On Levels of Detail in Terrains

Mark de Berg Katrin T. G. Dobrindt

UU–CS–1995–12
April 1995

Utrecht University
Department of Computer Science
Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : + 31 - 30 - 531454

On Levels of Detail in Terrains

Mark de Berg Katrin T. G. Dobrindt

Technical Report UU–CS–1995–12
April 1995

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924–3275

On Levels of Detail in Terrains*

Mark de Berg Katrin T. G. Dobrindt

Abstract

In many applications it is important that one can view a scene at different levels of detail. A prime example is flight simulation: a high level of detail is needed when flying low, whereas a low level of detail suffices when flying high. More precisely, one would like to visualize the part of the scene that is close at a high level of detail, and the part that is far away at a low level of detail. We propose a hierarchy of detail levels for a polyhedral terrain (or, triangulated irregular network) that allows this: given a view point, it is possible to select the appropriate level of detail for each part of the terrain in such a way that the parts still fit together continuously. The main advantage of our structure is that it uses the Delaunay triangulation at each level, so that triangles with very small angles are avoided. This is the first method that uses the Delaunay triangulation and still allows to combine different levels into a single representation.

Keywords: Computational geometry, hierarchical data structure, multiresolution modeling, terrain, Delaunay triangulation.

1 Introduction

A *terrain* is the graph of a continuous function that assigns to every point on the plane an elevation. Terrains model mountain landscapes and are thus an important class of scenes in several areas—geographic information systems (GIS) and flight simulation are obvious examples. For rendering purposes it is convenient to model a terrain as a collection of disjoint triangles. Such a representation is called a *triangulated irregular network*, or TIN, in geographic information systems, and it is called a *polyhedral terrain* in computational geometry.

For a realistic representation of a terrain millions of triangles are needed. In applications such as flight simulation a terrain should be rendered at real time, but even with modern technology it is impossible to achieve this when the number of triangles is this large. Fortunately, a realistic image of the terrain is only crucial when one is close to the terrain and only a small part of the terrain is visible; when one is flying high above the terrain a coarse representation suffices. So what is needed is a hierarchy of representations at various levels of detail; this makes it possible for a given view point to render the terrain at an adequate level of detail. Subsequent levels of the hierarchy should not differ too much in appearance: switching to more and more detailed representations when zooming in (this happens for instance in flight simulation, when one is landing) should not cause disturbing 'jumps' in the image. On the other hand, the reduction of the number

*This research was supported by the Dutch Organisation for Scientific Research (N.W.O.) and by ESPRIT Basic Research Action No. 7141 (project ALCOM II: *Algorithms and Complexity*)

of triangles in subsequent levels should not be too small, otherwise too many levels would be needed, resulting in an unacceptable increase in storage. It is in general insufficient for rendering purposes to use only one level of detail at a time: although some part of the terrain may be close to the view point, another part (the horizon, for example) can still be far away. Hence, one would like to combine parts from different levels into a single representation of the terrain, such that each part of the terrain is rendered with appropriate detail. This means that the levels cannot be completely independent, as it should be possible to glue them together smoothly. Although the idea of multiresolution models is a quite old, there are not many results for their automatic creation—Heckbert and Garland [HG94] survey the existing multiresolution techniques.

We study this problem in the following setting. We are given a collection of n data points in the plane, each with its own elevation. We wish to automatically compute a hierarchy of levels, where the most detailed level corresponds to a triangulation of the complete set of data points. Less detailed levels should correspond to triangulations of subsets of the set of data points. The hierarchy should be such that the approximation error between subsequent levels is small, but the amount of storage should not increase to much.

There are several papers dealing with this problem—De Floriani et al. [DMP94] give an overview. Previous approaches can be subdivided into two categories—see below. A related problem is the problem of simplifying general surfaces [HDD+93, DLR90] or terrains [FL79, CG88, Lee89, PM93].

In the first category one starts with a triangulation of a (small) subset of the data points [DFNP84, PF87]. This is the coarsest representation. To obtain the next level, the triangles are refined by adding new data points inside them and retriangulating each triangle with its new interior points. Thus each triangle is replaced by a number of smaller triangles. This process is repeated until all data points have been added, or some precision criterion is met. Such a hierarchy can be modeled as a tree. The nodes in this tree correspond to the triangles in the hierarchy, and there is an arc from the node corresponding to a triangle t to the node corresponding to a triangle t' if the triangles belong to consecutive levels and t' is contained in t. There is also an extra root node which is connected to all triangles of the first level. Figure 1 shows an example of a hierarchy and the corresponding tree. For tree-like hierarchies it is quite easy to combine

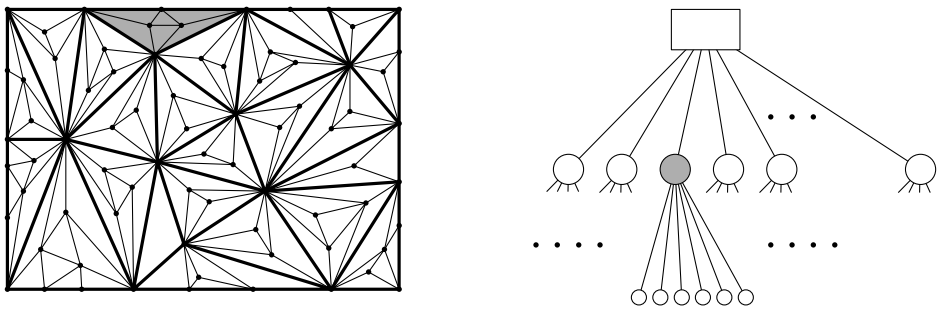

Figure 1: A two-level hierarchy with a tree structure.

different levels into one representation (see Section 3). Unfortunately, they have a serious drawback: the triangles at higher detail levels are very skinny, because the edges of the

initial triangulation remain present at more detailed levels.[1] This effect is already apparent in the two-level hierarchy of Figure 1. Skinny triangles can cause robustness and aliasing problems.

The second category uses the Delaunay triangulation [PS85] of the set of (two-dimensional) data points at every level. This triangulation has the nice property that it maximizes the minimum angle of the triangles—see Section 2 for more details. Thus robustness and aliasing problems are reduced. The hierarchies of this category can be represented by a directed acyclic graph: the nodes in this graph again correspond to the triangles in the hierarchy, and there is an arc from the node corresponding to a triangle at a certain level and a triangle at the next level if these triangles intersect. An example of such a hierarchy is the *Delaunay pyramid* [De 89], which is obtained by always adding the data point with the maximal error and retriangulating using the Delaunay criterion. Unfortunately, the hierarchies of the second category also have a serious drawback: because they do not have a tree-like shape it seems difficult to combine the triangles of different levels into one representation. This is illustrated in Figure 2. Left in this figure is the Delaunay triangulation of a subset of the points, and in the middle is the Delaunay triangulation of the whole set. The corresponding graph structure is also shown on the right.

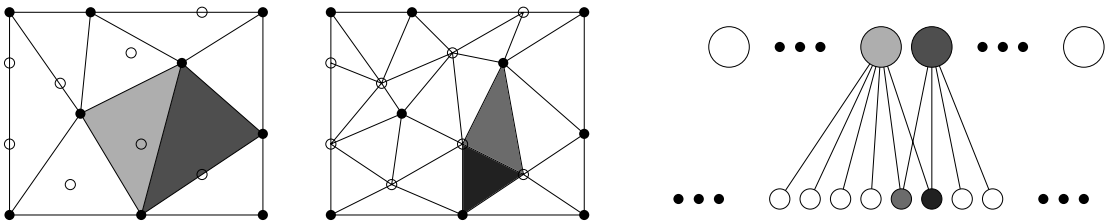

Figure 2: A two-level hierarchy that uses the Delaunay triangulation.

