EMODnet High Resolution Seabed Mapping (HRSM)

EMODnet Phase III

Quantized Mesh Generator for Cesium 3D visualisation

CONTRACT NUMBER – EASME/EMFF/2015/1.3.1.7/SI2.742125
Call No. EASME/EMFF/2016/005

Date: 12/12/2018

Prepared by: Ricard Campos (CORONIS), Josep Quintana (CORONIS), and Rafael Garcia (University of Gerona)

Table of Contents

Introduction

Nowadays rapid grow of information sources allows mapping the surface of the earth with increasingly finer scales. Computationally-efficient visualization of such world-scale data, possibly of very large resolution, has been a hot topic in recent years. In this direction, many standards for web-based rendering of large 2D maps have been developed recently, and several services, including OpenStreetMaps or Google Maps, have adopted them to render world-scale imagery. Unfortunately, a standard for transferring and displaying terrain geometry on the web is still not fully defined.

EMODnet Bathymetry aimed at updating the visualization of the global DTM to 3D. In order to achieve this goal, we wanted to use web-based visualization applications. In these applications, huge amounts of data needs to be passed through the net and the rendering happens on the user side. Thus, it is important to keep a balance between the amount of data to transfer and the amount of effort required for rendering it.

Consequently, Level of Detail (LOD) techniques, able to change the complexity of the displayed data based on the point of view required by the end user, are desirable in our context. These techniques focus on rendering the part of the world falling in the user's frustrum with a complexity that adapts to the distance from the viewer or the projected screen size. Indeed, at increasing distances from a given point of view, the data will project to less and less pixels on screen and, consequently, its details will not be visible. Therefore, the complexity of the data should adapt to the perception of the user given a point of view.

Inspired by the popularity and large adoption of these techniques for 2D image visualization, we propose to render the terrain using a multiresolution pyramidal tiled data structure.

After studying the state of the art in web-based 3D visualization libraries, we concluded that the best option was to use the Cesium library (https://cesiumjs.org/). Cesium implements a format called quantized-mesh-1.0 [QMSpec], allowing to represent terrains as Triangulated Irregular Networks (TIN), where the coordinates of its vertices are quantized within the bounding box of the mesh. Having a TIN instead of a regular grid is a better representation of the data, as the complexity of the map (i.e., the number and size of triangles) adapts to the variations of elevations in the scene.

However, **there is no free or open-source tool available nowadays** that is able to create tiles in quantized mesh format out of a raster format such as the ones generated so far in the EMODnet project. Thus, we have implemented this software ourselves in the following way: given a regularly gridded digital elevation

model (DEM), we create a pyramid of different LODs, where each LOD is a Triangulated Irregular Network (TIN) further subdivided into small regular tiles, following a structure similar to that of a quadtree subdivision (see Figure 1).

Building this structure from large-scale high-resolution data presents some issues. If we tackle the problem of creating a given LOD by simplifying the whole terrain, this is likely to require huge amounts of memory, and thus renders this process not amenable on commodity hardware. One can think that, since our structure is tiled, we can try to build the LODs at tile level, which would effectively solve the memory issues. However, the problem then is to maintain coherence between tiles. After all, when put together, they should form a single **continuous surface**. However, leveraging per-tile simplification with surface continuity is an open issue.

Therefore, in this project we focused on the insights of creating a multiresolution pyramidal tile-based data structure for large terrains, where all the processing is performed at tile level, while making sure the borders of the tile are coincident within a LOD. The key idea is to restrict the new tiles to be computed to stick to the borders of already triangulated ones. With this restriction in mind, we show how different TIN creation methods of the state of the art can be adapted to work at tile level, making in this way the generation of this data structure for large terrains amenable to low-end hardware.

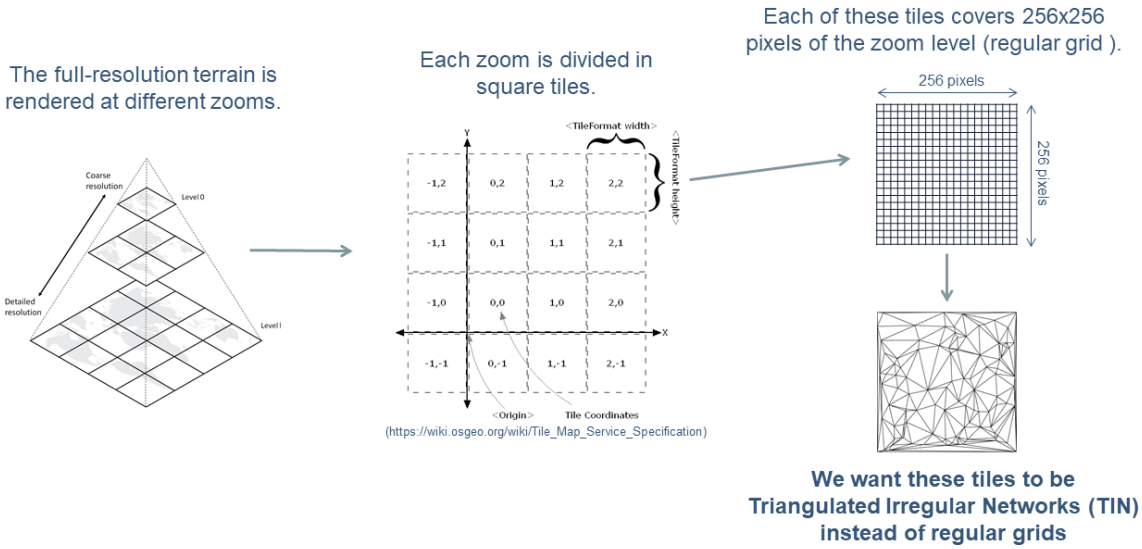


Figure 1. Overview of the proposed methodology. We aim at creating a multi-resolution tiled representation of a terrain as the one shown on the left hand side. Each of the LOD (also called zooms) is tiled following a regular grid. As shown on the right hand side, each of these tiles covers a 256x256 footprint of a raster file, and we aim at converting this regular

representation into a TIN.

