Building of the terrain model from the municipal-level GIS data

by Natalia Mirza, Alexey Skvortsov[1]

The terrain model is one of the basic data structures used in GIS. In this paper, a data structure is provided for the effective management methods pertaining to the terrain model that covers a huge territory. The developed algorithms are based on a special data structure, called a Multi-Triangulation (MT), which allows the manipulation of a surface at variable resolutions. By modifying of the original MT structure, it is possible to manipulate the data, which is larger than the available computer's RAM.

Introduction

The problem with manipulating a very large, 3d-model of open-air areas (examples: large-scale city models and extensive objects of industrial, civil, and transportation engineering; small-scale area models) is finding an application in various fields. Due to the increasing usage of laser scanners, there is a problem with processing the millions of measuring points. There is only one way to manage this huge amount of data. Regular raster grids provide an easy way to resolve the problem (e.g. works of Silva, 2001; Pajarola, 2002). However, sometimes it is impossible to use grids, not only because of the data redundancy, but also because of the source data for terrain construction. For GIS analysis, height fields are not enough. Structural lines are very important for a municipal-level GIS. In order to build the surface model of the road, it is very important to include the axis line, borders, edges etc. In such a case, a constrained triangulated irregular network (TIN) is used.

[1] Tomsk State University, Russia.

Modern algorithms focus mostly on the visualization speed and quality (e.g. Batched Multi-Triangulations by P. Cignoni, 2005). However, for GIS and CAD, the most important thing is to build the entire model for analysis. There is no effective approach to building a constrained TIN from data larger than the available RAM.

Preliminaries

To work with a TIN built on a massive data set, it is necessary to use high-performance hardware and special TIN simplification algorithms (e.g. works of C.T. Silva, 2001; E.Puppo, L. De Floriani, 1997). Using simplification algorithms means finding a compromise between the speed of terrain processing and the quality of the results. This problem is solved by using a *Multi-Triangulation* (MT) that can be loosely defined as a special data structure, representing a set of triangulation fragments, forming a Directed Acyclic Graph (DAG). Three-dimensional models of variable resolution based on an MT allow surface processing at the given level of detail. It means that model detailing (TIN triangles count) is set by some user-defined criteria. In this case, by selecting the appropriate criteria, one can create a model that will have high processing speed and high quality.

To provide a stronger MT definition, it is necessary to introduce some terms (described in works of M. De Berg, 1995; E. Puppo, L. De Floriani, 1997).

A *triangulation domain* is the portion of the plane covered by its triangles.

Let us consider *T* and *Ti* to be two triangulations, such that the domain of *Ti* is contained in *T*. In such a case, *T* and *Ti* are called *compatible*, if there is a triangles subset *T'* in *T*, which is a triangulation if it has the same domain as *Ti*. *Ti* is called a *fragment* (with respect to *T*) and *T'* is a *floor* of *Ti. Ti* is *minimally compatible* with *T* if there is no subset in *Ti* compatible over *T*.

A *local modification T* over *Ti* is an operation that replaces triangles of *T'* with triangles of *Ti* and denoted as $T \oplus Ti0$.

If every fragment *Ti* is compatible with *T*, than the sequence of fragments

$t = (T_0,...,T_n)$ is called a *compatible sequence* of triangulations.

As a result, a *Multi-Triangulation* is a DAG, having elements of compatible sequences of triangulations as nodes.

A *root* of the DAG is the most detailed (the coarsest) fragment, a *drain* of the DAG is the coarsest (the most detailed) fragment in a MT for building using simplification (refinement) algorithms.

An MT is represented by the following data structures:

1. A list of the MT triangles: Each triangle contains pointers to the three points composing it, pointer to the fragment containing it (*upper fragment*), and pointer to the fragment containing it in its floor (*lower fragment*).

2. A list of the MT fragments: Each fragment contains a list of pointers to the triangles composing it and a list of pointers to triangles composing its floor.

3. A list of the vertices: each vertex contains a necessary set of geometric (coordinates, normal) and attributive (color) data.

There are many algorithms based on an MT that allow for building a surface at variable resolution. One of them is implemented in the Microsoft DirectX API (Progressive Mesh by E. Puppo, 1998) and is widely distributed in various graphics systems based on DirectX.

