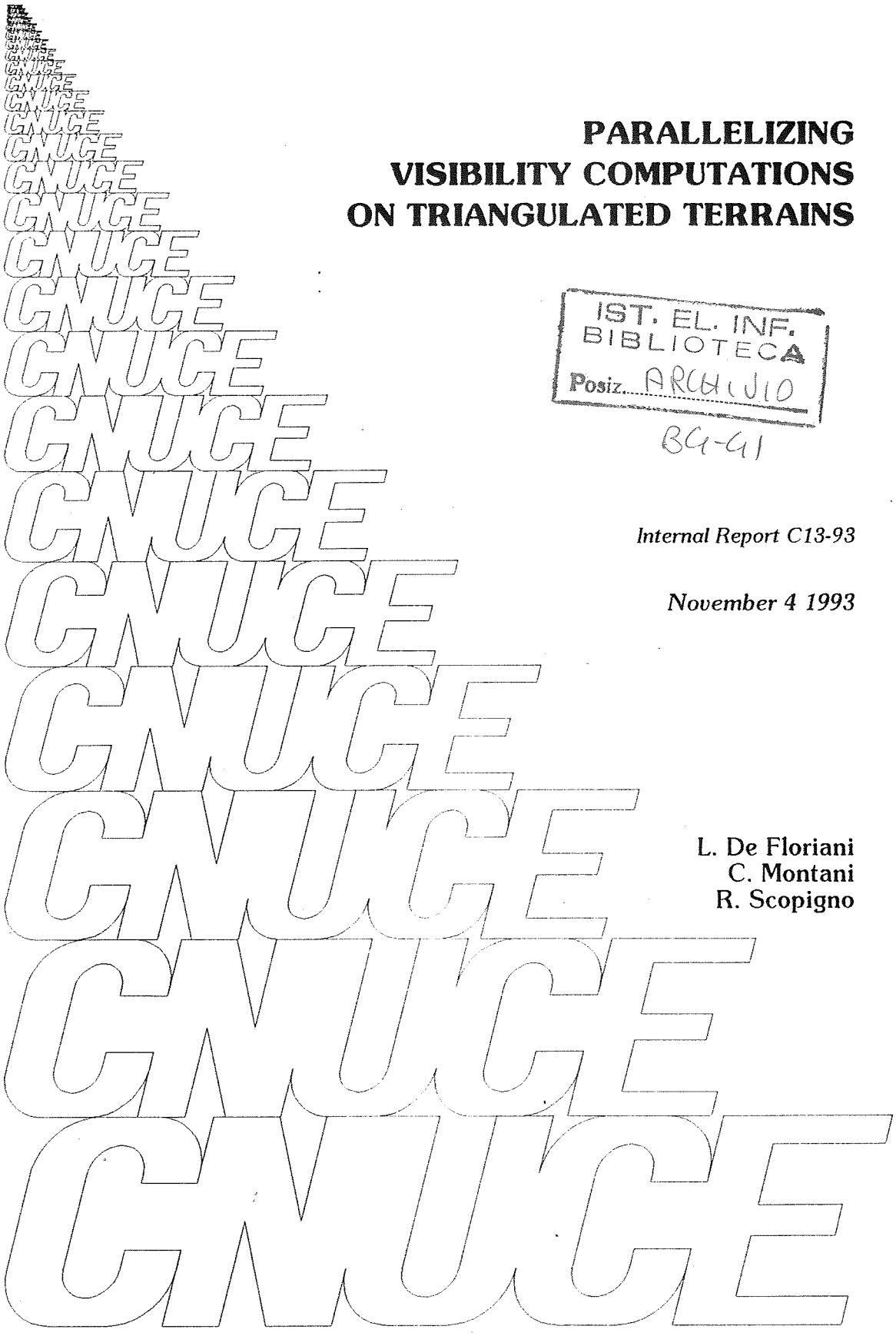

PARALLELIZING VISIBILITY COMPUTATIONS ON TRIANGULATED TERRAINS

Internal Report C13-93

November 4 1993

L. De Floriani
C. Montani
R. Scopigno

Parallelizing Visibility Computations on Triangulated Terrains

L. De Floriani[‡], C. Montani[†], R. Scopigno[⋆]

[‡] Dip. di Informatica e Scienze dell'Informazione - Universita' di Genova,

Viale Benedetto XV, 3, 16132 Genova, ITALY, E-mail: deflo@disi.unige.it

[†] I.E.I. – Consiglio Nazionale delle Ricerche, via S. Maria 46, 56126 Pisa, ITALY,

E-mail: montani@iei.pi.cnr.it

[⋆] CNUCE – Consiglio Nazionale delle Ricerche, via S. Maria 36, 56126 Pisa, ITALY

E-mail: scop@icnucevm.cnuce.cnr.it

November 12, 1993

Abstract

In this paper we address the problem of computing visibility information on digital terrain models in parallel. We propose a parallel algorithm for computing the visible region of an observation point located on the terrain. The algorithm is based on a sequential triangle–sorting visibility approach proposed in [4]. Static and dynamic parallelization strategies, both in terms of partitioning criteria and scheduling policies, are discussed. The different parallelization strategies are implemented on an MIMD multicomputer and evaluated through experimental results.

1 Introduction

The problem of computing visibility information on a terrain has several important applications. One application, which has been studied in great depth, is terrain visualization, which consists

of producing a two-dimensional image of the terrain. Apart from visualization, applications include the computation of a set of optimal observation points [18, 6], the location of transmitters and receivers for line-of-sight communication and surveillance [13, 14, 11], orientation and navigation [22] and the extraction of significant topographic features [7]. While image production is much faster using standard silicon-coded projective graphics, visibility information necessary for further analysis is required in object-space format (i.e., independent of the display device). In the literature, several object-space algorithms have been developed also for the visualization of a terrain [19, 16, 1, 15], which exploit the peculiar characteristic of the scene.

This paper outlines the development and experimentation of a parallel algorithm for computing the visibility model with respect to a discrete set of viewpoints located on the terrain surface. A terrain is described by a function $z = f(x, y)$ defined over a connected domain, and, in our application, approximated by a digital terrain model with triangular faces (called a Triangulated Irregular Network (TIN)). Two points are considered to be mutually visible when they can be joined by a straight-line segment lying above the terrain (and intersecting it only at its two extreme points). A visibility model of a terrain with respect to a set of viewpoints is a collection of visible regions, each of which is associated with a viewpoint. A *visible region* of a viewpoint is defined as the portion of the terrain visible from that point.

The problem of visualizing a terrain through an object-space approach has been studied in recent years in computational geometry, and interesting algorithms have been developed [19, 16, 1, 15]. With the exception of the one by Preparata and Vitter [16], all these algorithms are more theoretical (since they optimize asymptotic complexity) than practical. The algorithm by Preparata and Vitter uses a similar approach to the one proposed by Reif and Sen [19] with simpler data structures, while achieving the same worst-case time complexity. The time complexity of both algorithms depends on the number of vertices of the terrain model as well as on the size of the computed visible image.