Overview

Since there is no comparison in the literature regarding the performance of simplification methods when applied to terrain models, we decided to test several of the methods in the state of the art in order to decide which one would produce the results that best fit our needs. Therefore, our main objective was to adapt different TIN creation methods of the state of the art to the problem of creating a hierarchical pyramid of tiles out of a large scale high resolution regular gridded terrain.

To avoid having large amounts of data in memory while creating the TIN, we adapted existing and consolidated simplification algorithms to be able to work on small chunks of data. More precisely, given a zoom level within the multiresolution pyramid, we required the algorithms to be able to work at tile level. Since we will be processing regions of continuous data independently, we need the borders of the different tiles to coincide. That is, our focus will be put on **maintaining the proper coherence between tiles** while restricting simplifications as less as possible.

Even if performed off-line, the simplification of world-scale terrains should benefit from **parallel processing**. Since we **operate at tile level**, the process is highly parallelizable. However, as already mentioned, the tiles must coincide at their vertices. This presents some restrictions on what tiles can be processed in parallel at a given moment.

In this context, we adapted three popular types of simplification algorithms to our needs:

- **Greedy insertion**: it performs coarse-to-fine simplification. This algorithm starts from a very basic triangulation (for instance, the two triangles resulting from triangulating the vertices on the bounding box of the terrain in the XY plane). By keeping track of the points falling within each triangle in the XY plane, the point inducing the largest error is added to the surface at each iteration, until all the points are within a user-defined error.
- **Edge-collapse simplification**: given a gridded terrain or a TIN, it creates an approximation of it by iteratively applying an edge collapse operation. As its name suggests, an edge collapse consists in merging the two endpoints of an edge in a single point. At each iteration, it selects to collapse the edge that would induce least error.
- **Point set simplification**: in this case, the input is seen as a point set, without connectivity. Without the restriction of not having to stick to a mesh, these methods are easier to implement and more versatile. After simplifying the point sets, the

mesh/connectivity needs to be reconstructed somehow (by means of triangulating the points in the XY projection plane, for instance).

This section will start by reviewing how we create the tile pyramid, regardless of the simplification method used. Then, we will focus on the specific changes made to each simplification algorithm to be able to work on our framework (that is, to be able to maintain border edges as required). Finally, we will comment on additional issues that are to be taken into account when creating a world-scale dataset.

1.1 Creation of the pyramid of tiles

We consider as input a regularly gridded digital elevation model *H*. From this input, we aim at creating different resolutions or, following the nomenclature of tiled web maps, zoom levels. Then, within each zoom, we will further divide the mesh into regular square tiles of a fixed size. The hierarchical tiled structure follows a quadtree subdivision of the world: starting at the world within one (or two) base tiles, each of this tile is further subdivided in four tiles in the next zoom level (see left hand side of Figure 1). Each of these tiles can be indexed within a given zoom *z* by its tile coordinates, that is, their coordinates within the *x* and *y* position *H(z,x,y)*.

Creating the described structure for regularly gridded maps is straightforward. In fact, *H* can be considered as an image where the elevation value is encoded in each pixel value. Thus, crafting different resolutions is just a matter of subsampling the image at the required resolution, and tiling a given zoom just requires cutting the image following a regular pattern. When applied to visualizing terrain data, it also requires to triangulate the points in the XY plane so that we can then lift them to 2.5D using the elevation value of each cell. However, it has been proven that sticking to a regular grid when representing terrain data is far from ideal, as the resulting meshes are overly complex when compared to the variation of elevation present in a tile. In our approach we want each of these tiles *H(z,x,y)* of regularly gridded data to be represented by a more efficient TIN representation *T(z,x,y)*.

Common wisdom would say that building this multiresolution TIN structure, that we will refer to as *T*, can follow an approach similar to that of regular data: instead of subsampling the mesh at a given level, one only has to take the regular data, triangulate it in the 2D plane, and simplify the resulting mesh at a given resolution to finally cut it in square tiles following a regular pattern. The problem comes from the fact that simplifying a large terrain requires being able to maintain in memory the whole mesh, and this is unfeasible for world scale large resolution terrains as the ones we are considering. Note that this problem has been obviated in the approaches in the state of the art dealing with this same type of structure

[Christen11], and this prevents them from being applied on large terrain data. The examples they provide in those references support this statement, as their detail and resolution are far from the scale of the terrains we are considering in this project. Consequently, we want to tackle the problem of terrain simplification at tile level.

