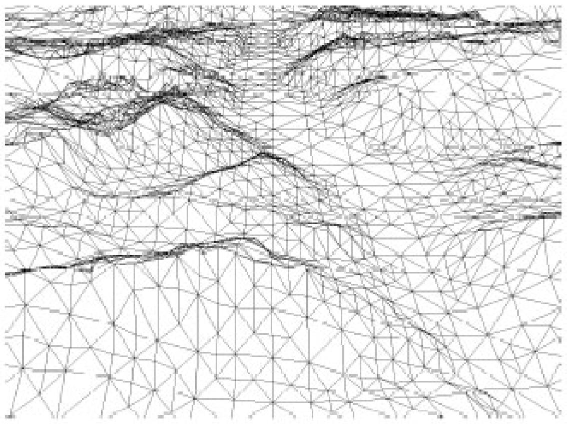
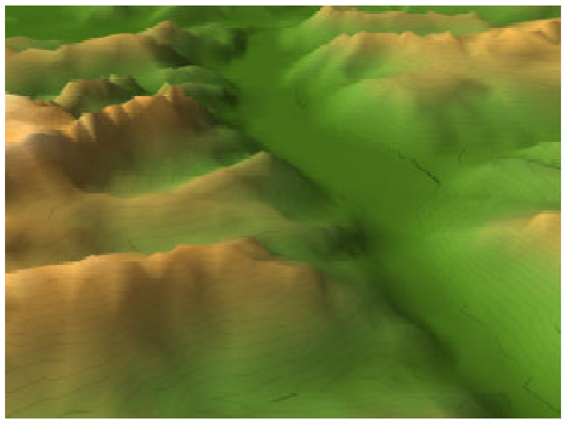
Terrains and triangulations

Terrain applications

Terrain data are used in a variety of applications, such as digital topographic maps, three-dimensional display of terrain for pilot training, virtual reality, games, landscape design and planning, viewshed analysis, planning of road or dam locations, computing of slope and aspect maps to assist geomorphologic studies, and estimation of erosion.

Terrain data sampling

Generally terrain data are measured using aerial photographs or stereo satellite image data. In the computer, a terrain must be approximated by some finite representation of it. A finite set of sampled points is taken, and the elevation of unknown points is estimated and predicted by similarity and proximity.

The set of points may be:

- Regular. A regular square grid is used to specify elevation values at a regular square tesselation of the *xy*-plane terrain domain. For every square on the tessellation the elevation at the center point of the square is stored.

- Irregular. Points are sampled at representative locations. A set of points in the plane is obtained, each with an elevation value.

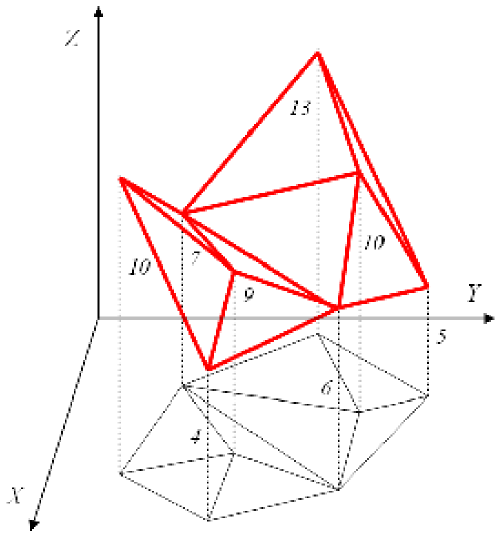
Irregular sampling requires fewer points than regular grid sampling for greater accuracy.

Triangulated Irregular Networks

A *terrain* is a surface in R^3 described by a continuous real bivariated function defined over a simple connected domain D of the xy-plane.

A terrain can be modeled as the graph of a *Triangulated Irregular Network* (TIN) or *Polyhedral Terrain*, $M=(T,F)$, formed by a triangulation $T=\{t_1, \ldots, t_n\}$ of the domain D and by a family F of piecewise linear functions such that:

- every function $z=f_i(x,y) \in F$ is defined on a triangle t_i of T, $i=1, \ldots, n$.
- for every pair of adjacent triangles t_i and t_j, $f_i(x,y)=f_j(x,y)$ for all points (x,y) $\in t_i \cap t_j$.