Another approach to computing visibility on a terrain consists in determining the lower envelope of the set of triangles in a TIN [10, 2]. The lower envelope of a set $\mathcal{T}$ of triangles in the 3D space defines a partition of the x-y plane into maximal connected regions, each labeled with a triangle of $\mathcal{T}$ in such a way that, if a region R is labeled with triangle t of $\mathcal{T}$, then t is the triangle with the minimum height over R. In [4] an algorithm was proposed which computes the visible region of a viewpoint on a triangulated network by examining the triangles in a spiral order around the viewpoint and monitoring a growing horizon at each step. The algorithm is suboptimal and works for TINs based on the Delaunay triangulation [17]. On the other hand, it

is conceptually simple and efficient, as shown by the results of our implementation (see Section 2).

The requirement for efficiency of visibility algorithms is related to the increasing use of visibility computation in a number of GIS applications. High performances are required, for istance, in line-of-sight problems or in navigation, where the visibility model of a terrain must be updated in real time.

Several proposals have been published regarding parallel visibility computations in the field of computer graphics (i.e., for computing the visible parts of the solid objects contained in a scene, from a viewpoint lying either within or outside to the bounding volume of the scene). Among the more recent proposals, not covered in the tutorial paper by Whitman and Parent [25], there are the object–space solutions proposed by Franklin et al. [9], Theoharis [23], and Scopigno et al. [20].

To authors' knowledge, there has only been one proposal for a parallel algorithm for terrain visibility, i.e. the one by Reif and Sen [19]. This algorithm is not a direct parallelization of the sequential algorithm developed by the same authors, but it embeds some key ideas. The time complexity complexity of such algorithm depends on the size of the output scene, and it employs a model of parallel computation slightly different from the conventional PRAM (Parallel Random Access Machine) model [24]. The algorithm is thus more of theoretical interest than of practical interest.

In this paper we propose a parallel solution to visibility computation on an MIMD architecture based on a parallelization of the sequential algorithm described in [4]. This algorithm can simply be parallelized by partitioning the input data set. Thus, we propose different partitioning criteria for the input data based both on a static and on a dynamic approach. In the design of our parallel solution, our main goal was to get maximal portability of the solution. This was achieved by reducing (and possibly avoiding) interprocess communication so as to design a parallel algorithm which could efficiently run on a distributed-memory MIMD parallel machine as well as on networks of workstations running under a distributed computing environment.

The remainder of the paper is organized as follows. Section 2 describes the sequential approach to visibility computation, on which our parallel algorithm is based; its implementation, developed by the authors, is also discussed. Section 3 describes the various parallelization strategies, the parallel implementation, and scheduling and local balancing issues. Section 4 is devoted to the presentation, discussion and comparison of experimental results.

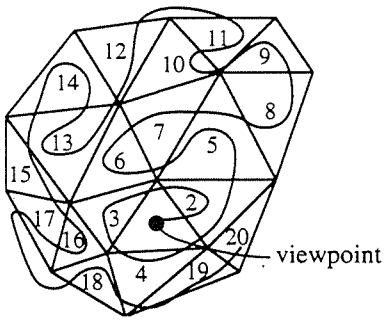

Figure 1: One of the possible radial orders (the approximated spiral visiting path is shown).

2 A Sequential Visibility Algorithm

In this Section, we propose a sequential algorithm for the computation of the visible region of a viewpoint on acyclic TINs. More precisely, given an acyclic TIN $\mathcal{D} \equiv (\Sigma, \mathcal{F})$, and a point of view V on $\mathcal{D}$, the algorithm determines the visible portion of each face of $\mathcal{D}$, i.e., for every triangle t_i of Σ, the subset made of polygonal portions of t_i such that each of these portions is completely visible from V.

This solution, called *Horizon-Cut*, is an optimized and revised version of the terrain visibility algorithm proposed by De Floriani et al. [4], developed in order to reduce its *practical* computational complexity, by simplifying its kernel processing phase.
Horizon-Cut algorithm operates in two steps:

- sorting of the triangles of Σ by increasing distance with respect to the projection $\bar{V}$ of the viewpoint V on the $x - y$ plane (*radial sorting phase*);
- computation of the visible portions of each triangle of $\mathcal{D}$ with respect to V (*visibility computation phase*).

2.1 Radial Sort

Radial sort is performed by building a *star-shaped* [1] polygon Π around $\bar{V}$ by incrementally adding one triangle at a time to an initial polygon formed by the triangles of Σ incident on $\bar{V}$. The acyclicity property of Σ ensures that at each step we can always add a new triangle, while maintaining the star shape of the resulting polygon [5]. The triangulation Σ is therefore

[1] A polygon Π is said to be *star shaped* with respect to a point $\bar{V} \in \Pi$ **if** $\bar{V} \in$ *kernel*(Π) [17].

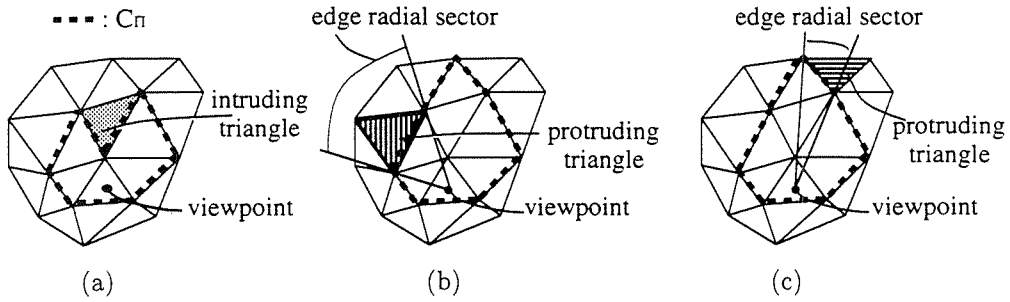

Figure 2: An example of intruding triangle, (a), and examples of two protruding triangles: in (b) star shape is mantained, in (c) star shape is not mantained.

ordered by an approximated spiral visit which covers all of the triangles from the viewpoint to the triangulation boundary (Figure 1).

The triangulation Σ is encoded using a triangle–based representation scheme [12]: for each triangle t of Σ, the three vertices and the three edge–adjacent triangles are stored.

Initially, Π consists of the union of the triangles of Σ containing $\bar{V}$ inside or on the boundary. At a generic step, an edge l on the boundary of Π is chosen, and the triangle t adjacent outwards to l is examined. Two situations may arise :

- if t is adjacent to Π along two edges (i.e., it is called a **intruding** triangle), it can be selected as the next triangle in the radial sorted list (Figure 2.a); it is therefore added to Π and to the sorted triangle list T_{sort};
- if t is adjacent to Π along a single edge ($\equiv l$) (i.e. it is called a **protruding** triangle), it can be added to Π iff the opposite vertex of t lies in the radial sector defined by $\bar{V}$ and l (Figure 2.b); otherwise, t is not added to Π because it does not maintain the star–shape of Π (Figure 2.c); t will be examined again later.

In our implementation, we keep the boundary C_Π of polygon Π. After examining an edge l of C_Π, we delete l from C_Π, whether the corresponding triangle t has been added to Π or not. In this way, a rejected protruding triangle t will be examined again only when another edge l' of t has been included in C_Π, i.e., when it is possible to add t to Π. Every triangle t of Σ is thus examined at most twice, and then the radial sorting operates in linear time in the number of triangles in Σ.