Thus, given a zoom level within the pyramid, we divide *H_z* in tiles, and then process each tile separately. Note that, as opposed to what happens when rendering a tiled image, the borders of neighbouring *T(z,x,y)* tiles should overlap to conform a continuous triangle mesh. Since these tiles should stitch together seamlessly, our main concern is to make the borders at *T(z,x,y)* coincident with those of its neighbouring tiles. Thus, the input of all the simplification methods below will be a triangulation *D(z,x,y)*, constructed using a Delaunay triangulation of, on the one hand, the border vertices inherited from already built tiles and, for the remaining area within the footprint of the tile, the regularly gridded data from the original terrain.

Therefore, in order for a simplification method to be applicable at tile level, it needs to be able to:

C1: Preserve the square shape of the tile in the XY plane. Surprisingly enough, most of the state of the art simplification methods do not take into account the preservation of the border shape. We need the simplification algorithm not to modify the tile coverage in the XY plane, so as to avoid gaps when they are visualized together.

C2: When the tile to construct is adjacent to an already constructed tile, we need to **preserve the borders of the tile** inherited from the neighbouring tiles. Unconstrained simplification of neighbouring tiles may result in different vertices at the borders, which will result in cracks on the surface [Ulrich02].

In the following sections, we will show how we modified several simplification methods to be able to fulfil the constraints **C1** and **C2** above.

1.2 Greedy Insertion

There are several variants of greedy insertion for terrains, especially in the early days [Fowler79,Polis93,Puppo94], but the most widely known and used is the one described by Garland and Heckbert [Garland95].

The method starts with a very rough triangulation of the data in the XY plane, which is typically the two triangles resulting of triangulating the four corners of the tile in the XY plane. Then, for each of the remaining points from the initial sample we compute the error from this approximation to the points. At each iteration, we select the point inducing the largest error and insert it in the triangulation (Delaunay insertion). Given the newly generated triangles, the errors

for each of the remaining points are recomputed, and the insertion process is repeated until all the errors are below a given threshold.

Modifying this method to adapt to the restrictions **C1** and **C2** is quite simple, as it only requires directly adding some vertices as part of the initial triangulation. As in the original reference, the restriction **C1** of maintaining the square shape of the tile is directly imposed by adding the corners of the tile as initial samples. In the same direction, constraint **C2** can be imposed by also adding as initial samples the vertices at the constrained borders of the mesh.

1.3 Edge-collapse Simplification

Simplification of 3D meshes has been a prolific field of research in the last years. Despite there are some other methodologies (e.g., vertex simplification), most of the approaches in the state of the art use edge-collapse operations to incrementally simplify the surface [Garland97,Lindstrom98]. As its name suggest, an edge collapse operation consists of changing one of the edges of the mesh to a single vertex, simplifying like this the mesh.

Basically, an edge collapse method requires to define two main ingredients: (1) the cost of collapsing an edge and (2) the vertex placement, i.e., how to choose the position of the vertex that replaces the edge after collapsing. The process starts by computing the collapsing cost for each edge in the mesh, so that they can be put in a priority queue. At each step, the edge representing the smallest cost is collapsed and, consequently, the connectivity of the mesh involving the triangles incident to the edge or its vertices are updated, and the costs of the edges involved are re-computed. The process ends at a given termination condition, that usually is related to the number of primitives.

While some constraints are used in the original methods to maintain the shape of the borders of a mesh [Lindstrom98], directly applying edge collapse in a tile is likely to remove the its corner vertices.

To avoid this, we modify the vertex placement operation. Thus, we override the common vertex placement defined by the algorithm. Given an edge to collapse, we take into account the following cases:

- If it is incident to a corner vertex, the result of collapsing that edge is always the corner vertex. This prevents the corners from disappearing of the mesh.
- If it is on a constrained border of the tile, the collapse is not possible. Thus, the cost of collapsing this edge is set to infinity to avoid the vertex placement from ever occurring.
- If one of its endpoints is on the border of the tile, the result is always that vertex. This maintains the borders of the mesh on a straight line in the XY plane. Note that,

when both endpoints of the edge are on the border, but this edge is not constrained, this restriction does not prevent the border from being simplified, it only restricts the result of the edge collapse to be one of the vertices of the edge, and thus maintain the square shape of the border of the tile.

While the modifications proposed can be used in virtually any edge collapse method, we applied them to the memory-less method presented by Lindstrom and Turk [Lindstrom98]. More specifically, we modified the implementation available in the CGAL libraries [CGAL-Simplification]. An example of the method working on a single tile is presented in Figure 2.

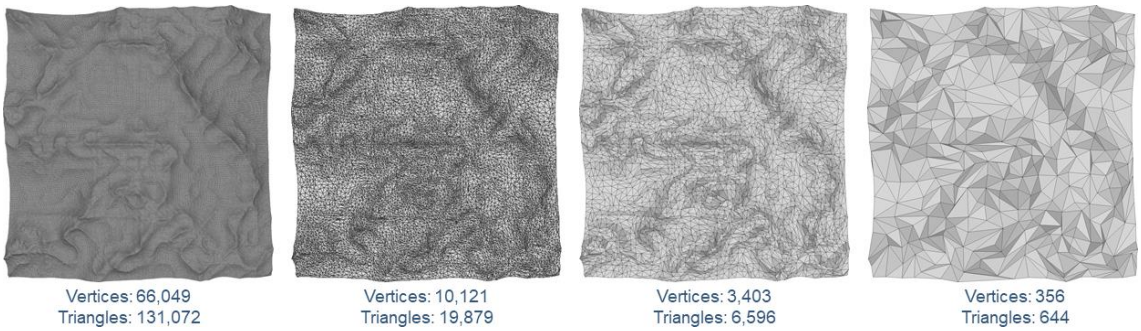


Figure 2. Different levels of detail of a tile using the edge collapse method. Notice how the reduction in complexity adapts the geometry to the variance in the input terrain, and how in all the cases our modifications preserve the square shape of the tile.