As far as we know, none of the hierarchical structures described so far simultaneously allows the use of Delaunay triangulations in the levels and the combination of parts from different levels into one representation. Whether this could be done was mentioned as an open problem by De Floriani et al. [DMP94, DP92]. We answer this question by describing a hierarchy that has both properties.

The idea behind our hierarchy is the following. We start with the initial terrain that forms the most detailed level of the hierarchy. To go from one level to the next, coarser level, a number of non-adjacent vertices of the terrain is removed, and the terrain is retriangulated. (This idea was introduced by Kirkpatrick [Kir83].) We allow the user some control over this process. In particular, it is possible to fix vertices that are relevant for the shape of the terrain and should not be removed, such as pits, peaks and passes. These 'important' vertices can either be specified explicitly by the user, or they can be computed using existing methods [Lee91]. Which of the non-fixed vertices are removed to go from one level to the next, coarser level is decided according to different heuristic strategies [Ede87, CG88, Lee89]. The hierarchy and how it is constructed are described in Section 2.

[1] There are methods that try to avoid this by adding extra points on the edges of the triangles [SP90, DP92]. These extra points are not necessarily original data points, so this does not quite fit into our model. The problem with this approach is that the introduction of extra vertices on the edges may cause slivers (small holes between adjacent triangles) when combining different levels.

The fact that the vertices we remove are non-adjacent allows us to combine different levels into a single representation. This is done by selecting small groups of triangles from the levels, instead of individual triangles. Section 3 describes this operation in detail. Which triangles are taken from each level depends on a criterion that can be specified by the user. For instance, for a given viewpoint we can ensure that parts of the terrain close to the viewpoint consist of triangles from levels of high detail and parts far away consist of triangles from levels of low detail. It is not only the case that the combined representation uses triangles from Delaunay triangulations of various levels, it can even be shown that the collection of selected triangles always forms the Delaunay triangulation of the vertices in the combination.

Our hierarchy uses only $O(n)$ storage, where n is the number of triangles in the initial terrain. Experiments show that storage increases by a factor of about 4 to 5—see Section 4. For cases where this is still too much, we allow a trade-off between storage and the approximation error of subsequent levels. Apart from the new operation that allows gluing together parts at different levels of details, our hierarchical structure supports the standard operations, such as point location and windowing queries: given a point on the terrain, we can determine in time $O(\log n)$ in which triangle it lies, and given a rectangular window, we can find the triangles inside the window in time $O(\log n + k)$, where k is the number of triangles inside the window.

2 The hierarchy

2.1 Preliminaries

We are given a collection of n data points in the plane, each with its own elevation, which sample a continuous bivariate function. The triangulation of the data points is called a *triangulated irregular network*, or TIN, in geographic information systems, and it is called a *polyhedral terrain* in computational geometry. The set of data points or *vertices* of is denoted by $\mathcal{V}$. We call the area inside the convex hull of the vertices in $\mathcal{V}$, which is the region where the terrain is defined, the *domain* of the triangulation. Figure 3 shows a triangular irregular network (left) and the triangulation of its data points (right).

Often the triangulation of choice is the Delaunay triangulation, which is the dual structure of the Voronoi diagram. For each (two-dimensional) data point p of $\mathcal{V}$, the *Voronoi region* $V(p)$ is the set of points in $\mathbb{R}^2$ that are closer with respect to the Euclidean distance to this data point than to any other data point. The *Voronoi diagram* of $\mathcal{V}$ is the partition of $\mathbb{R}^2$ formed by the Voronoi regions of the data points. The Delaunay triangulation of $\mathcal{V}$, denoted by $\mathcal{DT}(\mathcal{V})$, is the dual graph of the Voronoi diagram: its vertices are the data points, and two vertices are linked by an edge if and only if their respective Voronoi regions are adjacent. It can be shown that this dual structure of the Voronoi diagram forms a triangulation of the data points.[2] Another, equivalent way to define a Delaunay triangulation is to say that it contains all triangles for which the circumscribing circle does not contain any data point in its interior.

The Delaunay triangulation has several nice properties. For instance, it maximizes the minimum angle of the triangles [Sib78], which reduces robustness problems and aliasing problems when rendering. Intuitively, if the function is sufficiently sampled, Delaunay triangulation gives in general a good approximation because it connects the points by

[2]This assumes the data points are in general position, otherwise some extra edges have to be added.

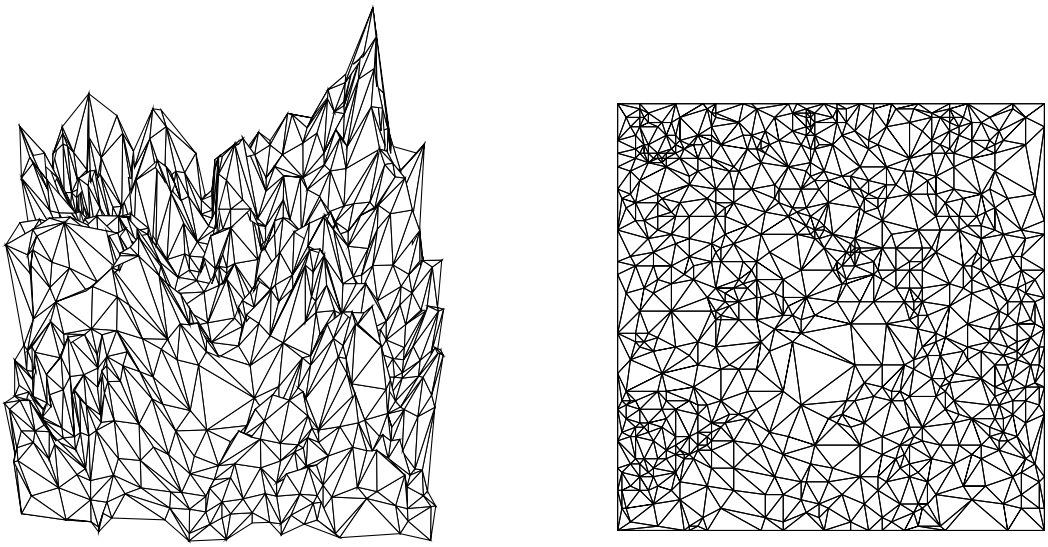

Figure 3: Perspective view of a triangular irregular network and the triangulation of its data points.

---

proximity. Rippa [Rip90] showed that when the Delaunay triangulation is used to interpolate a bivariate function it minimizes the *roughness* of the interpolation. In some cases specific data dependent triangulation can give better results, though. Dyn *et al.*. [DLR90] examine different optimization criteria. Scarlatos and Pavlidis [SP90] adapt the triangulation to surface characteristics (ridge and valley lines). Bern *et al.*. [BEE+93] propose an algorithm for computing the triangulation that minimizes the maximal gradient of the surface to be interpolated. We opted for the use of the Delaunay triangulation because of its properties described above are general and because there are many efficient algorithms for its computation.