The pseudocode description of the radial sorting algorithm is reported below.

```
Algorithm RADIAL SORT
    input
        Σ: acyclic triangulation with respect to point V̄:
        V̄: viewpoint, contained in the domain of Σ;
    output
        T_sort: ordered list of triangles;
begin
    {Initialization:}
        case position of V̄ in Σ:
            V̄ vertex of Σ:
                T_sort ← the triangles incident in V̄, in any order;
                C_Π ← the edges not incident in V̄ of the above triangles;
            V̄ on an edge e of Σ:
                T_sort ← the two triangles adjacent to e, in any order;
                C_Π ← the four edges ≠ e of the above triangles;
            V̄ inside a triangle t of Σ:
                T_sort ← {t};
                C_Π ← the three edges of t;
    {Main loop:}
        while there are triangles not included in T_sort do
            l ← an edge in C_Π;
            t ← the triangle not in T_sort adjacent to l;
            if Intruding(t) or (Extruding(t) and StarShape(C_Π,t))
                then
                    concatenate t to T_sort;
                    delete from C_Π the edges of t that are in C_Π; {among them l}
                    insert in C_Π the edges of t not yet considered;
                else delete l from C_Π;
end.
```

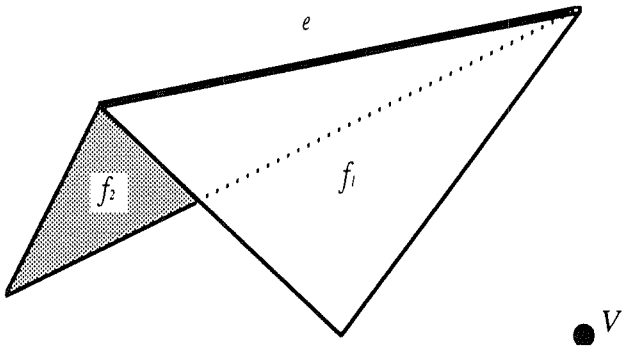

Figure 3: A blocking edge e with respect to viewpoint V: triangle f_1 is face–up, triangle f_2 is face–down.

2.2 Visibility Computation

The visibility computation step computes successive horizons incrementally, each horizon being restricted to the set of edges of the terrain examined so far. The horizon sequence is determined by examining the triangular faces of the model according to the ordered sequence produced by the radial sorting algorithm.

To describe the second step in more detail we need to introduce some definitions.
A face f_i of $\mathcal{D}$ is said to be **face up** with respect to a viewpoint V if its external side is oriented towards V; conversely, f_i is called **face down** (Figure 3).
We say that an edge e of $\mathcal{D}$ is a **blocking edge** when, if f_1 denotes the closest face incident in e and f_2 the furthest, then f_1 is face up and f_2 is face down. Figure 3 shows an example of a blocking edge.

The visibility algorithm constructs an active circular list of blocking edges, called *Active Blocking Edge Segment Sequence* (**ABESS**). ABESS contains all those portions of blocking edges belonging to triangles already visited which can cast a shadow on triangles not yet examined (see Figure 4). At the beginning of the computation, ABESS is initialized to an empty list of intervals. Then the triangles of Σ are processed one at a time, following the radial order, in order to compute their visible regions. Processing the current triangle t involves two kinds of operations:

- determining the visible portions of t: if t is face down with respect to V, then t is totally invisible. Otherwise, we consider all segments in ABESS which will project onto t (called the *relevant segments* of ABESS for t), given the observation point V. For example, in

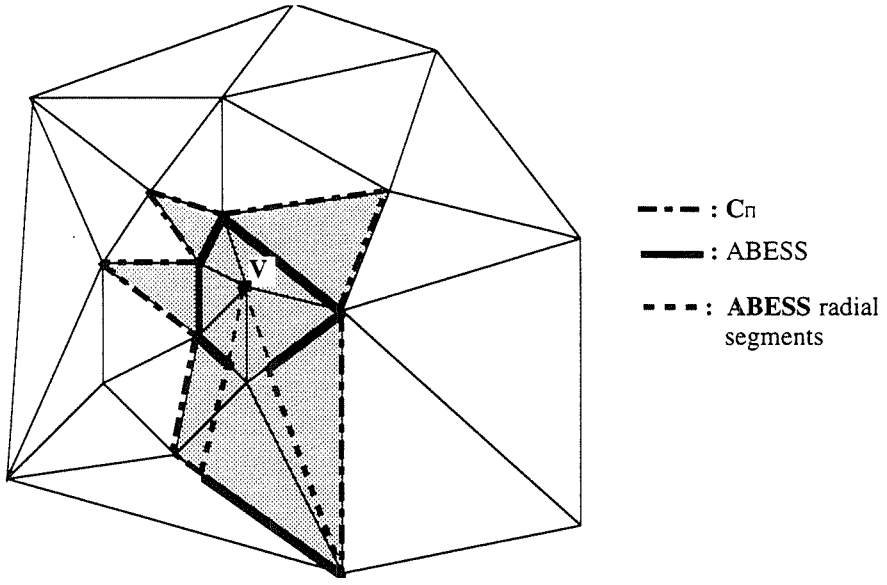

Figure 4: Configuration of ABESS and C_Π at an intermediate step of the algorithm (the viewpoint is the marked vertex; the triangles in Π are shaded).

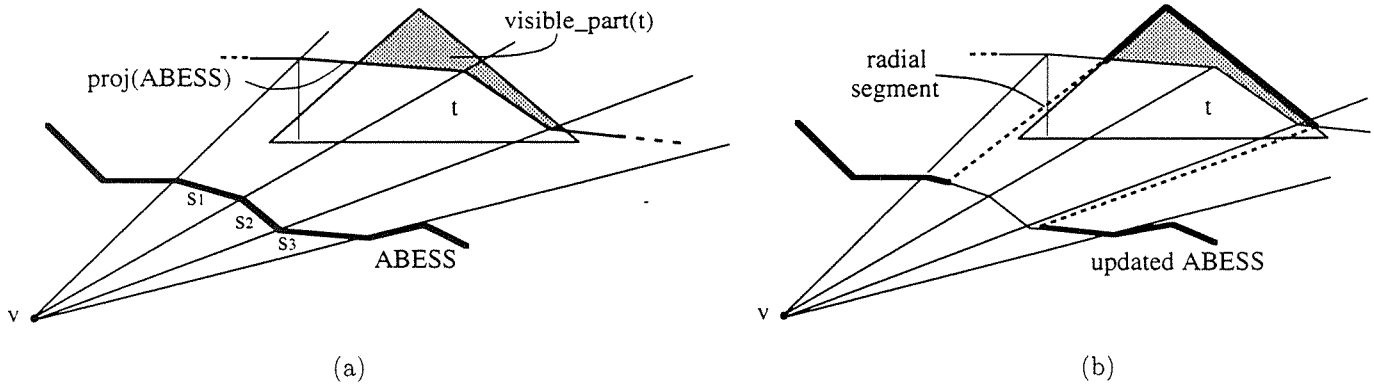

Figure 5: Clipping the visible part of the current triangle t (part above the projection of the ABESS horizon); ABESS is updated by replacing part of it with part of the upper edges of t plus two radial segments, inserted to maintain ABESS connectivity (drawn hatched).