1.4 Point Set Simplification

Several approaches dealing with point sampled surfaces have appeared recently [Pauly02, Huang09]. These methods assume no known connectivity of the points, so they are the ones requiring more care to fulfil constraints **C1** and **C2**. Moreover, as opposed to the previous approaches, sharp creases are prone to vanish when dealing with point set simplification strategies. This is mainly because most of these methods assume the sampled surface to be smooth, an assumption that may not hold true for terrains containing the crests and ridges typical of natural landscapes.

For both reasons, we treat the simplification of borders and sharp edges separately. We perform two separate simplifications: polyline simplification of the borders and sharp edges and point set simplification for the rest of vertices not in those edges. We understand by polylines a piecewise linear curve defined by a sequence of points joined by line segments. Since our points are 3D, they are 3D polylines. In this way, we can apply any point set simplification method without worrying of losing the square shape of the tile or smoothing out relevant features.

Given our input tile, we start by identifying the polylines to simplify. Given the base triangulated tile *D(z,x,y)*, we consider as a sharp edges those whose incident triangles form a dihedral angle larger than a threshold *t* (fixed to *t*=60 degrees).

After detecting those edges, we trace polylines by joining those edges having common endpoints, and stopping and creating new ones when more than two edges converge into the same point. We also impose a minimum number of edges on the polylines to be considered as such. Since these polylines come from the original regularly gridded data, they are too complex and need to be simplified.

The most commonly used polyline simplification algorithm is the Douglas-Peucker algorithm [Douglas73]. However, the method only works on a single polyline, without having into account that we may have more than a polyline in the terrain. If we treat each polyline separately, there is a high change that, when simplified, these polylines intersect (cross) each other. This is not desirable in our case, since this would mean that different structures in the terrain are changing its topology. For this reason, we decided to use the method presented in [Dyken09], which preserves the topological relationships between the polylines simplified in the 2D plane.

The algorithm works in a coarse-to-fine manner, starting from the original polylines it iteratively removes a point from one of them. As in all simplification methods, the selection of which point to remove next is governed by a measure defining the error of removing a given point. At the beginning, the 2D curves are input as part of a 2D Constrained Delaunay Triangulation (CDT). At each iteration, the point with the smallest error is selected as candidate for removal. However, before removing it, the algorithm checks if its removal would modify the topology between curves. Since the points are in a CDT, the union of triangles having that point as vertex is collected. Then, if the segment resulting from removing that point is contained within that union of triangles, the candidate point can be removed safely. On the contrary, if the segment intersects the boundary of the union, its removal would change topology between curves, and therefore is not possible to remove it. An example of this is shown in Figure 3.

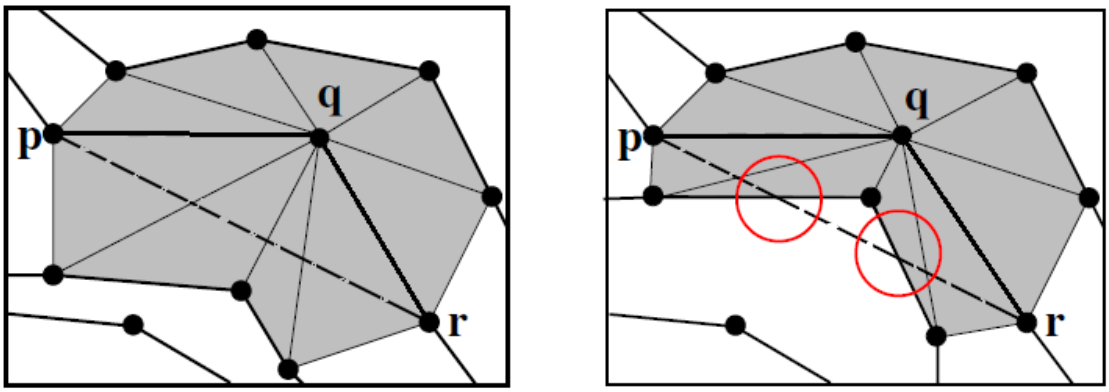


Figure 3. Visual depiction of the restrictions when simplifying more than a polyline. Image extracted from CGAL's documentation

(https://doc.cgal.org/latest/Polyline_simplification_2/index.html)

The main problem of this curve simplification method is that it is designed for simplifying 2D curves, but our curves are made up of 3D points. In this sense, we constructed a hybrid 2D-3D algorithm. That is, the simplification happens in 2D, as in the original method [Dyken09], but we compute the error measure guiding the removal ordering in height. Thus, for each candidate point in the curves to simplify, we compute as error the point-to-segment Euclidean distance in height. Note that, as in the original method, this point-to-segment distance is computed not only between the point to eliminate and the resulting segment, but also between all the points that were already eliminated and that were, at some moment, between the endpoints of the resulting segment (the reader is referred to the original reference [Dyken09] for more details on this procedure).