However, all existing algorithms based on an MT can work only if the MT structure fits into computer's RAM. Very often, for some real-world problems, it is important to process a terrain that is larger than computer's available RAM.

In such a case, it is necessary to develop some algorithms that can manage very large terrains and permit fast visualization without visible quality drawbacks.

Modification of a Multi-Triangulation

To build an MT that allows the handling a very large amount of input data, it is necessary to locate the fragments in memory that are more likely used, i.e. fragments that will be included in the result triangles list. Such fragments potentially have to satisfy the user's extraction criteria. That is why the MT structure should allow loading only necessary fragments and unload unnecessary ones in real time.

The original MT has a very chaotic structure. This means that in the general, the detailing of one triangle may cause the need to detail a set of random triangles through the DAG; in this scenario, the full MT structure must be located in RAM. However, this is impossible within the described problem. In order to handle very large data volumes, change the structure of the original MT in such a way that would decrease the triangles' interdependency.

Splitting a Multi-Triangulation up into parts

This suggested approach provides the order of the MT structure; it depends on the idea of separating a Multi-Triangulation DAG into several independent subgraphs. This will enable one to handle each subgraph separately.

Input Data: A (constrained) triangulation.

Algorithm Structure:

1. The source triangulation is divided into rectangular parts.
2. A Multi-Triangulation is built inside each rectangle by the classical algorithm (M. De Berg, 1995; E. Puppo, L. De Floriani, 1997).
3. Residuary triangulation is simplified up to the coarsest fragment.

End of the Algorithm

As a result, one will get the MT structure as schematically shown in Figure 1.

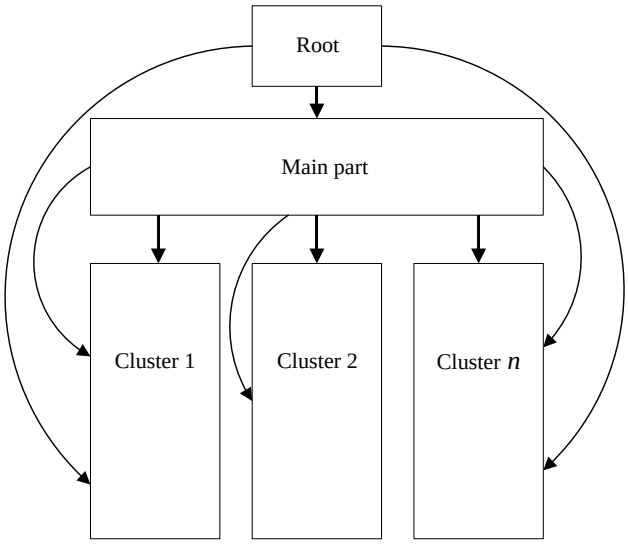

Fig. 1 - Multi-Triangulation structure

As one can see from this figure, there are several parts –*Clusters* and a *Main Part* (in fact Cluster 0), which can be loaded into the RAM independently.

A Multi-Triangulation that has structure as shown in Figure 1 is called *Clustered Multi-Triangulation (C-MT).*

Splitting an MT cluster up into blocks

M. De Berg, in his work, tries to decrease the DAG confusion by modifying the building algorithm. He suggests deleting only non-adjacent points at each step of simplification. As a result, the fragments of the current simplification step depend only on the fragments from the next steps as shown in Figure 2.

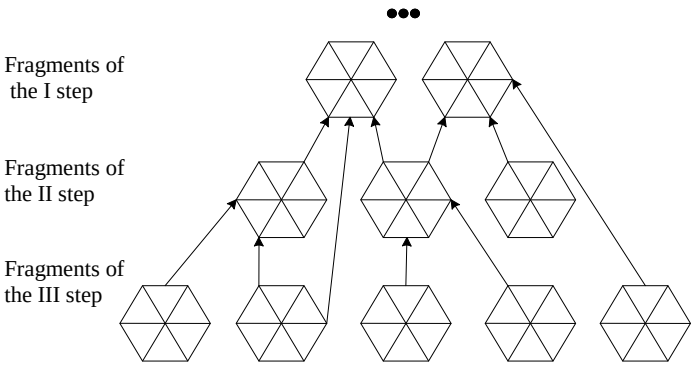

Fig. 2 – Fragments independency

We can join the fragments of each step into a *block*. As a result, the MT structure will be as shown in the Figure 3.