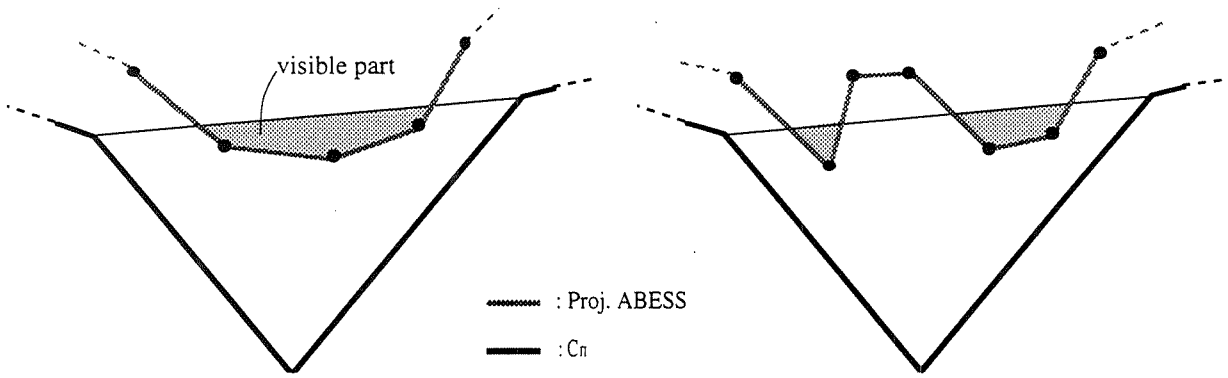

Figure 6: Projection of ABESS on the plane of an *intruding* triangle and visible part cutting.

Figure 5.a, the relevant segments for triangle t are s_1, s_2 and s_3. Then we compute the visible portion/s of t by cutting those parts of t which are above ABESS.

- updating ABESS: all (or part) of ABESS segments which project into t are replaced with the corresponding upper edge/s, or section of edges, of t (Figure 5.b).

The triangle–ABESS cut therefore requires a polyline–triangle clip, with the simplification that heuristics can be devised to reduce the number of tests by taking into account the specific characteristic of the problem.

Horizon-Cut performs the triangle–ABESS cut through a restricted number of simple tests. The core operation is the projection of each ABESS vertex on the plane of t, and an internal/external test with respect to t. In order to take into account the specific structure of the problem and to produce an efficient solution, a number of special cases must be correctly managed. Figures 6 and 7 show different spatial dispositions of *active edges* of t, of the projected ABESS and of the triangle visible parts. Note that the ABESS projection can only intersect *active edges*, i.e., the edges of t which are not part of the current C_Π. This is a significant simplification, because in the case of an *intruding* triangle only one edge is compared with each ABESS segment, and only two are compared for *protruding* triangles.

For each triangle, the portions of active edges which are above the current ABESS are computed as a by–product of the triangle–ABESS cut, and inserted into the updated ABESS. Note that ABESS *fragmentation* tends to increase; in the case of a protruding triangle, for example, a section of an ABESS segment may be replaced by two edge segments (Figure 5.b). ABESS fragmentation is a serious problem, as the computational complexity of the algorithm is linearly dependent on the number of segments in ABESS.

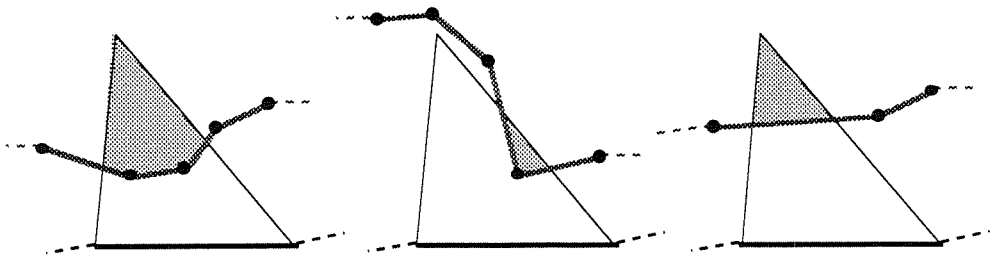

Figure 7: Projection of ABESS on the plane of a *protruding* triangle and visible part cutting.

The pseudocode description of the *Horizon-Cut* algorithm is reported below.

```
Algorithm HORIZON-CUT
    input
        D = (Σ, F): acyclic TIN model with respect to V, where Σ = (V, E, T);
        V: viewpoint;
    output
        {VR_i | VR_i ⊆ t_i, t_i ∈ T}: family of plane visible regions;
begin
    {Radial sort:}
        T_sort ← RADIAL SORT (Σ,V̄);
    {Visibility determination:}
        ABESS ← empty list;
        {Main loop:}
        while T_sort ≠ empty list do
            extract the next triangle t_i from T_sort;
            if t_i is face down then VR_i ← ∅; {t_i is totally invisible}
            else
                VR_i ← portion/s of t_i above ABESS;
                for every active edge l of t_i do
                    update ABESS by inserting sections of l above ABESS;
                    update ABESS by removing ABESS sections below l;
end.
```

For the sake of simplicity, the algorithm has been described by specifying the two phases, radial sorting and visibility computation, as two separate steps. In the actual implementation

these two activities are exploited at once: as soon as a triangle is selected by the radial sorter, it is immediately tested for visibility.

2.3 Evaluation of the Sequential Implementation

The overall structure of *Horizon-Cut* is similar to the algorithm originally proposed by De Floriani et al. [4]. The main modification is the implementation of the ABESS–triangle comparison: for each current triangle t_i a single *ABESS–triangle cut* is evaluated instead of many intersections with the shadow–polygons cast by the ABESS segments which project on t_i, as required by the original proposal. This produces:

- a much lower fragmentation of both ABESS and results (e.g. the number of visible regions);
- an effective speedup of the algorithm.

Fragmentation reduction

A by–product of computing the whole visible part at once is that ABESS updates do not cause the excessive fragmentation which characterizes the original algorithm. *Horizon-Cut* produces *at most* one new ABESS segment for each active edge – ABESS intersection point (clearly, the number of such intersections is lower than the ABESS segments which project onto the interior of t). *Horizon-Cut* therefore does not require searching and eliminating collinear segments after each ABESS update, while such search is mandatory in the original algorithm [4].

The results are less fragmented because, instead of having a visible region for each projected ABESS segment, we return a number of visible regions proportional to the number of intersection points between the active edges of t and the current ABESS (a single region as in Figure 6.a, or multiple regions as in Figure 6.b).

Speedup

The radial sorting step has a worst–case time complexity linear in the number of triangles in Σ; the visibility computation step has an $O(n^2\alpha(n))$ worst-case time complexity, since $O(n\alpha(n))$ is the size of ABESS in the worst case[2], and $O(n)$ triangles are examined. The size of ABESS is at worst $O(n\alpha(n))$ only if there are no colinear segments in ABESS; this complexity is achieved

[2]Where $\alpha(n)$ is the inverse of the Ackermann function [26].

