    char* const* end   = argv + argc; // after the last filename of a las file
    DPointset points(begin, end);

    Scheduler scheduler(12 /* threads */);
    DTriangulation tri(3 /* 3D */, 4 /* tiles in memory */, Serializer("tmp"));
    tri.insert(scheduler, points);
    tri.write(scheduler, CGAL::DDT::PVTU_serializer("out")); // -> paraview

    return EXIT_SUCCESS;
}
```

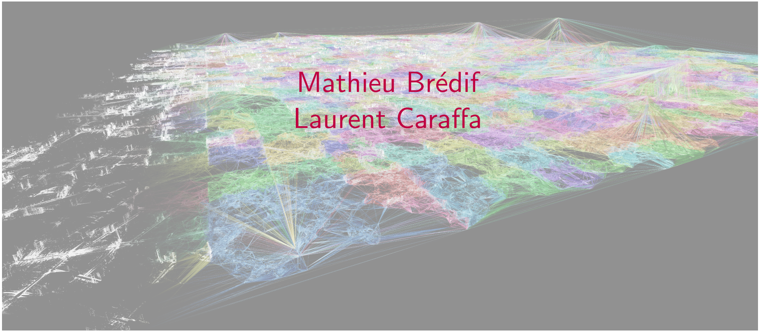
Distributed Triangulation : starsplaying with the Scheduler

Distributed Delaunay Triangulations in CGAL ?

- Star Splaying works in all dimensions:
  - Wraps the 2d/3d/static-Nd/dynamic-Nd specific calls, the rest being mostly unaware of the ambient dimension
- `Scheduler`: implements various scheduling policies
  - Sequential, Multithread, TBB... (MPI is WIP)
- A vertex is local if its id is equal to the tile id
- `Serializer`: memory (un)loading for out-of-core or streaming use cases
- `Distributed_point_set` loads lazily point sets.
- `Distributed_triangulation` provides vertex/facet/cell iterators over the overall triangulation, hiding the tiling.

IGN
INSTITUT NATIONAL
DE L'INFORMATION
GÉOGRAPHIQUE
ET FORESTIÈRE

Merci.



Mathieu Brédif
Laurent Caraffa