We store the Delaunay triangulation as a list (or resizeable array) of triangles. For each triangle we store three pointers (4-bytes) to its incident vertices and three pointer (4-bytes) to its adjacent triangles (in clockwise order). Each triangle requires 24 bytes. For each vertex we store its three coordinates (4-bytes) and a pointer (4-bytes) to an incident triangle. Each vertex consumes 16 bytes. By Euler's formula we know that the total number of triangles is at most 2 times the number of data points. Thus the total memory requirement for the Delaunay triangulation of n data points is $64n$ bytes. This data structure allows us to access for each vertex directly its adjacent vertices and for each triangle its three adjacent triangles.

2.2 Construction of the Hierarchical Representation

The *hierarchical representation* is a sequence $\mathcal{DT}_k, \ldots, \mathcal{DT}_i, \ldots, \mathcal{DT}_0$ of Delaunay triangulations at progressively finer levels of detail; the finest level has index 0, and the coarsest level the index k. It is stored as a directed acyclic graph, whose nodes correspond to the triangles of the Delaunay triangulations $\mathcal{DT}_0$ up to $\mathcal{DT}_k$. The leaf nodes correspond to triangles of the initial Delaunay triangulation $\mathcal{DT}_0$. There is an arc from a triangle t in $\mathcal{DT}_{i+1}$ to a triangle t' in $\mathcal{DT}_i$ if their interiors intersect—see below for details. We call t

a *parent* of t', and t' a *child* of t. A triangle can have several parents and children.

We store the hierarchical representation as a list (or resizeable array) of Delaunay triangulations at progressively finer levels of detail. For each triangle of a Delaunay triangulation we store a list (or resizeable array) of pointers to triangles of the next finer level that it intersects.

The bottom-up construction of the hierarchical representation starts with the finest level, which is the Delaunay triangulation $\mathcal{DT}_0$ of the given set of n data points. The set of vertices $\mathcal{V}_{i+1}$ of the Delaunay triangulation at the next coarser level is obtained from $\mathcal{V}_i$ by removing a *maximal independent* subset I_i of vertices of $\mathcal{V}_i$ that have degree at most d, where d is a constant ($d = 12$ turned out to work fine in our experiments). Two vertices of $\mathcal{V}_i$ are *independent* if they are not adjacent. The union of the triangles incident to such a vertex v of I_i forms a star-shaped polygon whose number of edges equals the degree of v.

Observation 2.1 *The edges of the star-shaped polygon that is the union of the triangles incident to one vertex of I_i are also edges of $\mathcal{DT}_{i+1}$.*

Proof. Indeed, the endpoints of such an edge define two adjacent Voronoi regions. When the vertex v is removed these regions can only grow, so that they must remain adjacent and the edge remains. □

In other words, if we remove a vertex and we retriangulate its surrounding polygon locally using the Delaunay criterion, then the global triangulation remains a Delaunay triangulation. Remark that our method works for any triangulation that meets the criterion of Observation 2.1. Because the vertices in I_i are pairwise non-adjacent, their surrounding polygons have disjoint interiors. Hence, to obtain the Delaunay triangulation $\mathcal{DT}_{i+1}$ we can remove these vertices and retriangulate the 'holes' using the Delaunay criterion. Figure 4 illustrates this. The data points indicated by circles belong to I_i and are removed (left). Their surrounding polygons are retriangulated to obtain $\mathcal{DT}_{i+1}$ (right). It follows

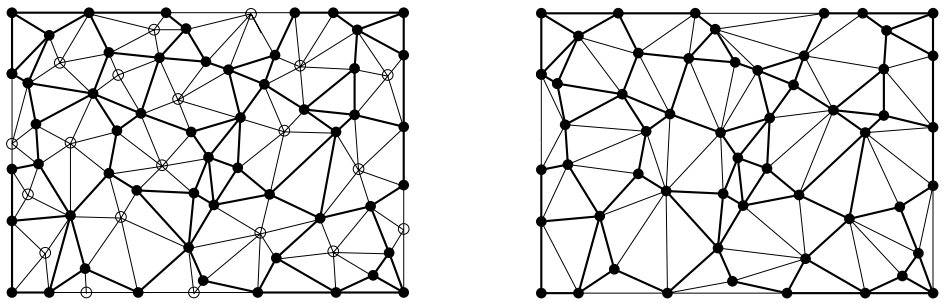

Figure 4: Generating the next level by retriangulation.

from the above observation that only the interior of the triangles of $\mathcal{DT}_i$ and $\mathcal{DT}_{i+1}$ inside the star-shaped polygons can intersect and must be connected by arcs. If no vertex of a triangle in $\mathcal{DT}_i$ is in I_i, then the triangle is duplicated and the two representations are linked by an arc[3] Hence, the outdegree of each triangle is bounded by d and the inde-

[3]In this case it is not necessary to create redundant triangles. Instead, it would be sufficient to update

gree by $d-2$. We recursively continue the process of deleting an independent set and retriangulating until we have obtained the coarsest level we want.

Of course we must be careful not to remove vertices that are characteristic for the surface, such as peaks, pits, and passes. The removal of these vertices may change the shape of the terrain completely. This is definitively not what we want when rendering. Therefore we compute (or let the user specify) a set of these important vertices using existing methods [Lee91] and we fix them. Fixed vertices are never removed, that is the set I_i consists of non-fixed and pairwise non-adjacent vertices of $\mathcal{V}_i$. The vertices of the convex hull of the (two-dimensional) data points are always fixed, so that the terrain keeps its original size. Note that we can fix an edge by fixing its two endpoints. We stop the recursive construction of the hierarchical representation when the set $\mathcal{V}_i$ is only a constant c times larger than the set of fixed points. This constant depends on the parameter d specifying the maximum degree of vertices in an independent set. For $d = 12$, setting $c = 2$ will work. Notice that there is a trade-off between accuracy and data reduction: the more vertices are fixed, the more accurate the representation is, but the less data is reduced. Below the algorithm is described in pseudo-code.

Algorithm *ConstructHierarchy*()

Input: A terrain given as the Delaunay triangulation $\mathcal{DT}(\mathcal{V})$ of a set $\mathcal{V}$ of data points, a set $\mathcal{V}_{fixed} \subset \mathcal{V}$ of vertices that should be in every level, a constant d specifying the maximum degree of a vertex that is allowed to be deleted, and a constant c used to determine when to stop the construction.

Output: The hierarchical representation of the terrain.

1. $i := 0$; $\mathcal{V}_i := \mathcal{V}$; $\mathcal{DT}_i := \mathcal{DT}(\mathcal{V})$;
2. **while** $|\mathcal{V}_i| \geqslant c|\mathcal{V}_{fixed}|$
3. **do** $\mathcal{V}_{i+1} := \mathcal{V}_i$; $\mathcal{DT}_{i+1} := \mathcal{DT}_i$;
4. Determine a maximal set I_i of non-adjacent vertices of $\mathcal{V}_i \setminus \mathcal{V}_{fixed}$ that each have degree at most d;
5. **for** all vertices $v \in I_i$
6. **do** Remove v from $\mathcal{V}_{i+1}$;
7. Remove v and all its incident edges and triangles from $\mathcal{DT}_{i+1}$;
8. Retriangulate the surrounding polygon using the Delaunay criterion;
9. Add the new edges and triangles to $\mathcal{DT}_{i+1}$;
10. Establish an arc of the hierarchy between a triangle of $\mathcal{DT}_i$ incident to v and a new triangle if their interiors intersect;
11. $i := i + 1$;

The set I_i can be determined using various heuristic strategies. For instance, we can choose the non-fixed vertices of $\mathcal{V}_i$ in order of non-decreasing degree. This should guarantee that many vertices are removed [Ede87]. We can also try to discard the least important points, where importance is computed using, for instance, the VIP method [CG88] or drop heuristic method [Lee89]. This should minimize the approximation error between two subsequent levels. It may seem that the last two strategies make the use of fixed points superfluous, but this is not true: an important vertex that is adjacent to less important vertices may be chosen by the algorithm if these adjacent vertices cannot be chosen (for example, because their degree is too high, or because they are adjacent to

for such a triangle its adjanceny relations, if in level $i+1$ it becomes adjacent to a triangle that was not in $\mathcal{DT}_i$.

yet other vertices that were chosen earlier). Figures 5–10 show the view of a terrain at three different levels of detail and the corresponding triangulation, generated using the VIP method.