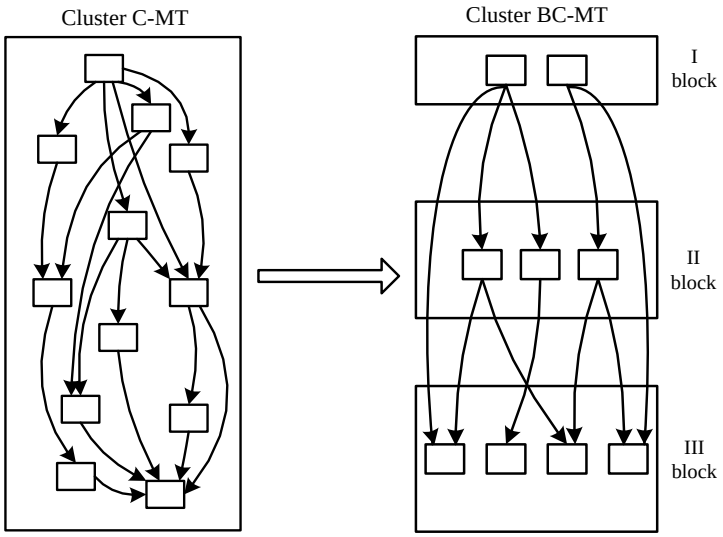

Fig. 3 – Blocks of a C-MT

A Clustered Multi-Triangulation with clusters divided into these blocks is called *Block-Clustered Multi-Triangulation (BC-MT).*

Calculation the number of the clusters

It is necessary to clarify into which rectangles source triangulation should be divided. The structure of the division directly influences the algorithm speed and correctness. The number of rectangles defines the number of MT trees: if this count is too small, then it is possible to get some regions with insufficient detail level, and if the number is too large, then a vast amount of memory will be required.

If the count of vertices in a C- MT is the same for every tree and for the main part, it is called a *balanced* MT.

Using a balanced C-MT, one can achieve a good visualization quality to size of located memory ratio.

The authors failed to obtain an analytical assessment of the specified number of clusters because of the uncertainty of the form of the data distribution. The experimental results show that the number of clusters for a balanced BC-MT is practically the same for different distributions. That is why we represent the results for a uniform distribution.

During the experiment, the authors constructed several BC-MTs and empirically established the number of clusters. Table 1 shows the dependence between the number of the clusters and the number of source points (*N*).

Tab. 1 – The number of the cluster and the number of the source points according to the experimental results

N, thousands	100	200	300	400	500	600	700	800	900	1000
ClusterCount	30	38	42	46	50	53	56	59	62	65

By the means of the least-squares method, the approximating function is received as follows:

$$CN(N) = \left\lceil \sqrt{\frac{N}{383}} - 1{,}866 + 13{,}732 \right\rceil \quad (1)$$

Figure 4 shows the difference between experimental results and the analytical function.