After simplifying the feature and border polylines, we apply a point set simplification strategy with the remaining vertices of *D(z,x,y)*. Since we already took care of constraints **C1** and **C2**, we can use the point set simplification method of our choice without modification. As a proof of concept, in this project we were able to apply each of the four methods implemented in the CGAL library [CGAL-PointSetSimplification]:

- **Grid**: divides the input point set in a regular grid, and selects a representative point from each cell.
- **Random**: randomly selects a fraction of the input points.
- **Hierarchy**: adaptive simplification of the point set through local clusters, implements the method in [Pauly02].
- **Weighted Locally Optimal Projection (WLOP)**: simplifies and regularizes a point set using the method in [Huang09]

Finally, in order to mix both contributions -polylines and points- in a single triangulation, we take advantage of the CDT resulting of the polyline simplification step. Basically, we insert the 2D projections of the simplified 3D points in the XY plane into the CDT to get the final triangle mesh. A sample of why we need this two-steps process is depicted in Figure 4.

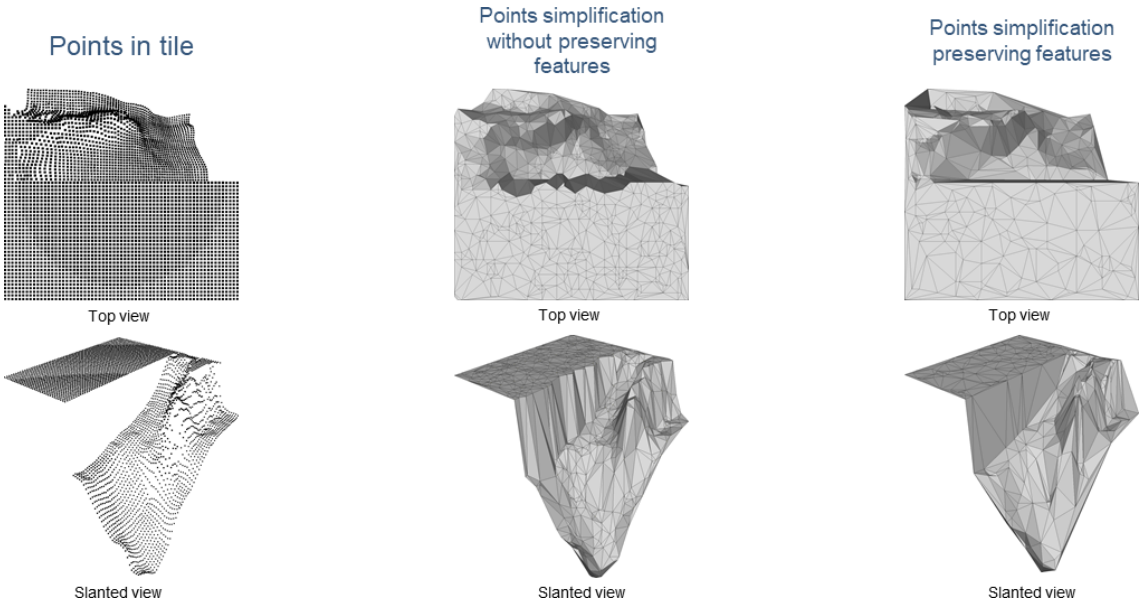


Figure 4. Point set simplification. We find the original point set on the right. In the middle, one can see the result of applying a point set simplification to the points directly. You can observe how the square shape of the tile is corrupted, and how the vertical "cliff" is not preserved. By using our two-steps method, as visible in the right part of the figure, we preserve the straight boundaries of the tile, as well as the sharp features present in the original data.

1.5 Implementation Details

In this section we will provide some implementation details that may not be obvious from the descriptions in the previous sections and that are key to our processing.

1.5.1 Scaled Coordinates

To be independent of the coordinate system of the input terrain, we apply the simplification of a given tile in scaled coordinates. That is, each coordinate is normalized to *[0..1]* according to the maximum and minimum of the values within the tile. This normalization is required because of the huge difference in scales for coordinates when working, for instance, with a raster dataset in WGS84 coordinate reference system. In this case, having latitude/longitude for X and Y and height in meters, which have values that are normally orders of magnitude different, may cause the triangles to degenerate, and specially provide numeric problems due to the huge difference in scales.

Nonetheless, when applying the point set simplification strategies, the point sets need to be transformed again. Since point set simplification techniques are normally designed to be applied on a metric space, the operations do not allow for a non-uniformly scaled coordinate system. Therefore, in this case, we apply point set simplification in metric Earth-Centered Earth-Fixed coordinates.

1.5.2 Parallelization

Due to the simplification being defined at tile level, parallelizing the processing is possible at zoom level. Given a zoom level to process, we define a *schedule*, that is, a preferred order to process the tiles. While the schedule defines a desired order, we try to spawn the creation of as many tiles as possible. Therefore, we keep track of the tiles being processed at each moment, and just spawn the process of the tiles that are not in the 8-neighborhood of already processing tiles. For each generated tile, we collect the vertices on its border, and store them in a temporary cache. When spawning the creation of a new tile, the first we do is check whether there are borders for the tile inherited from already constructed ones, and use them to create *D(z,x,y)*. Once the tiles incident to that borders are created, the information is removed from the cache to save memory. Note that this cache is only used for speed and that, if a completely memory-less version is required, one could load all the borders from the tiles already stored on disk.

However, it is worth noticing that parallelizing the process has an impact on the results. After all, the shape of the borders of the tiles greatly depends on the order in which we decide to process them. Moreover, by parallelizing the tiling as described, the results are not deterministic, since the ordering at which tiles are processed also depends on the processing time required for each tile.