Σ (no.tria.)	21K	41K	61K	80K	165K	236K
time (sec.)	11.54	20.68	30.51	36.48	74.13	107.47

Table 1: Computing times of *Horizon-Cut* algorithm on one *pe* and over different resolution datasets (average times over 100 runs with different viewpoints).

only through the reduction of ABESS fragmentation produced by our revised algorithm.
Apart from the effect on the worst case complexity, the simplification of the kernel phase (ABESS–triangle intersection) reduces the processing cost of a crucial phase, instanziated into the overall cycle of the algorithm, which requires a number of different geometrical tests and handling of many special cases.
The times of the sequential implementation over a number of tests confirm the practical effectiveness of our approach (Table 1). A comparison with the times of the original implementation of the algorithm [4], around 300 sec. for 5K vertices TIN on a Sun 3/60 workstation [3 Mips CPU], shows a remarkable increase in performance while processing similar dataset: *Horizon-Cut* times are reduced to around 6 sec. on a 7.5 Mips, 2.4 MFLOPS CPU (a single node of the multiprocessor computer described afterwards).

3 Parallelization of Visibility Computations

The strategies for splitting a sequential computation into parallel streams can be subdivided into **data partitioning** techniques, where the problem is independently solved on each partition of input/output data (also known as *geometric parallelism*), and **functional partitioning** techniques, where the overall algorithm is rewritten in terms of parallel threads and/or of a pipeline of sub–phases [8]. *Geometric parallelism* solutions are common in Computer Graphics and they are either based on the partition of the input dataset or of the output space. In both cases the same code is applied on a number of processing elements (*pe*) independently and the solution is obtained as the union of local solutions. Classical examples of output space partition are distributed ray tracing, where pixel values can be independently computed, and image–space visibility algorithms, where different scan lines can be computed on different processing nodes independently.

A parallel visibility solution based on geometric parallelism is proposed here. *Horizon-Cut* algorithm can be simply parallelized by partitioning the input dataset, but the partition must guarantee the feasibility of independent visibility computations. This property is maintained by

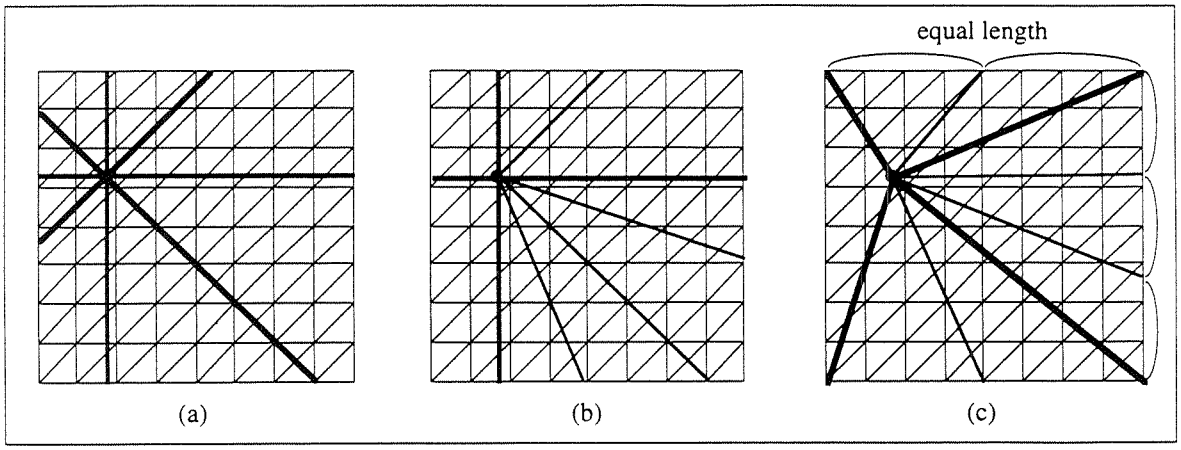

Figure 8: On the left, partition in *equal angle* sectors; in the center, *quadrant* based partition; on the right, *equal area* partition.

partitioning the input dataset Σ into *sectors*, having their common vertex at the viewpoint: all the triangles that may partially cover a triangle t are contained in the same sector (apart from the triangles on the sector borders, see Figure 8). Visibility can be computed on each sector independently, because all the triangles which are needed to solve the visibility of a triangle t are contained within the sector.

In the design of parallel *Horizon-Cut* one main requirement has been highlighted: *maximal portability* of the solution. A coarse–grain multi–process software architecture directly derives from the characteristics of the sequential algorithm; while in designing an implementation for an MIMD architecture, our goal was to reduce (and, if possible, to avoid) inter–process communication, in order to produce a parallel algorithm which could be executed efficiently both on distributed–memory MIMD parallel machines and on networks of workstation running under a distributed computing environment, such as PVM [21] or Linda [3]. Due to high instability and rapid technological advances in parallel processing, portability is considered by the authors at least as important as efficiency and scalability.

3.1 Data Partitioning Criteria

A straightforward sector partitioning can be performed by simply dividing the triangulation into m **equal–angle** sectors (see Figure 8). This equal–angle solution is simple, but generates uneven partitions when the viewpoint is not located close to the center of the bounding rectangle.

Balanced partitions (in terms of the number of contained triangles), with sufficiently good chance, require similar visibility computing time and therefore allow balanced loads. But exact equal–sized partitioning criteria are not feasible, because they would require knowledge of the actual geometry of the triangulated terrain and thus partitioning costs would be too high. However, "approximated" equal–sized criteria can be devised as proposed below.

Figure 8.b depicts a second criterion, the **quadrant** partition, based on a two–step subdivision process. The area is first subdivided into four quadrants, with center at the viewpoint; then the ratio r_i between the area of each quadrant and the bounding rectangle area is computed. Each quadrant is then divided into $m * r_i$ equal–angle sectors. This criterion gives a better subdivision than the former one, but partitions are still too unbalanced in each quadrant when the viewpoint is situated in a non–central position.

A third partitioning criterion (see Figure 8.c), called **equal area** partition, is an attempt to solve this problem by applying a different subdivision rule. The bounding rectangle is divided into four sectors by the four lines which connect the viewpoint to the region vertices. Each sector is again subdivided into $m * r_i$ subsectors, but now the subsectors are selected in such a way that all their *non–radial edges*[3] are the same length. This partition guarantees a sufficiently good balanced space subdivision and requires only the computation of a simple function over the current viewpoint location.

Effective processing load not only depends on the characteristic of the partition, but also on the geometric properties of the triangulated surface and on the position of the viewpoint. In fact a partitioning criterion, which subdivides the triangulation into equal–sized triangle subsets, may also result in unbalanced loads. This fact, along with higher processing costs, means that exact partitioning criteria are not practical.

