A Coarse Grained Parallel Solution to Terrain Simplification *

M. Ghodsi
Computer Engineering Department
Sharif University of Technology
Tehran, Iran
`ghodsi@sharif.ac.ir`

J.-R. Sack
School of Computer Science
Carleton University
Ottawa, ONT, K1S 5B6
`sack@scs.carleton.ca`

Abstract

In this paper, we propose a coarse grained parallel solution for approximate simplification of a terrain given in Regular Square Grid (RSG) model. The simplified surface will be represented as a Triangulated Irregular Network (TIN). The algorithm partitions the input data into a constant, relatively small, number of pieces each of which is assigned to one processor. Processors Delaunay-triangulate their initial partitions separately and simplify them in parallel up to a given approximate error bound. Processors then follow a protocol to combine their results in this step and iterate until no more points can be removed.

1 Introduction

Simplification of graphical objects is a fundamental problem which arises in many applications from different fields such as GIS, computer graphics, and computer vision. This operation is designed to considerably reduce the amount of data needed to represent the object so that the approximated image preserves the object's shape and features up to a certain tolerance error with the goal that the required processing can be performed efficiently.

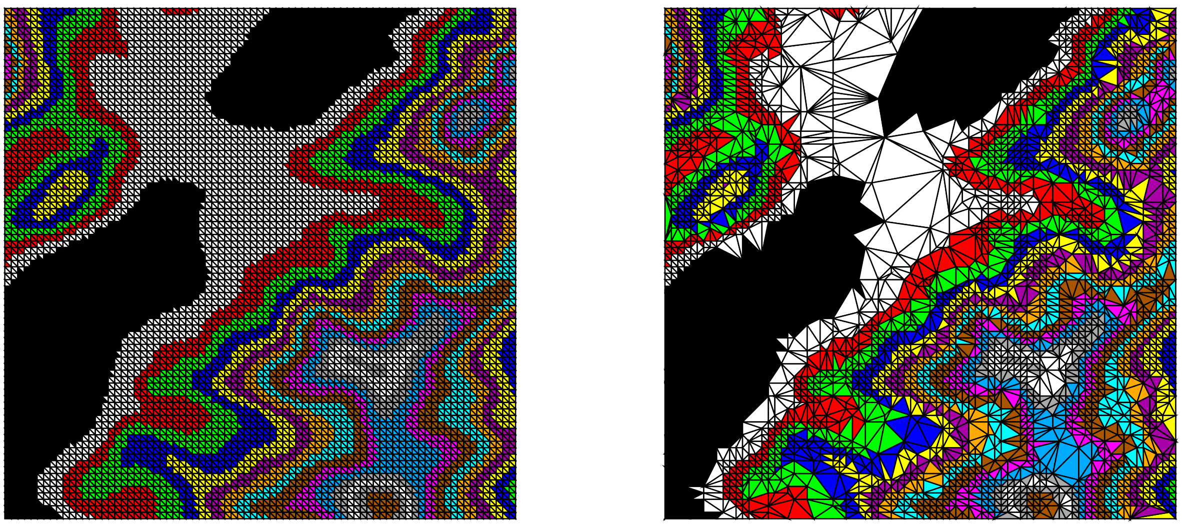
In this paper, we are interested in simplification of a natural terrain that is generally viewed as a $2\frac{1}{2}$-dimensional surface. Mathematically, this is expressed as a function $z = f(x, y)$ where z is the elevation of the surface at a point (x, y) of the Euclidean plane. In practice, this function is represented discretely based on the set of elevations of the points sampled from satellites images.

Two main digital terrain models are the Regular Square Grid (RSG), and the Triangulated Irregular Network (TIN). The RSG exhibits a highly regular topology and is easily constructed from the data obtained by satellites. It is also directly embeddable in parallel computers with fixed regular topologies. The TIN, on the other hand, exhibits an irregular topology and is a better model for representation of large natural terrains that are usually irregular. Although TINs are very suitable for representation of simplified terrains, they require complex techniques for parallel implementation and processing.

In this paper, we propose a parallel algorithm that receives a terrain as input given in RSG model and constructs a simplified TIN representation that approximates the original terrain to a given error tolerance.