1.5.3 Per-zoom Parameter Setting

Note that each method has its own set of parameters that need to be carefully tuned. Moreover, the parameters for the algorithm are scale-dependant and, as such, they should depend on the zoom level of the pyramid.

Consequently, we allow setting the parameters of the simplification per zoom level separately, allowing the user to input a value for each of the depth levels of the multiresolution pyramid. This is the most elegant solution for those parameters that do not depend on the scale of the data, such as the number of edges in the edge collapse algorithm. However, if a parameter is scale-dependant, we can take advantage of the quadtree structure ruling the tiling of the pyramid, so that we can just set the values of the parameters for the root level, and infer the rest. If we refer to a parameter for a zoom z as p_z, we allow the user to just set p_0 and then we compute the values for other zooms as $p_z = p_0/2^z$, for those parameters whose scale needs to be lowered at deeper levels, or $p_z = p_0 2^z$, for those whose scale needs to grow with depth.

1.5.4 Usage in Virtual Globes

In our application, the created terrain will be visualized in a virtual globe. A virtual glove renders the world in spherical coordinates, so we need to take special care of the simplifications at the lower zoom levels of the pyramid.

At these levels, the curvature of the ellipsoid of the earth is more prominent than the underlying terrain. However, since we are performing the simplification in projected space, in the XY plane, simplification at those levels may get to an overly simplified terrain for those shallow zoom levels. For instance, at zoom 0, the height data is negligible compared to the extension of the terrain in the XY plane contained in the tile. Thus, applying any of the simplification methods directly to the data may result in as few as two triangles covering the whole tile. Obviously, when using this tile in a spherical view, the shape of the world ellipsoid is completely lost.

Therefore, at those scales, we impose a minimum complexity. The way in which we impose it depends on the method we are applying:

- **Greedy insertion**: we just add a regular grid of samples to the initial samples when simplifying low depth zooms.
- **Edge collapse simplification**: we set the *shape* weighting of the cost function on the original reference [Lindstrom98] to a non-zero value. This forces the shape of the triangles to be close to regular.
- **Point set simplification**: we only impose a maximum size for border edges. Since point set simplification methods are not prone to oversimplifying the data, no other change is needed in this case.

Results

In order to provide a qualitative comparison of the behaviour of the methods, we first present in Figure 5 the results of applying the different TIN creation strategies applied to the data.

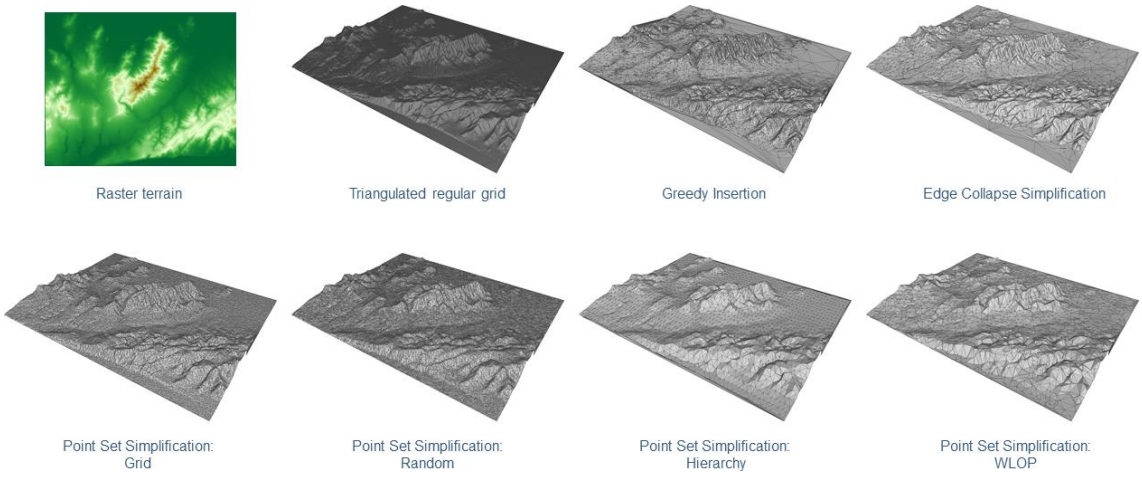


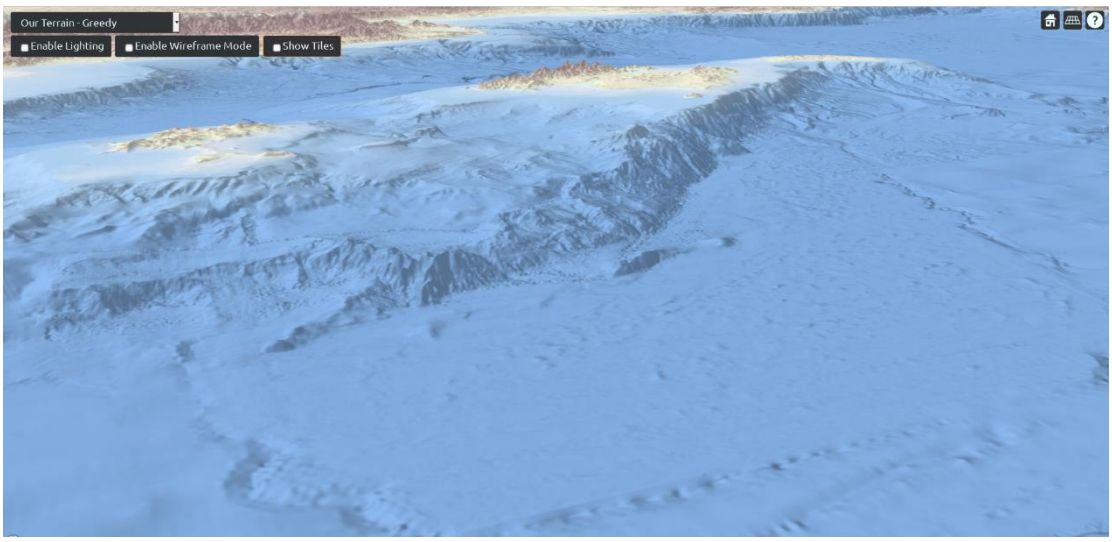
Figure 5. From left to right, top to bottom, we can see the original raster as an image, the result of triangulating all the samples in the terrain (and the huge number of triangles that this produces), and then the result of applying the different simplification methods studied in