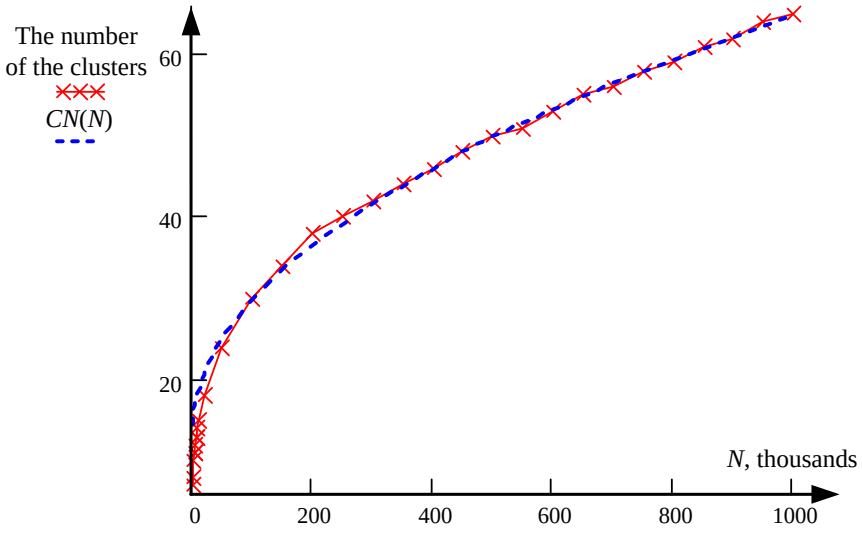

Fig. 4 – Approximating function CN(N) that returns the number of the cluster from the number of the source points

Building BC-MT within a RAM constraint

Based on experimental research, the number of triangles in an MT exceeds the number of the source triangles by a multiple of three.

This means that even if there is enough RAM for building the source triangulation, it may be insufficient for an MT.

To solve the problem, consider locating only the main part, and not more than one cluster of an MT, in RAM. After building a cluster, it should be saved into a file and unloaded from the RAM.

There is only one problem with this approach: when the main part needs to be built, all the clusters will be unloaded, and it will not be possible to set the links between the main part and the clusters. To avoid this situation, some temporal structures are used.

The *gateway* of the cluster is a fragment, consisting of the triangles of the cluster, which connect this cluster to the main part or the drain.

If one builds a gateway for every cluster, the structure of the C-MT will be as shown in Figure 5.

After including gateways in the structure, it is possible to unload the clusters without any consequences, because all necessary information for building the main part and drain is stored in the gateways.

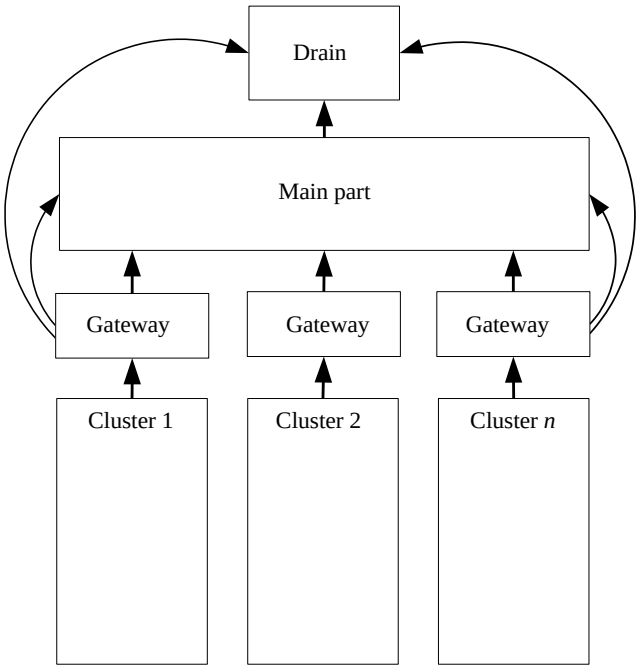

Fig. 5 – BC-MT structure with gateways

The algorithm of building BC-MT is as follows.

Algorithm of BC-MT building

Here is the algorithm that shows the using of the gateways to build an MT that is larger than available RAM.

Algorithm Structure:
Input Data: Triangulation.

1. According to the experimental results, the number of clusters in a BC-MT is calculated by eq. (1).
2. The set of source points is divided into rectangles in such a way that every rectangle that contains *CN(N)* points inside.
3. Within every rectangle, a cluster is built by the classical algorithm (by M. De Berg, 1995; E. Puppo, L. De Floriani, 1998). All the fragments of each simplification step form the block of cluster.
4. After building a cluster, it is necessary to create the gateway, consisting of the triangles that do not have links to the lower fragment.

The cluster is saved into a file and unloaded from memory.

5. Build the main part by the means of simplification of the triangulation, using the data remaining after cluster building.
6. Build the drain.
7. Each cluster is loaded to the memory and by the means of the gateway, the links between that cluster and main part are set.
8. Delete the gateway, and the cluster is saved and unloaded.

End of the Algorithm.

The complexity of the algorithm is $N^{1.5}$.

After building the BC-MT , it goes to the memory manager that manages its loading level in compliance with the extraction criteria.