2.2.1 Analysis

The size and depth of the hierarchy depends on the size of the independent set that we can remove at each step: the more vertices we remove, the smaller the structure. In particular, if we can prove that it is possible to remove a constant fraction of the vertices each time we go from one level to the next, then the structure has linear size and logarithmic depth. The following proposition shows that this is indeed the case. The proof of this proposition follows the analysis of Kirkpatrick [Kir83] closely; the differences only come from the existence of fixed vertices.

Proposition 2.2 *Let d be a constant specifying the maximum degree of the vertices in the independent sets. Then there are constants c and α, $c > 1$ and $0 < \alpha < 1$, such that as long as $|\mathcal{V}_i| \geqslant c|\mathcal{V}_{fixed}|$ the independent set of vertices that is removed in step i of the algorithm has size at least $\alpha|\mathcal{V}_i|$.*

Proof. Obviously $\mathcal{V}_i \supseteq \mathcal{V}_{fixed}$. Consider the subgraph induced by the vertices of $\mathcal{V}_i \setminus \mathcal{V}_{fixed}$ of degree at most d. By Euler's formula and the principle of double counting, this subgraph has at least $\frac{(d-5)(|\mathcal{V}_i|-|\mathcal{V}_{fixed}|)+12}{(d-2)}$ vertices. Since each vertex has degree at most d, an independent subset I_i contains at least $\frac{1}{(d+1)}\frac{(d-5)(|\mathcal{V}_i|-|\mathcal{V}_{fixed}|)+12}{(d-2)}$ vertices. For $\mathcal{V}_{fixed} = \emptyset$ follows that $\alpha \geqslant \frac{(d-5)}{(d+1)(d-2)}$. Else, since $|\mathcal{V}_{fixed}| \leqslant \frac{1}{c}|\mathcal{V}_i|$ follows that $\alpha \geqslant \frac{(d-5)(c-1)}{c(d+1)(d-2)}$. □

In practice (see Section 4) it turns out that with a choice of $d = 12$ and $c = 2$, we can remove around 25% of the vertices at each step. The next result follows easily from the proposition above.

Corollary 2.3 *[Kir83] The hierarchical representation of a terrain given as the Delaunay triangulation of a set of n data points can be constructed in $O(n)$ time. The hierarchy uses $O(n)$ storage and its depth is $O(\log n)$.*

If the Delaunay triangulation of the data points is not given to us beforehand, we have to compute it ourselves. In this case the preprocessing takes $O(n \log n)$ time [PS85].

2.2.2 Point location and windowing operations

Depending on the application, several operations can be required from a hierarchical representation of a terrain. For rendering purposes—this is the application we are mostly interested in—it is important that one can extract a representation at an appropriate level of detail. This is discussed extensively in the next section. Sometimes it is also convenient to be able to perform some other operation. Here we briefly discuss two of them.

The first operation is point location: Given a two-dimensional query point and a certain level, determine which triangle of that level contains the point. To be able to answer such queries efficiently, we add a search structure on top of our hierarchy: after we have computed the hierarchy, we continue the process of deleting independent sets until

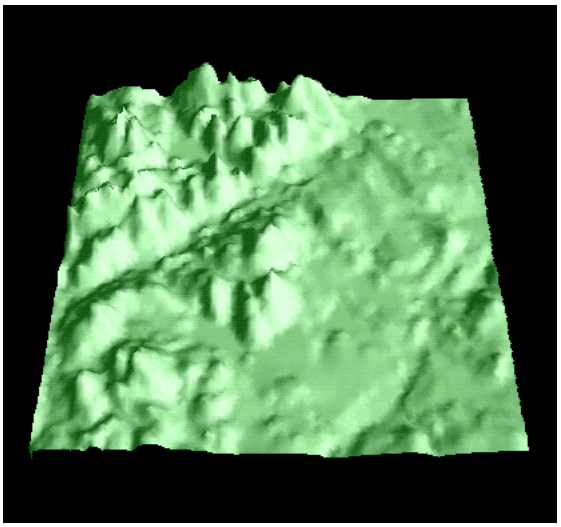

Figure 5: Initial terrain (Gouraud shaded).

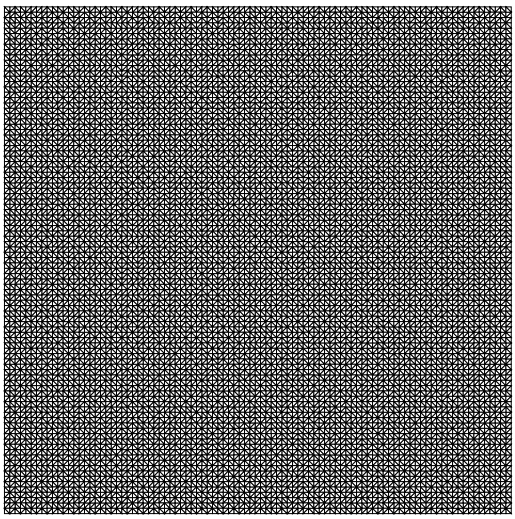

Figure 6: Its triangulation (9,216 pts).

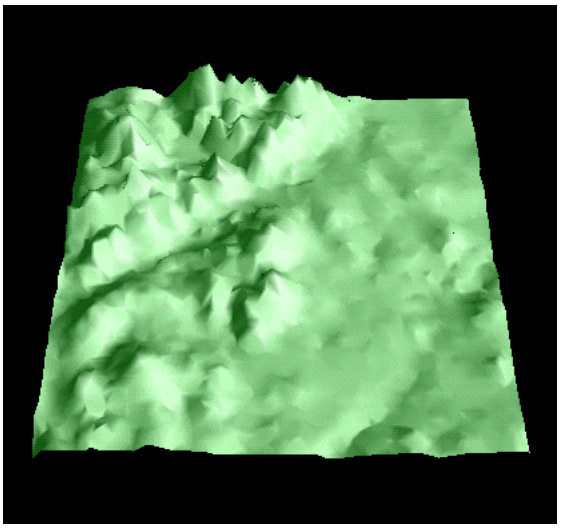

Figure 7: Level 5 (Gouraud shaded).

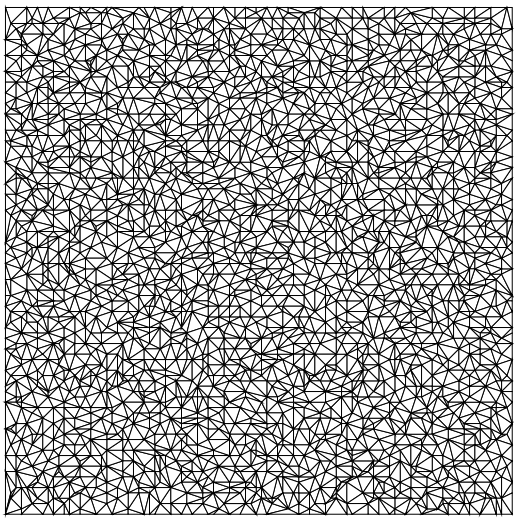

Figure 8: Its triangulation (2,296 pts).