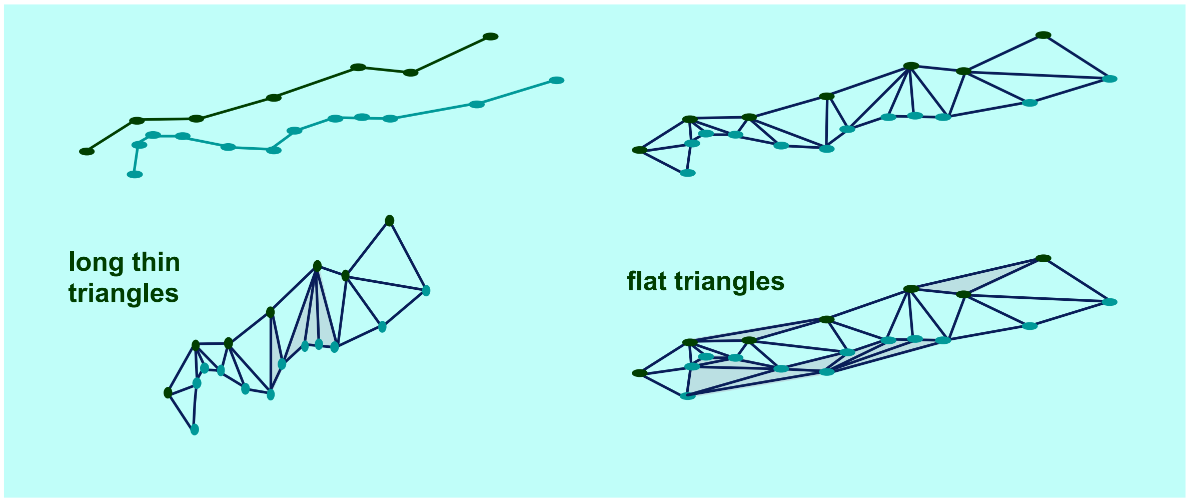
TIN: Triangulated Irregular Network.

Often, a Delaunay triangulation is used as domain because of its good behavior in numerical interpolation.

For any triangle t_i of T, $\bar{t}_i = f_i(t_i)$ is a triangle in R^3 that we will call a *face* of the TIN, and *edge* and *vertex* of the TIN the restriction of each function f_i to and edge and vertex, respectively, of T.

Shortcomings of TIN

- A TIN is generated with a Delaunay triangulation in *xy*-plane, which does not recognize surface morphology;



- Large TINS may be slow to visualize.

Construction of a TIN from a set of irregular points

The *xy*-coordinates of the irregular sampled points are used to create a Delaunay triangulation in the *xy* -plane.

Construction of a TIN from a grid of elevation data.

The algorithm selects a subset of the grid points, such that the Delaunay triangulation of this subset is a TIN that approximates the elevation at all grid points to within a prespecified error ε. In other words, at all grid points, the absolute difference of its z-coordinate and the interpolated z-coordinate of the TIN is at most ε.

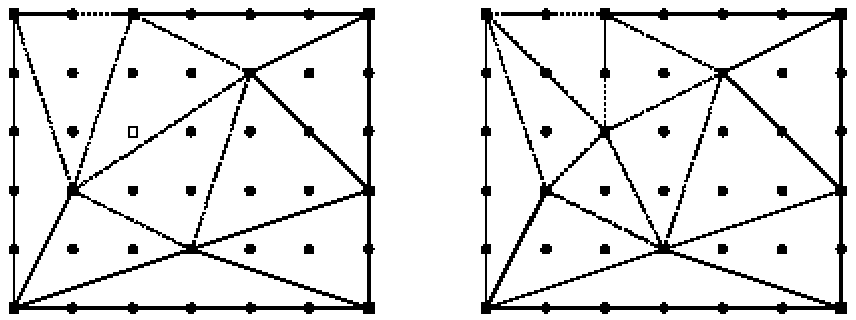
The approach is to start with a coarse TIN with only a few vertices, and keep adding more points from the grid to the TIN to obtain a TIN with smaller error:

1. Let *P* be the set of midpoints of grid cells, with their elevation value. Take the four corner points and remove them from *P*, and put them in a set *S* under construction.