Using a BC-MT for building of a terrain model from the municipal-level GIS data

In the previous sections, the modification of the Multi-Triangulation is considered. The source data for a BC-MT building is a triangulation that cannot be constructed from the huge data of the municipal-level GIS. The discussion now leads to the using of a BC-MT to solve that problem.

Common idea

Information:

We have 3D points and lines as the source data for terrain model building.

Algorithm Structure:

1. It is necessary to divide the source data in such a way so that it is possible to build a triangulation for every part.
2. For every triangulation, a BC-MT is built and saved in the file.
3. The list of BC-MTs goes to the memory manager.

End of the Algorithm.

Out-of-core memory control

At the beginning, only the roots of each BC-MT are loaded in the memory. The load level of other clusters depends on the extraction criteria. For instance, if MT clusters are rather close to the observer (or camera), then it is necessary to load them up to the maximum level of detail; the farther the cluster is from the observer, the lower the level of detail needed.

In the course of the extraction algorithm, each triangle is checked for criteria satisfaction. If a triangle does not satisfy the criteria, it should be refined by the triangles of a low fragment. The link to the low fragment causes the need to load the block in which it is stored. The block demands loading of all parent blocks. It is very important to manage the memory in a way that provides loading of BC-MT at maximum width. Otherwise, if BC-MT is loaded lengthwise, we may waste all the memory on detailing one small piece of the terrain.

Since a terrain model cannot be stored in the RAM, loading some blocks will require unloading other ones. The standard manner in this case is unloading those blocks that are unused for the longest time. However, MT is a DAG, so it is impossible to unload any block without possibly breaking the structure. Therefore, it is necessary to unload only terminal elements in the DAG.

Information:

The queue of blocks is for loading forms according to the extraction criteria. In the queue, every cluster is represented by only 1 block with a maximal number.

The memory control can be represented by 2 procedures, working asynchronously.

Memory managing

1. The first element from the queue is loaded only if the last loaded block was from another cluster. Otherwise, it will move to the next element. It is necessary to perform the loading of a BC-MT in width.
2. If memory is not sufficient for loading a block, it is necessary to unload some blocks. The blocks are chosen from the ones that were unused for the longest period, and the memory manager has a maximal number of the blocks. If the previous unloaded block was from the same cluster, then it is necessary to check the next one.

To make the algorithm clearer, it is necessary to give a simple example.

For example, a part of the territory of a city was scanned by the laser scanner. By the received data (i.e. points with X,Y,Z coordinates), it is possible to build a BC-MT. As one can see on the Figure 6, all of the points are divided into 4 parts. Then, according to these parts, 4 clusters of BC-MT are built. The extraction criterion in the example depends on the distance from the observer. Each cluster can be loaded independently; Figure 6 illustrates the level of data, which is loaded within each cluster.