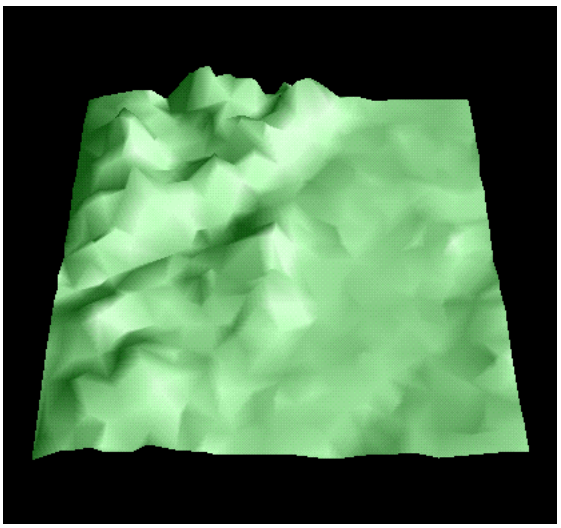

Figure 9: Level 10 (Gouraud shaded).

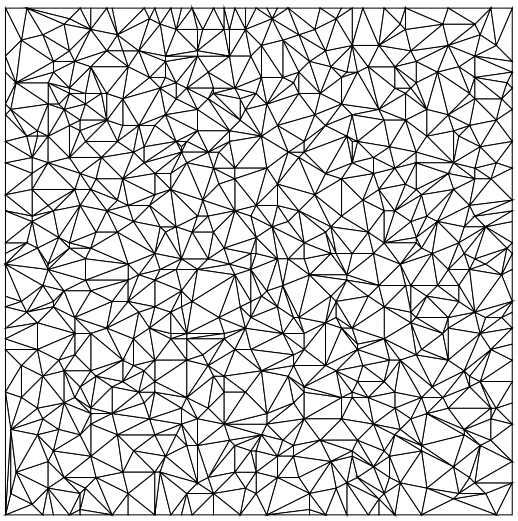

Figure 10: Its triangulation (612 pts).

we are left with a constant number of triangles. In this stage we no longer have fixed vertices. Thus our structure is the same as Kirkpatrick's point location structure. This implies that point location can be performed in $O(\log n)$ time: starting at the root, we follow in the hierarchical representation the path of triangles that contains the query point until we reach required level.

The second operation is windowing: given a rectangular window and a certain level, extract the triangles from that level that lie inside the window. This can be done in $O(\log n + k)$ time, where k is the number of triangles inside the window. To this end we perform a point location query with one of the window vertices to find the triangle at the required level that contains this vertex; starting from this triangle we can then walk through the triangulating using the adjacency relations and visit all other triangles that lie inside the window.

3 Combining different levels

For rendering purposes, a hierarchical representation of a terrain should allow the extraction of a representation at an appropriate detail level. It is generally not sufficient to render the whole terrain at the same level of detail. What we want is, given a viewpoint above the terrain, to combine parts from different levels of our hierarchy into a single representation of the terrain, such that each part of the terrain is rendered with appropriate detail. A criterion of what is appropriate could be the Euclidean distance: parts that are close to the viewpoint should be rendered with high detail and parts that are far away with less detail. It could also be the size of the projection on the view plane: parts whose projection is larger than a threshold should be rendered with high detail and parts whose projection is smaller than a threshold should be rendered with less detail. Another appropriate acceptance criterion might be the screen positional error that is proportional to the quotient of the vertical error and the distance from the view point. Our goal, however, was not to show which heuristic performs the best, but to show that when using the Delaunay triangulation at each level of the hierarchy it is still possible to extract a combination of parts from different levels. The possibility to combine different levels of detail reduces the render time significantly, compared to the time needed when the terrain is rendered at one (high enough) level of detail—see Section 4 for more details.

If the hierarchy is a tree structure [DFNP84, PF87], this can be easily done as follows. We perform a partial top-down traversal of the tree in which the parts of the terrain are extracted at the appropriate level of detail. This recursive traversal starts at the coarsest level of detail. At each level we extract the triangles that satisfy the chosen criterion or that are leaves of the hierarchy. The other triangles are replaced by their child triangles, and we continue recursively with these new triangles. During the traversal we visit only a subtree of the hierarchy and at each level only a subset of the triangles of the triangulation at this level.

The extracted triangles are disjoint, since by construction of the hierarchy the union of the children of a triangle is exactly the triangle, and when we extract a triangle, its parent has not been extracted. Furthermore, the union of extracted triangles is the domain of the triangulation.

When we try to use this algorithm for hierarchical representations that are not tree structures, we run into trouble: we can no longer guarantee that the extracted triangles

are disjoint and that their union is the domain of the terrain. This happens because the hierarchy is a graph, where child triangles can properly intersect their parent triangles, whereas in the tree structure they were completely contained in their parent triangle. However, as already observed in Section 2, the intersections between triangles of $\mathcal{DT}_i$ and level $\mathcal{DT}_{i+1}$ are relatively local: the interior of a triangle t of $\mathcal{DT}_i$ and a triangle t' of $\mathcal{DT}_{i+1}$ can only intersect if

(i) t is incident to a vertex v of I_i, and t and t' are both contained in the star-shaped polygon that is the union of the triangles of $\mathcal{DT}_i$ incident to v, or,

(ii) no vertex of t belongs I_i, and t and t' are different representations of the same triangle.

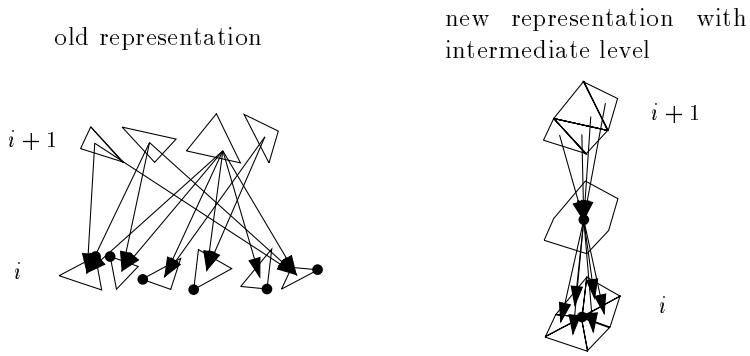

Figure 11: The intermediate level (the dot correspond to a vertex of I_i)

This is the property that allows us to combine different levels: although we cannot replace one triangle at level $i+1$ by several triangles at level i without running into trouble, we *can* replace a group of triangles at level $i+1$ by a group of triangles at level i. This means that sometimes we are not allowed to accept a certain triangle because another triangle from the same group needs to be refined. Because the groups have only constant size, this has only a small, local influence.