2. Compute the Delaunay triangulation *DT(S)*.

3. Determine for all points in *P* in which triangle of *DT(S)* they fall. For points on edges we can choose either one. Store with each triangle of *DT(S)* a list of the points of *P* that lie in it.

4. If all points of *P* are approximated with error at most ε by the current TIN then the TIN is accepted and the algorithm stops. Otherwise, take the point with maximum approximation error, remove it from *P* and add it to *S*. Continue at step 2.



The right TIN shows the situation if the *square* grid point is the one with maximum error.

If we assume a simple and slow implementation of the algorithm, we observe that at most *n* times a Delaunay triangulation is computed. For each one, the points in *P* are distibuted among the triangles of *DT(S)*. This requires for the whole algorithm $O(n^3)$ tests of the type point in triangle, if $O(n)$ points are added to *S*.

In [1] a method is given that has a worst case performance of $O(n^2 \log n)$ time, and in typical situations even better: typically $O(n \log n)$ time.

Operations on TIN

Traversal

A posible storage schema of a TIN is the DCEL structure. This representaron allows for all necessary traversal operations in an efficient manner.

Point location

Locate a point *a* in a triangle of a triangulation. There exist two basic algorithms:

1. Kirpatric algorithm

Preprocess: $O(n \log n)$ time and $O(n)$ espace.

Query time: $O(\log n)$.

The method is difficult to implement.

2. Jump-and-walk strategy

Choose at random *h* points $p_1, \dots, p_h$ from the set of vertices of the triangles of *T*, and store a copy of those points in an unsorted list. With the copy we store a pointer to the vertex record in the DCEL structure. When we query with a point *a*, we first determine the access point p_i closest to *a* in $O(h)$ time. Then we trace the line segment p_ia, starting at the triangle containing p_i, in $O(m)$ time and space, where *m* is the number of traversed triangles.

If we use a Delaunay triangulation and $h=\lfloor n^{1/3} \rfloor$, the expected query time is $O(n^{1/3})$.

The method is practical and easy to implement.

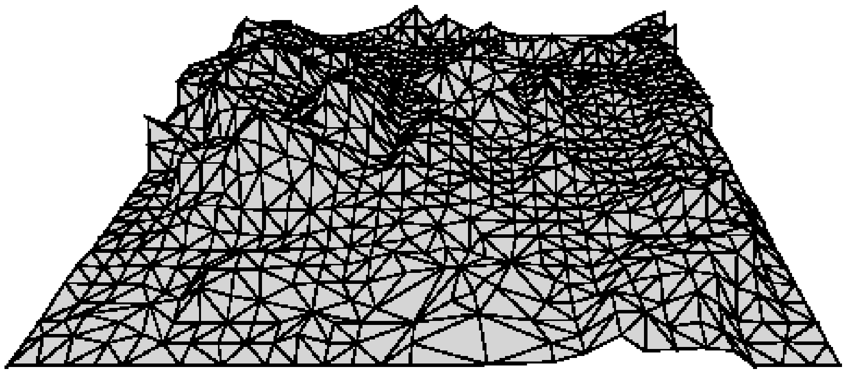
Windowing

Given a triangle or a convex quadrilateral (window) *W*, report all triangles of *T* that intersects *W*.

The time and space needed to determine all intersecting triangles is $O(k)$, where *k* is the total number of triangles that intersect the window *W* [4].