Puppo *et al.* [7] proposed a fine grained parallel algorithm to convert a RSG data into an approximate TIN. Their algorithm is a parallel implementation of the early sequential greedy insertion technique proposed by Fowler and Little[2]. An initial approximation of the terrain with small number of points is constructed as a TIN. New points are added iteratively and the triangulation is updated, until the error is below the required threshold. In the parallel implementation there is one process for each triangle in the current triangulation, and one process for each point not yet inserted. Iteratively and in parallel, each triangle finds a point with the maximum error that lies inside it as a candidate for insertion. In a complex process involving substantial message passing, all candidates are inserted into the TIN in parallel and new processes are created dynamically for new triangles. The

*This research was supported by NSERC and Almerco Inc.



(a) (b)

Figure 1: (a) Initial triangulation of the terrain, (b) final simplified TIN.

algorithm was implemented on a CM-2 with 16K processors. Although good speedup has been reported, Garland and Heckbert showed that parallel candidate insertion produces poorer results than the sequential greedy insertion algorithm [3]. The algorithm is also not suitable for implementation on a system with a limited number of processors.

Our algorithm is coarse grained and is implementable even on workstations interconnected by a standard high speed network.

2 Overall Algorithm

The input terrain is given as $N \times N$ points in RSG format. Input data is partitioned into a constant number of pieces (P) and each piece is assigned to one process. The number of partitions are usually the same as the number of processors and this is assumed to be the case in the rest of this paper.

Each process triangulates its initial data and sets up a data structure for the resulting TIN so that different operations are performed efficiently and the TIN remains Delaunay triangulated. Each process also keeps track of the points on the border of its TIN and those that are shared by other processes. These processes are called TIN-processes and are numbered from 1 to P. There is one more process, numbered zero and called *coordinator* that makes global decision based on information it receives from the TIN-processes and sends coordinating messages to them.

There are three main steps in our algorithm that are repeated by all TIN-processors until a final level of overall simplification is reached. These steps are called: 1) *Simplification*, 2) *Shrink*, and 3) *Expand*.

Simplification step is performed by all processes on their TINs in parallel. A breadth-first variation of Lee's drop heuristic [5] is used to remove some points based on a given *error*. This step is repeated until no more point can be dropped. Clearly, simplification cannot be performed on the border points that are shared by other processes, because the TIN-process does not have information on some of the points in the star-shaped polygon surrounding the border point considered for removal. That is why no points on the shared border are dropped.

Shrink and Expand steps are designed so that removal of the points on a shared border is possible. In Shrink step, a TIN cuts one or more layers of its triangles on its border to an specified neighboring TIN and sends this information to the relevant receiving process. This step is involved and will be explained later.

Expand step is performed by a process that receives *shrink information* from a neighboring process. Upon receipt of this information, the process patches the points and triangles to its TIN, and updates its data structures that is needed for next simplification step in which points on the old border can now be dropped.

Figure 1(a) shows the input terrain partitioned in nine pieces and triangulated by nine processes. Figure 1(b) shows the final simplified TIN.

3 Data Structures Used

In our implementation, we have used data structures provided by LEDA [6]. `delaunay_triang` is the main class used for storing the TINs. Our algorithm needs several operations on TINs that are efficiently provided by this class. These operations are: insertion and deletion of points, point location, adjacent edges for each node, adjacent faces for each node, etc.

Each process keeps a priority queue Q for the points in its TIN except those that are on the shared border. There are points on the border of the initial un-partitioned TIN that are not shared by any other TINs. These points are included in Q. The priority of a point p in Q, which is called *fitness(p)* is defined as $|z_p - z'_p|$ where z_p it the elevation of p given as input and z'_p is the interpolated elevation of p assuming that it has been removed from the TIN. The value z'_p is computed as follows: A Delaunay triangulation of the star-shaped polygon R surrounding p which includes points that are connected directly to p, is constructed and z'_x is interpolated from the elevation of the vertices of the triangle in which p lies.

To facilitate updates in Q, a pointer is also set from each point in the TIN to its corresponding node in Q.

A circular link list is constructed for the points on the border of the TIN. The nodes on this list are ordered clockwise. For each node in this list, the process numbers that share this node is also saved. This structure facilitates identification of *external edges* that are parts of the Delaunay triangulation of the given points, but are not included in the triangulation of the complete un-partitioned TIN. This list also facilitates cutting and patching of border points that is performed in Shrink and Expand steps.