To facilitate the combination of levels, we slightly change our hierarchical structure. Instead of linking the triangles of $\mathcal{DT}_{i+1}$ and $\mathcal{DT}_i$ directly by arcs, we add an intermediate level whose nodes corresponds to star-shaped polygons. More precisely, for each vertex in I_i there is one intermediate node, which corresponds to the star-shaped polygon that is the union of the triangles of $\mathcal{DT}_i$ incident to that vertex. There is an outgoing arc from such a star-shaped polygon to each triangle of $\mathcal{DT}_i$ that is contained in the polygon. Also, the polygon has an incoming arc from each triangle of $\mathcal{DT}_{i+1}$ that is contained in the polygon. (This change in the representation also reduces the memory requirement of our structure—see Section 4). Figure 11 illustrates this. The triangles in the new representation are shown closer together to indicate that they are contained in the same polygon. Note that a triangle at level i is in fact part of two polygons: one for the interaction with its parent triangles from level $i+1$—we call this the *parent polygon* of t—and one for the interaction with its children at level $i-1$—we call this the *child polygon* of t.

Algorithm *ExtractCombination*, which is given below, contains pseudo-code for the combination of different levels of the hierarchical representation into one. It uses a function *Accept*(G), which takes as input a group G of triangles, and decides whether to accept

this group or to refine it. This function can be specified by the user. However, it should always accept groups at the finest level. The algorithm is a partial top-down traversal of the hierarchy, starting at the coarsest level. At each level it collects groups of triangles, and tests whether they can be accepted or should be refined.

Figures 12—14 show the result of the algorithm on an example: the terrain at the highest level of detail is shown in Figure 12, and a less detailed level consisting of one fourth of the data points in Figure 13. The extracted combination in Figure 14 has the same number of triangles as the representation in Figure 13, but is detailed close to the viewpoint and crude farer away.

Algorithm *ExtractCombination*()

```
1.  Add all triangles of the coarsest triangulation DT_k to an empty queue Q;
2.  while Q is not empty
3.     do remove the first triangle t from the queue Q;
4.        remove from Q the collection G_t of all triangles in Q that are in the same child
          polygon as t;
5.        if not all triangles from the child group of t were stored in Q
6.           then accept all triangles in G_t;
7.           else if Accept(G_t)
8.                   then accept all triangles in G_t;
9.                   else append to the end of Q the triangles of the parent polygon to
                          which t is linked through an intermediate node;
```

Theorem 3.1 *Algorithm ExtractCombination accepts a collection of triangles whose projections have disjoint interiors and that collectively cover the domain of the terrain. Moreover, the projections of the triangles is the Delaunay triangulation of the set of projected vertices of the triangles.*

Proof. The first part of the theorem follows if we can show that the following invariant holds: (the projections of) the triangles in Q are disjoint and their union is exactly the region that still has to be extracted. This invariant holds before entering the while loop since the triangles from the coarsest level form a triangulation of the domain. We now prove the invariant remains true after the execution of the **while** loop. When the triangles from G_t are accepted then the invariant obviously remains true. When they are not accepted, then they are replaced by the triangles of parent polygon. These triangles are disjoint and have the same union as the triangles in G_t (namely, the corresponding star-shaped polygon). Hence, the invariant also remains true in this case.

In the same way we can show that the triangles in Q plus the already accepted triangles form the Delaunay triangulation of the set of vertices of these triangles. Again, this invariant holds at the beginning, since the coarsest level is a Delaunay triangulation, and it remains true after each execution of the **while** loop: When we replace a group of triangles by another group, then the triangulation of the new group is a Delaunay triangulation locally. Moreover, the new vertex cannot have influence outside the star-shaped polygon, due to Observation 2.1. □

The number of extracted triangles can vary between the size of the finest level and the size of the coarsest level, depending on the acceptance criterion. In practical situation,

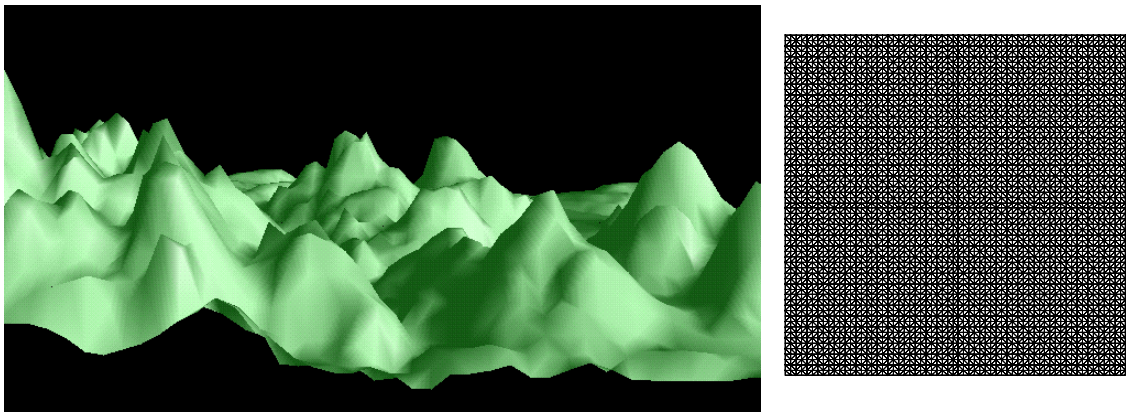

Figure 12: Initial terrain (Gouraud shaded) and its triangulation (4,096 pts).

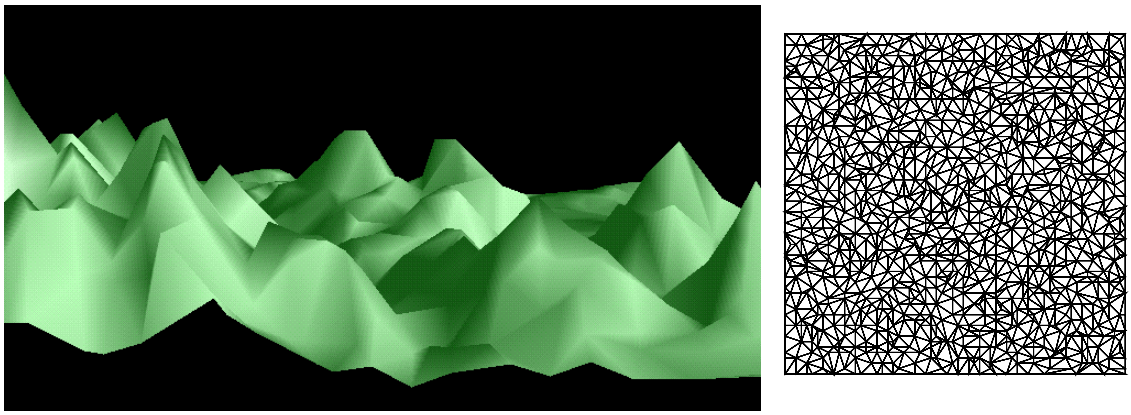

Figure 13: Level 5 (Gouraud shaded) and its triangulation (1,033 pts).

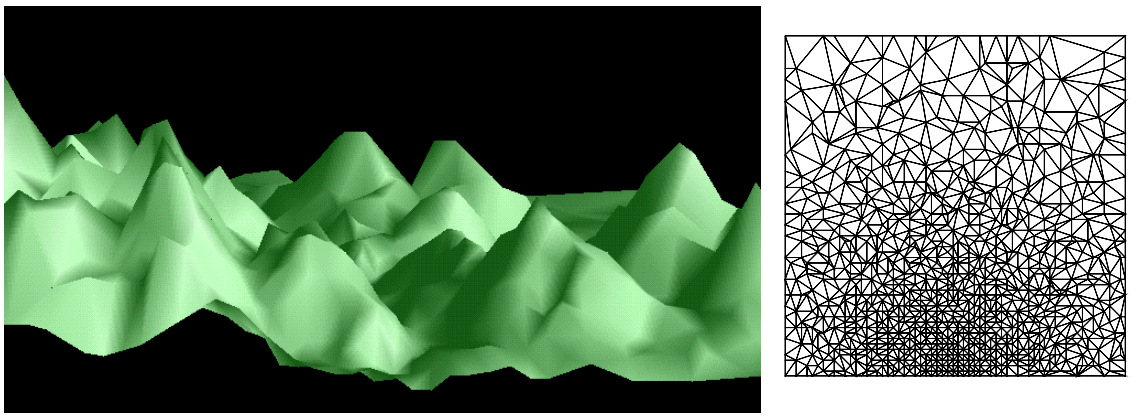

Figure 14: Combination (Gouraud shaded) and its triangulation (980 pts).

however, it can reduce the number of triangles significantly, as compared to rendering the whole terrain at one (high enough) level of detail—see the next section.