All criteria described above are *static*, i.e., the extents of each sector are computed at the beginning of the computation. Another possibility is to apply an *adaptive dynamic partitioning*, where computation starts by using an initial simple partition (for example, into four quadrants). The subdivision is then refined at run time, when the load of the current processing element becomes too heavy. The effective load of *Horizon-Cut* algorithm can be approximately measured by the current length of ABESS (in terms of number of segments). A distributed dynamic partitioning criterion can therefore monitor ABESS length on each running visibility process p_1. As soon as the maximum value is reached, the computation is split by generating a new

[3] A *non–radial* edge is the intersection segment between the sector edges and the bounding rectangle of the triangulation on the x–y plane

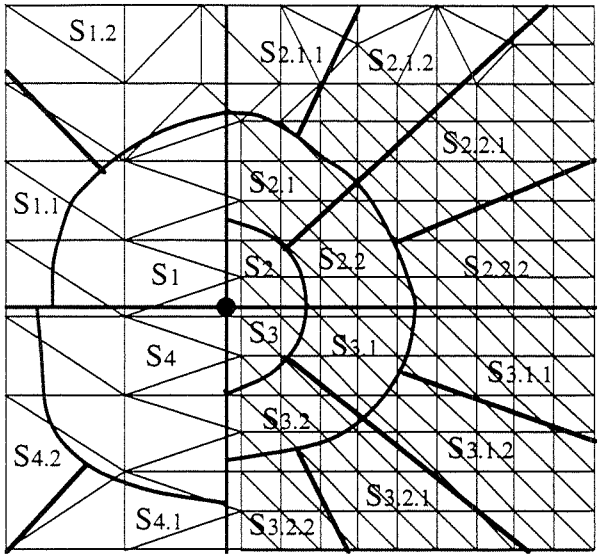

Figure 9: Adaptive dynamic partitioning.

process p_2: the current sector s is divided into two halves, s_1 and s_2, as well as the current ABESS and C_Π. Then, the computation continues with process p_1, by computing visibility on s_1 following ABESS_1 and C_{Π_1}, and process p_2 by computing visibility on s_2 following ABESS_2 and C_{Π_2} (Figure 9).

This strategy aims at load balancing optimization and a criterion such as ABESS complexity could be more sound than the equal area partition of the former static strategy, though at the cost of a much more complex implementation, processing overheads and lower portability. To be efficiently implemented, this approach entails dynamical process generation and allocation, or the availability of a *master* process which manages the splitting requests of *worker* processes. Moreover, efficient inter–process communications are required, in order to allow fast transmissions of current C_Π and ABESS to the split processes. Finally, the subdivision of dataset and data structures (S_i, C_{Π_i} and $ABESS_i$) is required at run time for each split, and this originates an overhead which will reduce the overall performance.
We thus decided to concentrate on a more simpler and portable solution based on the previous static approaches.

3.2 A Parallel Visibility Computation Algorithm

Horizon-Cut requires some minor modifications in order to run properly on a single sector.

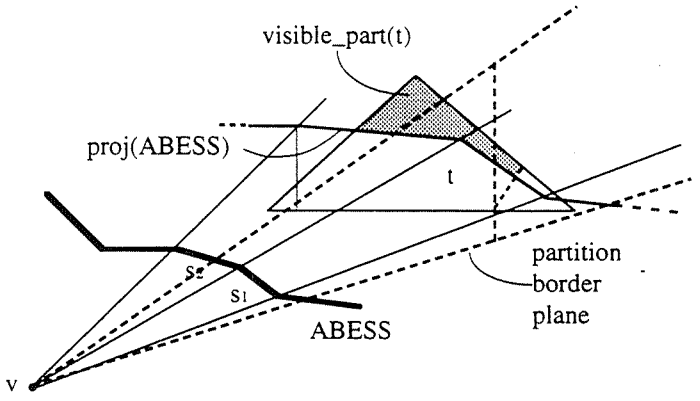

Figure 10: ABESS – triangle cut in the case of a border triangle.

The input dataset is loaded at initialization time. We decided to load all datasets on each *pe*, in order to make successive runs of the same algorithm (with different viewpoints) possible, without requiring further input triangulation loads. This is adequate for architectures where the local memory is sufficient for storing the whole triangulation; in other cases, the triangulation load can be simply limited to the triangles contained in the local sector.

Parallel *Horizon-Cut* first computes the extents of the local sector. It then marks the triangles on the border of the sector with a *border triangle* mark. If topological information is maintained in the data structure representing the triangulation (as in our implementation), such marking phase can be implemented with high efficiency, by accessing only the border triangles. ABESS and C_Π are implemented in this case as non–circular doubly linked lists and, clearly, they only connect triangles contained (partially or completely) in the local sector.

At the end of the initialization phase, visibility processing starts. The modifications required are limited to the control of ABESS and C_Π end–list condition (which requires inverting the scanning direction) and the management of ABESS – triangle cut when the latter is a *border triangle* (i.e., when the triangle is intersected by one of the sector's radial edges).
This is the only significant modification to the sequential version. In fact, when a triangle t_b is a *border* triangle its visibility depends on the triangles contained in both adjacent sectors. But the problem can be divided into two subproblems, by splitting t_b into two regions by the sector border plane (the plane orthogonal to the x-y plane and whose intersection with x-y is the border line between the two sectors) and separately computing visibility on these two regions. ABESS–triangle cut is therefore implemented on border triangles by computing only the visible subpart within the sector; ABESS is updated accordingly (see Figure 10).

3.3 Scheduling and Load Balancing

Two scheduling policies have been implemented. The first is a **static scheduling** policy: the number of partitions generated is equal to the number of *pe*'s, and a single partition is assigned to each *pe*. In this solution, different processing times due to uneven loads of the generated sectors might reduce global efficiency.

For better exploiting the available processing power, a **dynamic scheduling** policy has also been implemented. In the dynamic scheduling implementation, a greater number of partitions than the number of *pe*'s is selected statically. Then, a **master** process allocates tasks to **worker** nodes. As soon as a *worker* has terminated visibility computations on the assigned sector, it communicates to the *master* its free state, and the master replies with a new sector extent. The communication rate between master and worker is extremely low, since it is limited to the exchange of *"free state"* messages and new sector extents (a few bytes).

4 Experimental Results

Our parallel implementation has been designed and tested on a hypercube multicomputer, an nCUBE 2 system model 6410. The nCUBE 2 has an MIMD distributed memory architecture, with 4/16 MBytes of local memory on each *pe* in the configuration used. The CPU technology of this machine is quite conservative: running at a clock rate of 20 MHz, a single nCUBE 2 node processor is rated at 7.5 MIPS and 3.3 MFLOPS, single precision, or 2.4 MFLOPS, double precision.

All computing times presented in Tables 1, 2, 3 and 4 are computed as the average of the computation times (in seconds) over 100 different runs of the algorithm on the same terrain, but with different viewpoints. Viewpoints are randomly generated as points contained in the rectangular domain of triangulated terrains.

Table 1 presents the computing times of algorithm *Horizon-Cut* on increasing resolution triangulated terrains. Run time increases less than linearly with the dataset resolution, giving a sublinear practical complexity much lower than the asymptotic worst case complexity compiled in [4], $O(n^2\alpha(n))$.