4 Simplification

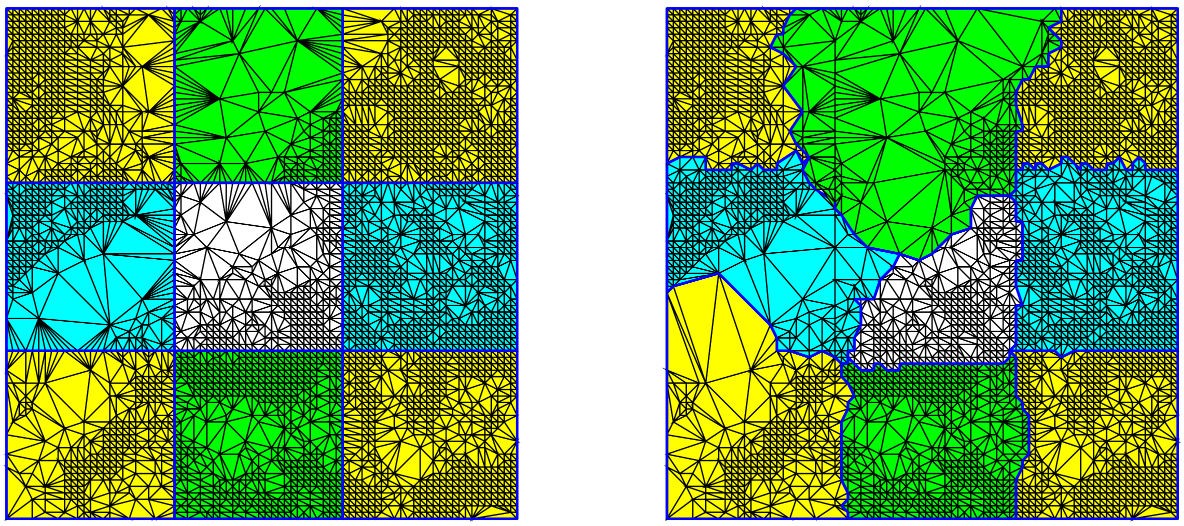
The following algorithm is performed by all TIN-processes in parallel for simplification of their TINs:

1. Repeat until there is no more point to remove
   (a) Find, `mis-lip`, a maximal independent set of *least important* points in the TIN:
      i. Repeat until Q is empty or the minimum priority of Q is less than *error*
         A. Remove the node u with the minimum priority from Q and append it to `mis-lip`.
         B. Remove all neighbors of u in the TIN from Q, if they exist.
   (b) For each node v in `mis_lip` do,
      i. Construct the star-shaped polygon R consisting of all neighbors of v,
      ii. Delete v and its attached edges from TIN,
      iii. Delaunay triangulate R,
      iv. For each node w in R, if v is not on shared border, find its new fitness value and update w in Q.

Each TIN-process sends a message to the coordinator at the end of the simplification step. The message includes the number of points removed, number of points left, and number of points shared by neighboring processes.

5 Shrinking and Expansion

Based on information that the coordinator receives from the simplification reports, it decides the order of Shrink/Expand steps that need to be performed by each TIN. This decision is made with three goals in mind: (1) all points on the current shared border should be transferred to one of the involved TINs so that they can be considered for removal in the next



(a) (b)

Figure 2: (a) TINs after first simplification, and (b) after first Shrink/Expand.

simplification phase, (2) the order of Shrink/Expand steps performed by the TINs should satisfy the serial constraints that exist between some of these operations, but it should allow parallel executions of these steps as much as possible, and (3) the load among processes after Shrink/Expand steps is balanced as much as possible. The latter is needed because the number dropped points usually differs largely from one TIN to another. Coordinator can dictate the Shrink/Expand steps in a way that the number of remaining points in the TINs in the next step is almost balanced.

Coordinator takes these issues into consideration and computes an ordered list of Shrink/Expand commands for each TIN and sends it to the relevant TIN process. Each TIN-process follows the commands it receives and performs the Shrink and Expand operations as requested.