Lemma 3.2 *The running-time of the algorithm ExtractCombination is linear in the number of extracted triangles.*

Proof. In the following, we assume that there are no duplicate occurences of the same triangle. (Recall that these duplicate occurences are not necessary and were only introduced to simplify the description.) Let N_E denote the number of extracted triangles up to a given point during the algorithm, let N_V the number of visited triangles up to that point and let N_Q denote the number of triangles currently in the queue Q. We claim that at any point the following invariant holds:

$$N_V \leqslant \frac{d}{2}(N_E + N_Q)$$

Since $N_Q = 0$ at the end of the algorithm, this will prove the lemma. Because $d \geqslant 2$, $N_V = N_Q$ and $N_E = 0$ at the start of the algorithm (after line 1), the invariant is initially true. We now prove the invariant remains true after the execution of the **while** loop. When the triangles from G_t are accepted, then the invariant obviously remains true. When they are not accepted, then they are replaced by the triangles of parent polygon, and thus N_Q increases by two. The number of visited triangles increases by at most d, since the parent polygon contains at most d triangles. Hence, the invariant also remains true in this case.

If we take the duplicate occurences into account, then we can still argue that the running-time is linear in the number of extracted triangles times a logarithmic factor, which is a very weak upper bound. □

4 Experimental results

The algorithm for constructing the hierarchical representation and the supported operations has been implemented in C++ on a Silicon Graphics work station. The unoptimized code is about 4000 lines of C++, including the I/0 and debugging code. The program uses the PlaGeo and SpaGeo library for planar and spatial geometry [Gie94] and the Forms library for the graphical interface [Ove93]. The source code to calculate Delaunay triangulations [Wat92] was obtained from Watson at `maths.uwa.edu.au`.

We tested our algorithm on data set that are based on 1:250,000-scale digital elevation models obtainable by anonymous ftp from `edcftp.cr.usgs.gov:/pub/data/DEM/250`. The chosen data sets consists of samples of the size 128x128 and the spacing of the elevations is 90m (3 arc-seconds). The terrains have different topographic features. The Lake Charles terrain contains many plateaus of constant elevation (elevations between 0m and 106m), the Bangor terrain is a gentle terrain (elevations between 0m and 457m), the San Bernardino terrain is quite rough (elevations between 0m and 3417m). We also tested our algorithm on TIN's that are not based on regular grids.

In our tests, at each recursion step about 25 % of the vertices are removed. The number of triangles in the hierarchy is only at most three times larger than the number of triangles of the initial triangulation—see Table 1. The constants are smaller when we try to select the vertices to be removed in order of non-decreasing degree than in order of importance using the VIP method [CG88] for TIN's based on regular grids, or the drop

	$\|V_{fixed}\| = \sqrt{n}$			$\|V_{fixed}\| = 4$		
	VIP	DH	degree	VIP	DH	degree
Lake Charles (West)	2.90	2.89	2.53	2.89	2.88	2.52
Bangor (East)	2.90	2.88	2.54	2.90	2.87	2.52
San Bernardino (East)	2.91	2.89	2.53	2.90	2.88	2.52
Mount Marcy		2.84	2.51		2.83	2.49
Random TIN		2.89	2.45		2.88	2.43

Table 1: Increase in the number of triangles for some data sets (based on regular grids and TINS) for different numbers of fixed points and different strategies for determining the independent set (not counting the duplicate occurrences of the same triangle in different levels).

heuristic method [Lee89]. The influence of the number of fixed vertices is not significant, as long as this set is not too large.

We next examine the storage overhead caused by the maintenance of the hierarchy. The total amount of storage in bytes (with respect to the data structure described in Section 2) is only a factor of about 4 to 5 larger than the storage of the initial triangulation. Recall that the storage requirement of the initial triangulation of n data-points was about $64n$ bytes. Hence, the total memory requirement for a hierarchical representation of a terrain of n data-points is $256n$ (resp. $320n$) bytes.

To save memory we first remove the duplicate occurrences of the same triangles, which were only introduced for the sake of simplicity. Instead, we store for each triangle the level at which it was created in a 2-byte integer. Furthermore we note that the adjacent triangles of such a triangle can change when going from one level to the next, and thus we store its adjacent triangles in a sorted list (or resizeable array) of pointers. Next, we empirically observed that the representation which include intermediate levels as described in Section 3 needs less storage. This together reduces the storage requirement by one third (the storage constant is roughly 3.5).

Depending on the application we can further reduce the storage requirement. For instance, for rendering we do not need the adjacency relations for the triangles and the vertices. In this case the total storage requirement of the hierarchy is only about a factor of 3 larger than the storage of the initial triangulation. Since we do not store the adjacency relations, the storage requirement of the initial triangulation of n data-points in this case is less than $44n$ bytes. Hence, the total memory requirement for a hierarchical representation of a terrain of n data-points is $132n$ bytes.

If the increase in storage would still be too much, it is possible to have a trade-off between the size of the structure and the difference between subsequent levels. In particular, instead of requiring the deleted vertices to form an independent set, we can require that they form independent groups of a certain size. This way it is possible to delete more vertices, when going from one level to the next, thus reducing the size and depth of the hierarchy. For instance, when we allow the algorithm to remove not only independent vertices, but also independent edges, then at each recursion step about 35 % of the vertices (instead of 25 %) are removed and the number of triangles in the hierarchy is only about two times larger than in the initial triangulation (instead of 2.5 to 3 in

Table 1).

The quality of an approximation depends on the application and on the terrain itself—Lee [Lee91] compares the different methods. The advantage of our algorithm is that we can just plug in the different existing selection methods that have already proven their efficiency. In terms of visual comparison the VIP method seems to perform better than the other methods and is the fastest. In terms of difference in elevation the drop heuristic method seems to performs slightly better than the other two methods—see Figure 15.

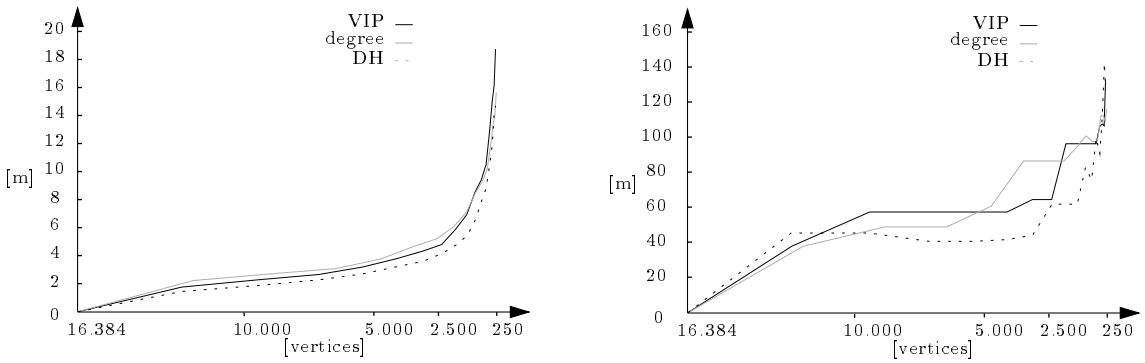

Figure 15: The root mean square error (i) and maximal vertical error (ii) with respect to the initial TIN for the different methods for the Bangor terrain.