The results obtained by using the three static partitioning strategies are reported in Table 2 (on each column, a different number of sector is used, with number of sector equal to the number

		1	4	8	16	32	64	128
Σ: 21K triangles		11.54						
equal angles:	times		6.08	3.79	2.26	1.36	0.85	0.59
	speed up		1.89	3.04	5.09	8.46	13.46	19.47
quadrant:	times		5.92	2.74	1.59	1.06	0.74	0.56
	speed up		1.94	4.21	7.25	10.87	15.42	20.54
equal areas:	times		4.65	2.28	1.28	0.80	0.57	0.44
	speed up		2.48	5.04	8.97	14.26	20.20	25.87
Σ: 61K triangles		30.51						
equal angles:	times		16.02	9.88	5.66	3.23	1.89	1.17
	speed up		1.90	3.08	5.38	9.43	16.06	25.87
quadrant:	times		15.83	6.84	3.75	2.39	1.61	1.13
	speed up		1.92	4.46	8.11	12.74	18.91	27.00
equal areas:	times		12.07	5.53	2.79	1.60	1.04	0.74
	speed up		2.52	5.51	10.89	19.01	29.19	41.11

Table 2: Evaluation of the three static partition strategies (times in seconds).

of *pe*'s). The *equal area* criterion gives the most balanced load, confirming the assumptions outlined in Section 4.1.

The results obtained using *equal area* partitions on a number of triangulations are presented in Table 3. Times, speedup, efficiency and the number of triangles computed per seconds (t/sec) are reported. In order to get a closer look at the effective *pe* load, we report the time required by the most complex sector (t_{max}), together with the time of the least complex (t_{min}) and the mean of the times required to process all the sectors (t_{mean}). Obviously, efficiency and speedup are computed using t_{max} values. Figure 11 shows the efficiency of parallel *Horizon-Cut* with increasing dataset resolution and number of *pe*'s used.

The **dynamic scheduling** version of *Horizon-Cut* has been evaluated by generating, for each terrain map, a number of partitions which doubles the number of *pe*'s actually used. In Table 4, the times obtained by dynamic and static scheduling are compared. Each reported value is, as in the previous tables, the average of 100 runs, each performed with a different viewpoint. Times with dynamic scheduling are, generally, slightly higher, because the dynamic scheduling approach reduces the times of runs with unbalanced loads but, at the same time, the finer partition used increases overheads (which depend on the number of border triangles

	t_{max}	t_{min}	t_{mean}	*speedup*	$eff.$	t/sec
4 *pe*'s						
Σ: 21K tria.	4.65	1.36	3.01	2.48	0.62	4.5 K
Σ: 41K tria.	8.29	2.25	5.32	2.49	0.62	4.9 K
Σ: 61K tria.	12.07	3.24	7.70	2.52	0.63	5.0 K
Σ: 80K tria.	13.52	4.94	9.02	2.69	0.67	5.9 K
Σ: 165K tria.	28.89	7.41	18.75	2.56	0.64	5.7 K
Σ: 236K tria.	41.93	10.42	27.19	2.56	0.64	5.6 K
16 *pe*'s						
Σ: 21K tria.	1.28	0.58	0.90	8.97	0.56	16.4 K
Σ: 41K tria.	1.99	0.96	1.49	10.34	0.64	20.6 K
Σ: 61K tria.	2.79	1.36	2.12	10.89	0.68	21.8 K
Σ: 80K tria.	3.12	1.70	2.45	11.67	0.72	25.6 K
Σ: 165K tria.	6.15	3.11	4.78	12.05	0.75	26.8 K
Σ: 236K tria.	8.68	4.44	6.78	12.37	0.77	27.1 K
64 *pe*'s						
Σ: 21K tria.	0.57	0.20	0.36	20.20	0.31	36.8 K
Σ: 41K tria.	0.80	0.34	0.54	25.68	0.40	51.2 K
Σ: 61K tria.	1.04	0.48	0.74	29.19	0.45	58.6 K
Σ: 80K tria.	1.14	0.56	0.80	31.84	0.49	70.1 K
Σ: 165K tria.	1.96	1.06	1.47	37.65	0.58	84.1 K
Σ: 236K tria.	2.58	1.49	2.01	41.54	0.64	91.4 K

Table 3: Times on different resolution dataset, using the equal area partition.

	4 *pe*'s		**8** *pe*'s		**16** *pe*'s		**32** *pe*'s	
	stat	dyn	stat	dyn	stat	dyn	stat	dyn
Σ: 21K	4.65	4.80	2.28	2.35	1.28	1.39	0.80	0.94
Σ: 41K	8.29	8.68	3.87	3.91	1.99	2.15	1.19	1.38
Σ: 61K	12.07	11.99	5.53	5.41	2.79	2.91	1.60	1.80

Table 4: Comparison of dynamic and static scheduling times; the number of sectors is equal to the number of *pe* for static scheduling, or double to the *pe*'s number in the case of dynamic scheduling.

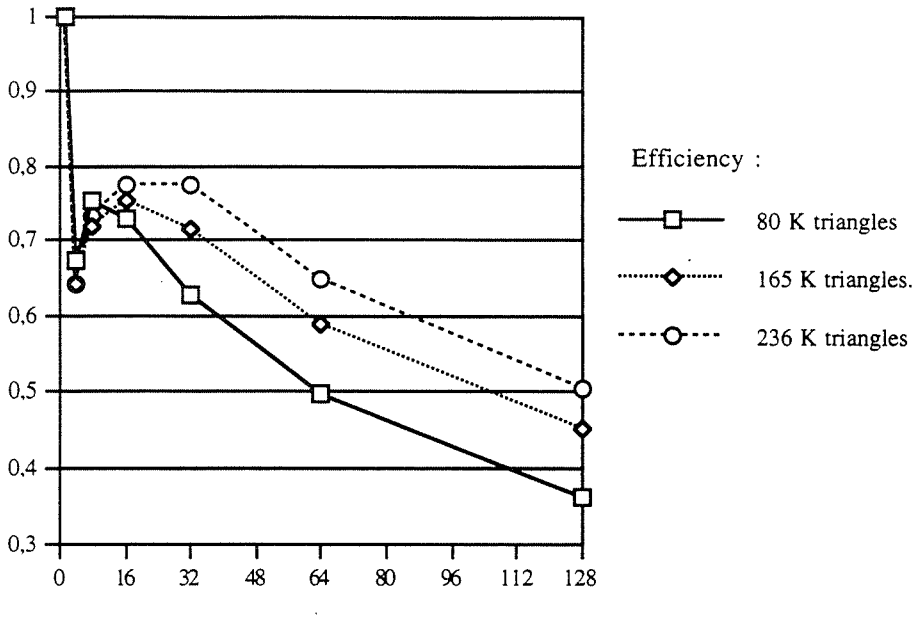

Figure 11: Efficiency of parallel *Horizon-Cut* on three datasets.

to be marked and clip on both the adjacent partitions).

If both the static and the dynamic versions are executed on a single unbalanced load configuration, the computing time of the dynamic version results generally lower. For example, *Horizon-Cut* times on two unbalanced configuration (a highly irregular TIN with 59K triangles processed using 8 *pe*'s) are reduced from 10.35 sec. (static) to 6.65 sec. (dynamic), and from 9.02 sec. (static) to 6.75 sec. (dynamic). Critical configurations, however, cannot be detected "a priori", i.e., before computing visibility. As already pointed out, unbalance depends on both the shape of the surface and the position of the viewpoint. The two scheduling strategies thus cannot be compared on unbalanced configurations only, but on a number of runs as in Table 4. This table shows that dynamic scheduling tends to give better results only if (a) few *pe*'s are used and (b) the dataset resolution is high (and, therefore, when the overhead due to the finer partition used is proportionally lower).