A Shrink messages to process i tells i to shrink k layers for a neighboring process j. An Expand message is also sent to process j to expect shrink messages from process i and to expand towards i after receiving shrink information.

A process i that is to shrink k layers for a neighboring process j calculates three sets of points: S_1 containing the points that are currently shared by i and j, S_2 containing points to become the new border between i and j, and the third, possibly empty, set S_3 that includes other points that are transferred from i to j. Process i sends these sets to j in three messages, removes the points in S_1 and S_3 from its TIN, and updates its border information and its priority queue.

The points and triangles that process i removes from its TIN, may affect the border information of its other neighboring processes. Process i may not remain neighbor of process $l \neq j$ and l may become new neighbor of j. It is important that the border information is consistent in all TINs. Therefore, the shrinking process i sends special messages to those neighbors that need to update their border information.

Process j will be in Expand state waiting for messages from i. This is guaranteed by the algorithm executed by Coordinator. After receiving three messages from i, j inserts the points in S_2 and S_3, patches the triangles to its TIN and updates its border information accordingly. It can be shown that j triangulates these new points in exactly the same way that they were triangulated in i.

A process goes into a series of Shrink and Expand states dictated by the coordinator. Due to the serial nature of a Shrink and its relevant Expand, there are blocking periods for a process waiting to expand, but it is claimed that these steps are parallelized as much as possible in practice.

Figure 2(a) shows the TINs after the first simplification step, and figure 2(b) shows the TINs after processes are done with their Shrink/Expand steps and can start the next simplification step.

6 Implementation

The parallel algorithm is being implemented on a 16-node Pentium-based network of workstations. LEDA is used to provide different data structures and Message Passing Interface (MPI) [4] library is used to provide message passing primitives.

Early experimentation shows that the simplification steps are the most costly phase of the algorithm which is performed in parallel by all TIN-processes. It has also been observed that most of the points are removed in the first simplification step and the number of removed points is drastically reduced in the next steps. The algorithm usually ends after three or four simplifications steps and the resulting simplified TIN is very "similar" to the one resulting from the simplification algorithm performed on the overall TIN.

We have also observed that the amount of information sent by different messages is reasonably small, so we do not expect much overhead on the performance due to these messages. Therefore, we expect that our algorithm provides a good speed up in practice.

7 Conclusions

In this paper, we presented the overall view of a coarse grained parallel algorithm for simplification of a large terrain given in RSG model. The resulting terrain is represented in TIN. The initial data is partitioned among a small number of processes. Processes triangulate their data and perform a serial simplification based on Lee's drop heuristic in parallel to simplify their TINs. Based on a cut-and-patch strategy, the border points and their adjacent triangles are transferred among TINs so that they can be chosen for removal in the next simplification step. These steps are repeated until no more point can be removed.

Our experience with a simulation of this algorithm shows a good performance with reasonable message passing overhead. We believe that our parallel implementation will provide a good speedup.

Our algorithm can be a good basis for hierarchical multi-resolution representation of terrains [1]. It is also possible to modify this algorithm to compute approximate data-dependent TINs [8].

References

[1] L. De Floriani and E. Puppo. A hierarchical triangle-based model for terrain description. *LNCS*, (639):236–251, 1992.

[2] R.J. Fowler and J.J. Little. Automatic extraction of irregular network digital terrain models. *Computer Graphics*, 13(3):199–207, 1979.

[3] M. Garland and P. S. Heckbert. Fast polygonal approximation of terrains and height fields. Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[4] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, 1994.

[5] J. Lee. A drop heuristic conversion method for extracting irregular network for digital elevation models. In *GIS/LIS'89 Proc.*, pages 30–39, 1989.

[6] K. Mehlhorn and S. Näher. Leda, a platform for combinatorial and geometric computing. *Communications of the ACM*, 38(1):96–102, 1995.

[7] E. Puppo, L. Davis, D. De Menthon, and Y. A. Teng. Parallel terrain triangulation. *Int. J. Geographical Information Systems*, 8(2):105–128, 1994.

[8] S. Rippa. Adaptive approximation by piecewise linear polynomials on triangulations of subsets of scattered data. *SIAM Journal on Scientific and Statistic Computing*, 13(1):1123–1141, 1992.