It is difficult to give useful figures on the reduction of the rendering time that can be accomplished with our method, because this varies with the acceptance criterion used, with the position of the viewpoint, and so on. This is a direction for future research. The following examples can give an indication of the savings that can be achieved. These examples use terrains whose finest level consist of 32,258 triangles, and whose hierarchy has 16 levels. We applied our extraction algorithm with a view point that is roughly above the center of the terrain, so that the terrain lies completely inside the view window.

We first used an acceptance criterion based on the area of the projection of the triangles. We choose the criterion such that the area closest to the viewpoint had to be rendered with the highest detail. It turned out that the extracted combination consisted of triangles from all levels (thus parts that were far away or very steep consisted of triangles from the coarsest level, as we hoped)—see Figure 16.

The possibility to combine different levels of detail reduces the render time significantly, as compared to the time when the whole terrain is rendered at one (high enough) level of detail while the visual quality of the resulting Gouraud shaded image was the same—see Table 2 Note that the extraction algorithm is not at all optimized.

We also used different acceptance criteria based on the Euclidean distance from the view point, one of which is based on the two-dimensional distance between the projections—see, for instance, Figure 14.

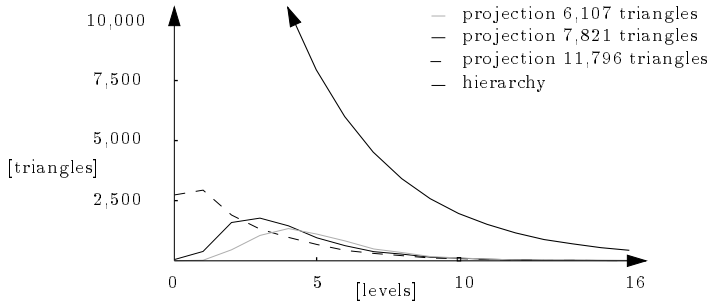

Figure 16: The number of triangles extracted from each level of the hierarchy for the Bangor terrain (i). The acceptance criterion is based on the area of the projection of the triangles and uses different thresholds.

	rendering	extraction
6,107 triangles	0.40	0.49
7,821 triangles	0.57	0.75
11,796 triangles	0.85	0.97
32,558 triangles	2.42	

Table 2: Rendering time (per frame for a 700x700 image) in seconds for the combinations of Figure 16 and for the terrain at the finest level (32,258 triangles) on SGI Indigo2 R4400 with a GU1-Extreme graphics board (hardware z-buffer).

5 Concluding remarks

We proposed a hierarchy of detail levels for a polyhedral terrain that, given a view point, allows to select the appropriate level of detail for each part of the terrain in such a way that the parts still fit together continuously. The main advantage of our structure is that it uses the Delaunay triangulation at each level, so that triangles with very small angles are avoided. This is the first method that uses the Delaunay triangulation and still allows to combine different levels into a single representation. We believe that our technique can be useful in three dimensions and are currently investigating this issue.

Acknowledgements

We thank Jack Meijerink for implementing a first version of the algorithm, and Paul Heckbert for his valuable comments on an earlier version of this paper.

References

[BEE+93] M. Bern, H. Edelsbrunner, D. Eppstein, S. Mitchell, and T. S. Tan. Edge-insertion for optimal triangulations. *Discrete Comput. Geom.*, 10:47–65, 1993.

[CG88] Z. Chen and J. A. Guevara. System selection of very important points (VIP) from digital terrain model for constructing triangular irregular networks. In *Proc. 8th Internat. Sympos. Comput.-Assist. Cartog. (Auto-Carto)*, pages 50–56, 1988.

[De 89] L. De Floriani. A pyramidal data structure for triangle-based surface representation. *IEEE Comput. Graph. Appl.*, 9:67–78, March 1989.

[DFNP84] L. De Floriani, B. Falcidieno, G. Nagy, and C. Pienovi. Hierarchical structure for surface approximation. *Computers & Graphics (Pergamon)*, 8(2):183–193, 1984.

[DLR90] Nira Dyn, David Levin, and Samuel Rippa. Data dependent triangulations for piecewise linear interpolation. *IMA Journal of Numerical Analysis*, 10:137–154, 1990.

[DMP94] L. De Floriani, P. Marzano, and E. Puppo. Hierarchical terrain models: Survey and formalization. In *Proc. ACM Sympos. Applied Comput.*, 1994.

[DP92] L. De Floriani and E. Puppo. A hierarchical triangle-based model for terrain description. In *Proc. Internat. Conf. GIS: Theory and Methods of Spatio-temporal Reasoning in Geographic Space*, Lecture Notes in Computer Science, pages 236–251. Springer-Verlag, 1992.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[FL79] R. J. Fowler and J. J. Little. Automatic extraction of irregular network digital terrain models. *Computer Graphics*, 13(2):199–207, August 1979.

[Gie94] G.-J. Giezeman. PlaGeo, a library for planar geometry, and SpaGeo, a library for spatial geometry. Manual, Dept. Comput. Sci., Univ. Utrecht, Utrecht, Netherlands, 1994.

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proc. SIGGRAPH '93*, pages 19–26, August 1993.

[HG94] P. S. Heckbert and M. Garland. Multiresolution modeling for fast rendering. In *Proc. Graphics Interface '94*, pages 43–50. Canadian Inf. Proc. Soc., 1994.

[Kir83] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12:28–35, 1983.

[Lee89] J. Lee. A drop heuristic conversion method for extracting irregular networks for digital elevation models. In *Proc. of GIS/LIS '89*, pages 30–39, 1989.

[Lee91] J. Lee. A comparison of existing methods for building irregular networks models of terrain from grid digital elevation models. *Int. J. of GIS*, 5:267–285, 1991.

[Ove93] M. Overmars. Forms Library, a graphical user interface toolkit for Silicon Graphics workstations. Manual, Dept. Comput. Sci., Univ. Utrecht, Utrecht, Netherlands, 1993.

[PF87] J. Ponce and O. Faugeras. An object centered hierarchical representation for 3d objects: the prism tree. *Comput. Graphics and Image Proc.*, 38(1):1–28, 1987.

[PM93] Michael F. Polis and David M. McKeown, Jr. Issues in iterative TIN generation to support large scale simulations. *Proc. of 11th Intl. Symp. on Computer Assisted Cartography*, 1993.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[Rip90] S. Rippa. Minimal roughness property of the Delaunay triangulation. *Comput. Aided Geom. Design*, 7:489–497, 1990.

[Sib78] R. Sibson. Locally equiangular triangulations. *Comput. J.*, 21:243–245, 1978.

[SP90] L. Scarlatos and T. Pavlidis. Adaptive hierarchical triangulation. In *Proc. 10th Internat. Sympos. Comput.-Assist. Cartog. (Auto-Carto)*, pages 234–246, 1990.

[Wat92] David F. Watson. *Contouring: A Guide to the Analysis and Display of Spatial Data.* Pergamon, 1992.