5 Concluding Remarks

A parallel visibility algorithm has been presented. The algorithm works on triangulated terrains and returns the set of regions visible from a set of viewpoints positioned within the triangulation. The sequential solution is an optimized version of the algorithm described in [4] which

has led to a reduction of both ABESS and output data fragmentation, and significantly affects algorithm efficiency. Speedups of an order of magnitude over the original sequential implementation have been obtained.

Several parallelization stategies for the previous algorithm have been discussed. A static partition strategy has been implemented on an NCUBE 2 multicomputer. The structure of the parallel algorithm allow simple and portable implementation on environments with coarse grain processor elements, such as networks of workstation or multicomputers.

The paper reports the efficiency of an implementations on an nCUBE 2 hypercube multicomputer. The scability of our solution is sufficiently good, showing an efficiency higher than 0.5 up to 64 *pe*'s (considering a TIN with about 100K triangles); efficiency clearly improves with higher TIN resolution. The most efficient runs have been obtained using more than 8 *pe*'s, in order to guarantee a sufficiently balanced partition, and allowing a subdivision with more than 2K triangles per sector. By using 8 or more *pe*'s, 0.5 efficiency is in general obtained on data partitions with sectors containing 2K triangles or more, while efficiency 0.75 is obtained on dataset which allow the generation of partitions containing not less than 10K triangles.

Running times suffer from the low performance of the nCUBE 2 *pe*'s; porting *Horizon-Cut* on a state-of-the-art workstation cluster should guarantee times of more than one order of magnitude lower. The performance obtained by executing *Horizon-Cut* in parallel on 128 NCUBE nodes should be reachable on a network of 4/8 cooperating workstations.

Acknoledgements

The work described in this paper has been partially supported by the Progetto Finalizzato "Sistemi Informatici e Calcolo Parallelo" of the Italian National Research Council.

The authors would like to thank Raffaele Perego, Davide Marzilli and Giuseppe Giurleo for valuable discussions and support in the implementation.

References

[1] M.De Berg, D.Halperin, M.Avennars, J.Snoeyink, and M. Van Kreveld. Efficient ray shooting and hidden surface removal. In *Proceedings 7th ACM Symposium on Computational Geometry*, pages 21–30, 1991.

[2] J.D. Boissonnat and K.Dobrindt. On-line construction of the upper envelope of triangles in E^3. In *Proceedings 4th Canadian Conference on Computational Geometry*, August 1992.

[3] N. Carriero and D. Gelenter. How to write a parallel program: a guide to the Perplexed. *ACM Computing Surveys*, 21(3):323–358, September 1989.

[4] L. De Floriani, B. Falcidieno, D.M. Jung, G. Nagy, and C. Pienovi. Polyhedral terrain description using visibility criteria. Technical Report No. 17, Istituto per la Matematica Applicata del C.N.R., October 1989.

[5] L. De Floriani, B. Falcidieno, G. Nagy, and C. Pienovi. On sorting triangles in a Delaunay tessellation. *Algorithmica*, (6):522–532, 1991.

[6] L. De Floriani, B. Falcidieno, C. Pienovi, D. Allen, and G. Nagy. A visibility-based model for terrain features. In *Proceedings 2nd International Symposium on Spatial Data Handling*, pages 235–250, Seattle, July 1986.

[7] L. De Floriani, G. Nagy, and H. Nair. Visibility-oriented criteria for surface characterization. Technical Report 88-824, ECSE Dept., Rensselaer Polytechnic Institute, 1988.

[8] G.C. Fox et al. *Solving Problem on Concurrent Processors.* Prentice Hall, 1988.

[9] W. R. Franklin and M. S. Kankanhalli. Parallel object-space hidden surface removal. *ACM Computer Graphics*, 24(4):87–94, Aug. 1990.

[10] H.Edelsbrunner, L.J.Guibas, and M.Sharir. The upper envelope of piecewise linear functions: algorithms and applications. *Discrete and Computational Geometry*, (4):311–336, 1989.

[11] J.Lee. Analysis of visibility sites on topographic surfaces. *International Journal of Geographic Information Systems*, 5(4):413–425, 1991.

[12] C.L. Lawson. Software for C1 Surface Interpolation. In J.R. Rice, editor, *Mathematical Software III*, pages 161–164. Academic Press, 1977.

[13] M.Cazzanti, L. De Floriani, E. Puppo, and G.Nagy. Visibility computation on a triangulated terrain. In *Proceedings 8th International Conference on Image Analysis and Processing*, September 1991.

[14] M.F.Goodchild and J.Lee. Coverage problems and visibility regions on topographic surfaces. *Annals of Operation Research*, 20:175–186, 1989.

[15] M.J.Katz, M.H.Ovemmars, and M.Sharir. Efficient hidden surface removal for objects with small union size. In *Proceedings 7th ACM Symposium on Computational Geometry*, pages 31–40, 1991.

[16] F.P. Preparata and J.S.Vitter. A simplified technique for hidden line elimination in terrains. In *Proceedings STACS'92*, pages 193–200, Paris, February 1992.

[17] F.P. Preparata and M.I. Shamos. *Computational Geometry - An Introduction.* Springer-Verlag, 1985.

[18] R.Cole and M.Sharir. Visibility problems for polyhedral terrains. *Journal of Symbolic Computation*, 7(1):11–30, 1989.

[19] J.H. Reif and S. Sen. An efficient output–sensitive HSR algorithm and its parallelization. In *4th ACM Symposium on Computational Geometry*, pages 193–200. A.C.M. Press, 1988.

[20] R. Scopigno, A. Paoluzzi, S. Guerrini, and G. Rumolo. Parallel Depth–Merge: a paradigm for hidden surface removal. *Computers & Graphics*, 17(5):583–592, 1993.

[21] V.S. Sunderam and G.A. Geist. Network–Based Concurrent Computing on the PVM System. *Concurrency: Practice and Experience*, 4(4):293–311, July 1992.

[22] T.D.Garvey. Evidential reasoning for land-use classification. In *Proceedings Workshop on Analytical Methods in Remote Sensing for Geographic Information Systems*, pages 171–202, Paris, 1986.

[23] T. Theoharis. *Algorithms for Parallel Polygon Rendering.* Lecture Notes in Computer Science No.373, Springer Verlag, 1989.

[24] L.G. Valiant. General Purpose Parallel Architectures. Technical Report TR-07-89, Harvard Technical Report, 1989. Also Handbook of Theoretical Computer Science, J.van Leeuwen Ed., North Holland, Amsterdam, 1990.

[25] S. Whitman and R. Parent. A survey of parallel hidden surface removal algorithms. In *Parallel Processing and Advanced Architectures in Computer Graphics*, pages 157–173. A.C.M. SIGGRAPH '89 Course Notes, 1989.

[26] A. Wiernik and M. Sharir. Planar realizations of nonlinear Davenport–Schinzel sequences by segments. *Discrete and Computational Geometry*, 3:15–47, 1988.