this project (the name of each appears below its corresponding subfigure).

We can observe how the different methods provide different results in terms of the number of triangles, their shape and their adaptability to the terrain variance. Both the greedy insertion and edge collapse methods provide meshes whose triangles are larger in planar areas and smaller in places where the terrain presents larger variance. For the point set simplification methods, the grid, random and hierarchy approaches provide vertices that present some regularity. That is, they are not sticking to any regular grid, but they are not so adaptive to the terrain variance as in the greedy insertion and the edge collapse method. The only exception of this is the WLOP algorithm, which presents a more non-uniform distribution of the vertices of the TIN along the terrain.

However, there are other parameters to take into account when selecting the best method. On the one hand, the edge collapse method provides nice results, is fast to compute and presents low memory footprint while processing. However, the termination condition is based on the complexity of the mesh, not on the approximation quality. That is, we only impose a desired number of triangles in the final mesh, but we are not relating this to a measure of the error with respect to the original data at full resolution. On the other hand, the two steps process defined for point set simplification algorithms allowed us to use many different point set simplification methods in the state of the art. However, we can see that the results obtained are not optimal when compared to the greedy insertion and edge collapse methods. Finally, the greedy insertion method has a termination condition that is directly related to the approximation quality: the maximum height difference with respect to the original raster values.

Given the above mentioned reasons, we decided to render the EMODnet DTM released in 2018 using the greedy insertion method. Figure 6 shows some screenshots of the viewer using Cesium running in the web portal.