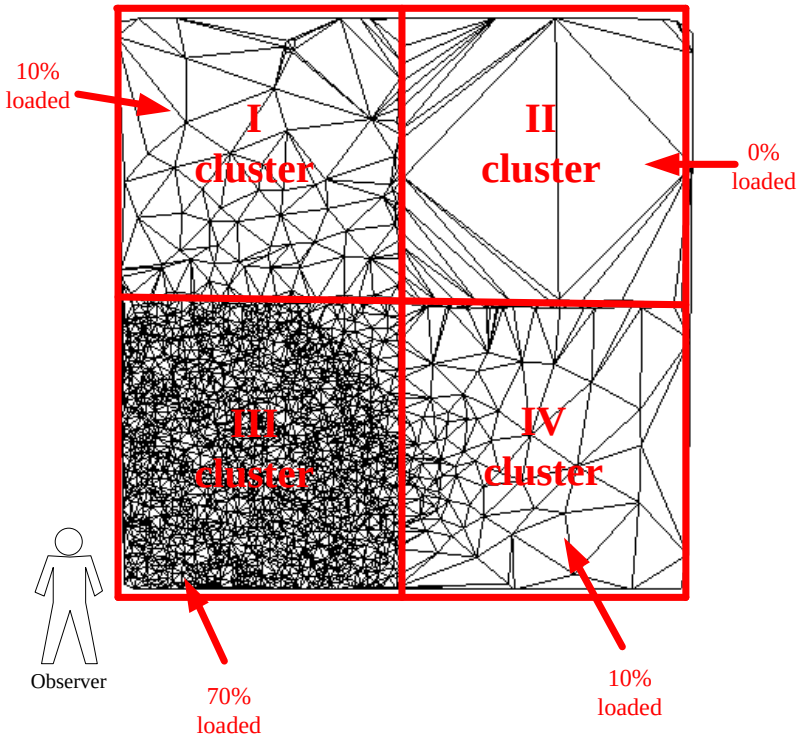

Fig. 6 – An example of BC-MT loading

Then, if observer moves, the level of detail changes in real time according to his new position. While changing the blocks, the closest clusters are loaded and the blocks of all the other clusters are unloaded.

Conclusion

The proposed algorithm can be used in the GIS and CAD systems for the terrain's analysis (Figure 7). It is very important for the city planning process as well.

In this article, algorithms are proposed for the building of very large terrain models based on a Multi-Triangulation, allowing the management of models that essentially exceed the size of the available computer's RAM. Experimental modelling of the developed algorithms shows high speed of visualization and acceptable size of occupied memory at the same time.

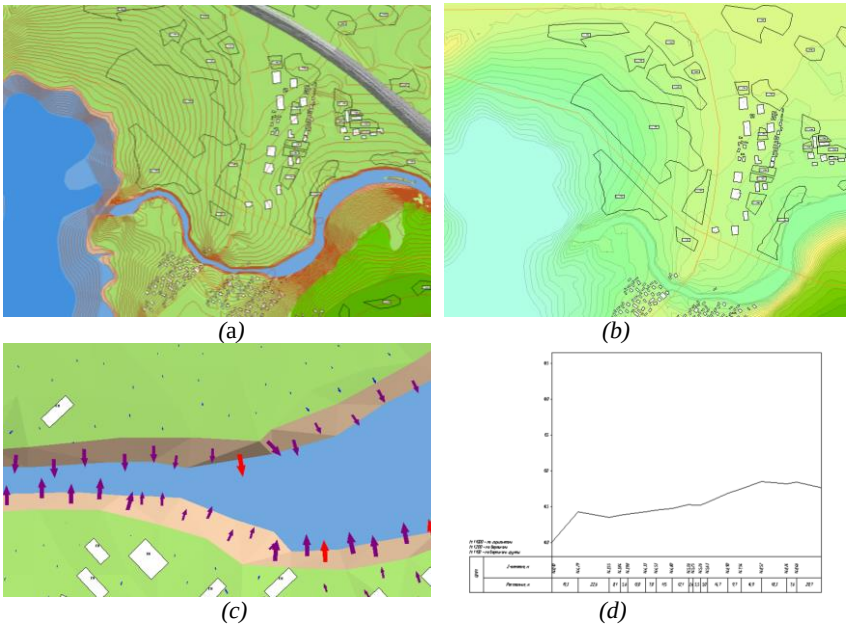

Fig.7 – An example of terrian's model analythis: a) – isolines; b) – isocontours; c) – slopes; d) – a section of the terrian.

References

Cignoni P., Ganovelli F., Gobetti E., Marton F., Ponchio F., Scopignio R. (2005), 'Batched multi triangulation', *Proceedings IEEE Visualization*, 8: 207-214.

De Berg M., Dobrindt K. (1995), 'On levels of detail in terrains', *Proceedings of the eleventh annual symposium on Computational geometry*, 95, 2: 426-427.

De Floriani L., Magillo P., Puppo E. (1997), 'Building and traversing a surface at variable resolution' *On Visualization*. 97, 6: 18-24.

Pajarola R. (2002), 'Overview of Quadtree based Terrain triangulation and Visualization', *Tech. Rep. UCI-ICS TR 02-01*, Department of Information, Computer Science University of California, Irvine.

Puppo E. (1998), 'Variable resolution triangulations', *Computational Geometry*, 18: 219–238.

Farias R., Silva. C. T. (2001), 'Out-Of-Core Rendering of Large, Unstructured Grids'. *Computer Graphics & Applications*, 10: 42–51.