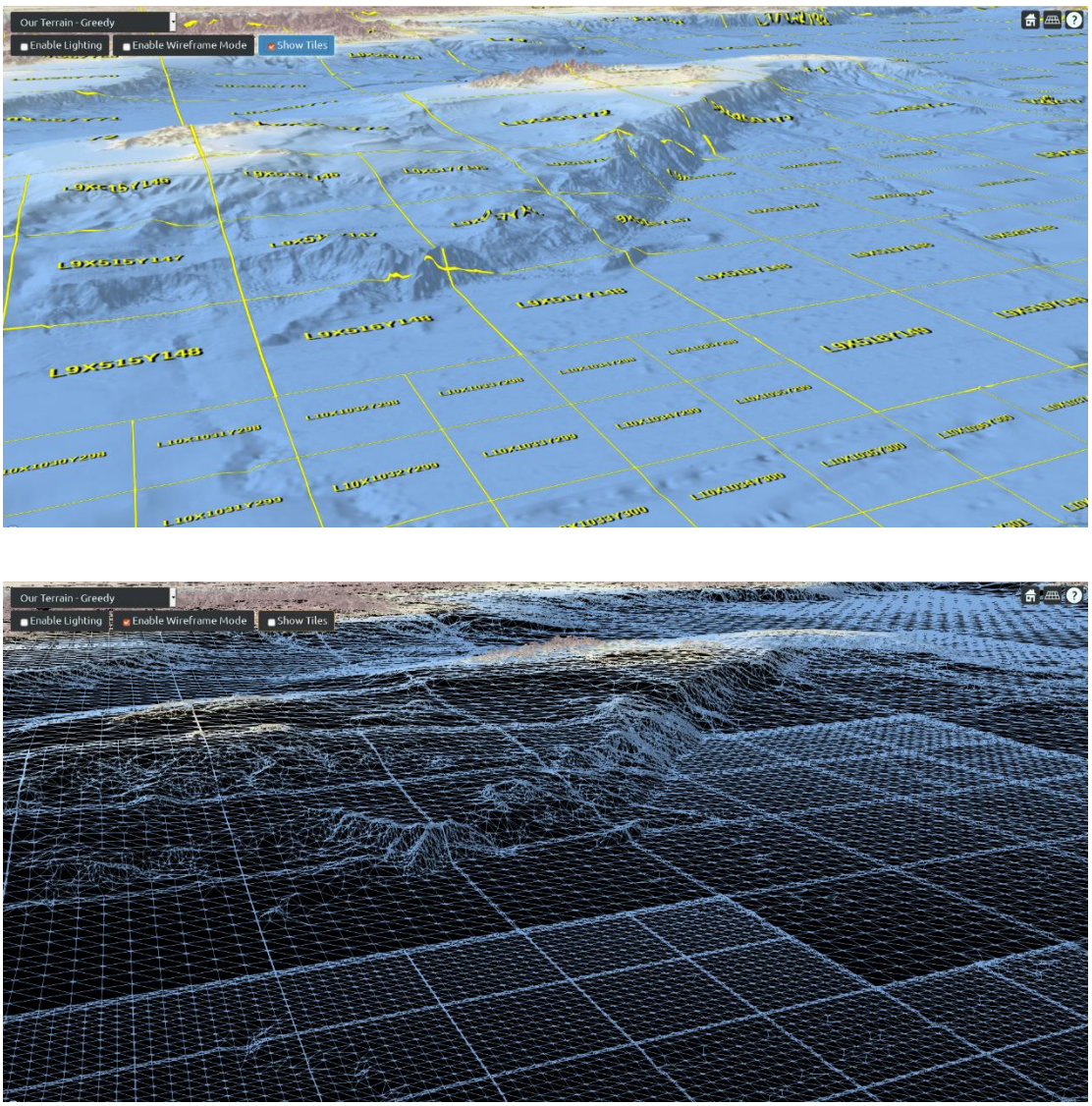


Figure 6. The top image shows an arbitrary slanted view of the EMODnet DTM. In the middle one, we highlight the tiles that we are actually rendering (L indicates zoom level, and X Y are the coordinates of the tile within the zoom). Finally, on the bottom image, we show the triangles forming each tile.

Conclusions

In this deliverable we described a completely viewer-independent manner of generating a hierarchical LOD structure of regular square tiles. We focused on creating static data, which is crack-free within a given zoom level, and applied several methods in the state of the art to tackle the TIN creation problem. After a review of the properties of each method, we rendered the full global DTM of EMODnet Bathymetry using the greedy insertion method. The results obtained can be visualized in the portal (http://portal.emodnet-bathymetry.eu/).

Finally, we believe that the results in this project may be useful for the community. For this reason, on the one hand, we published our code in github: https://github.com/coronis-computing/emodnet_qmgc, along with documentation and wiki pages providing all the information required to run the software. Moreover, we are preparing a journal article (to be submitted in the coming weeks to the *ISPRS Journal of Photogrammetry and Remote Sensing*) containing and extending the developments described in this document, as well as providing quantitative comparisons of the results provided by each method.

References

[CGAl-PointSetSimplification] P. Alliez, S. Giraudot, C. Jamin, F. Lafarge, Q. Mérigot, J. Meyron,L. Saboret, N. Salman, S. Wu, Point set processing, in: CGAL User and Reference Manual, 4.13 Edition, CGAL Editorial Board, 2018. URL https://doc.cgal.org/4.13/Manual/packages.html#PkgPointSetProcessingSummary

[CGAL-Simplification] F. Cacciola, Triangulated surface mesh simplication, in: CGAL User and Reference Manual, 4.13 Edition, CGAL Editorial Board, 2018. URL https://doc.cgal.org/4.13/Manual/packages.html#PkgSurfaceMeshSimplificationSummary

[Christen11] M. Christen, S. Nebiker, Large Scale Constraint Delaunay Triangulation for Virtual Globe Rendering, Springer Berlin Heidelberg, Berlin, Heidelberg, 535 2011, pp. 57-72.

[Dicken09] C. Dyken, M. Daehlen, T. Sevaldrud, Simultaneous curve simplification, Journal of Geographical Systems 11 (3) (2009) 273-289.

[Douglas73] D. H. Douglas, T. K. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, Cartographica: The International Journal for Geographic Information and Geovisualization 10 (2) (1973) 112-122.

[Fowler79] R. J. Fowler, J. J. Little, Automatic extraction of irregular network digital terrain models, SIGGRAPH Comput. Graph. 13 (2) (1979) 199-207.

[Garland95] M. Garland, P. S. Heckbert, Fast polygonal approximation of terrains and height fields, Technical report CMU-CS-95-181, Carnegie Mellon University (September 1995).

[Garland97] M. Garland, P. S. Heckbert, Surface simplification using quadric error metrics, in: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997, pp. 209-216.

[Huang09] H. Huang, D. Li, H. Zhang, U. Ascher, and D. Cohen-Or. Consolidation of unorganized point clouds for surface reconstruction. ACM Transactions on

Graphics, 28:176:1–176:78, 2009.

[Lindstrom98] P. Lindstrom, G. Turk, Fast and memory efficient polygonal simplification, in: Proceedings of the Conference on Visualization '98, VIS '98, IEEE Computer Society Press, Los Alamitos, CA, USA, 1998, pp. 279-286.

[Pauly02] Mark Pauly, Markus Gross, and Leif P Kobbelt. Efficient simplification of point-sampled surfaces. In Proceedings of the conference on Visualization'02, pages 163–170. IEEE Computer Society, 2002.

[Polis93] M. F. Polis, D. M. McKeown, Issues in iterative tin generation to support large scale simulations, in: Proceedings of Auto-Carto 11 (Eleventh International Symposium on Computer-Assisted Cartography), 1993, pp. 267-277.

[Puppo94] E. Puppo, L. Davis, D. D. Menthon, Y. A. Teng, Parallel terrain triangulation, International Journal of Geographical Information Systems 8 (2) (1994) 105-128.

[QMSpec] Cesium. Quantized-mesh-1.0 specification. https://github.com/AnalyticalGraphicsInc/quantized-mesh

[Ulrich02] T. Ulrich, Rendering massive terrains using chunked level of detail control, SIGGRAPH Course Notes, vol. 3, no. 5, 2002.