Visualization

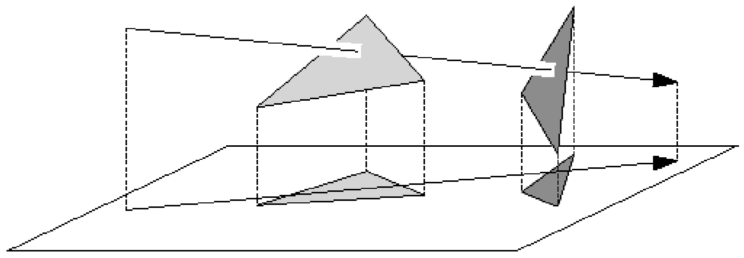
We can produce a *parallel* or *perspective view* of a terrain using the *Painter's Algorithm*, where all triangles are drawn from back to front, so that the ones more to the front erase the ones more to the back. Observe that the image is computed píxel by píxel, so the "structure" of the view is lost.



Perspective view of a TIN.

To simplify we are going to suppose a parallel projection, and let *d* denote the wieving direction. Given a TIN, every two triangles that share an edge must be drawn in a specific depth order. We say that a triangle *t* is in front of a triangle *t'* if there is line with direction *d* that first intersects *t* and then intersects *t'*. If *t* is in front of *t'*, we write $t < t'$. A depth order for a collection *T* of triangles is an ordering $t_1 < t_2, \ldots, t_n$ of the triangles in *T* such that $t_i < t_j$ implies $i > j$. This means that a triangle that is in front of another triangles should come later in the ordering.

Because the projections of triangles of a TIN do not overlap, we can compute the order in the projection: we can project all triangles of the TIN onto the *xy*-plane, project the viewing direction as well, and compute a depth order for the resulting planar scene. This is possible since if a directed line in 3-dimensional space intersects two triangles, then the projected line intersects the projected triangles in the same order.



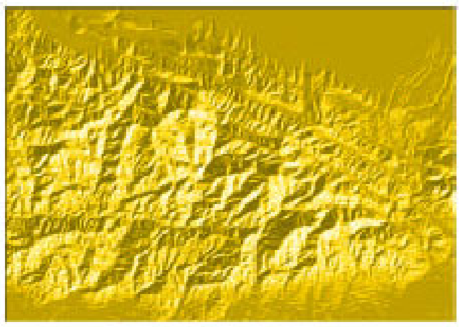
The order of intersection does not change by projection.

By using the dual graph of the TIN, it is possible to compute a depth order for a TIN consisting of *n* triangles in *O(n)* time [3]. Consequently, the TIN can be drawn in perspective view in *O(n)* time using the Painter's Algorithm.

Hill shading

Hill shading is a technique where an imaginary ligth source is placed in 3-dimensional space, and parts of the TIN that don't receive much light are shaded.

One can commpute hill shading in *O(n)* time, where *n* is the number of triangles in the TIN.



Visibility

Given a terrain *M=(T,F)*, we will say that a point *Q=(x,y,z)* is *above* the terrain if the domain point *(x,y)* belongs to some t_i in *T* and $z>f_i(x,y)$. We will call *observation point* any arbitrarily chosen point *V* belonging to or above the terrain. A point *P* belonging to the terrain is called *visible* from *V* if every point *Q* of the interior of the line segment joining *V* and *P* lies above the terrain. We will call *observation segment* (*observation triangle*) any arbitrarily chosen segment *s* (triangle *t*) whose points belong to or are above the terrain, respectively. A point *P* belonging to the terrain is called *weakly visible* from segment *s* (triangle *t*) if *P* is visible at least from a point of *s* (*t*).

We define the *visible domain* of a observation element *o*, a point *V*, a segment *s* or a triangle *t*, as the subset *D′* of the domain *D* formed by the points *(x,y)* such that if *(x,y)* is in t_i then point $(x,y,f_i(x,y))$ is visible from *o*. The *invisible domain* of *o*, denoted *D″* is the complement of *D′* in *D*. The sub-domains *D′* and *D″* are conformed by connected regions, that only in the case in which the observation element is a point are polygonal.

The visibility with respect to *o* can be coded as a map, the *visibility map* of *o*, which consists of a partition of the domain into maximal connected regions, each of which is labeled: or with a face of the terrain, in such a way that if a region *R* is labeled with face *f* then all points of *f* whose vertical projection is in *R* belongs to *D′*, or with invisible if it is a subset of *D″*.

Multi-visibility maps related to multiple observation points/segments are defined combining the single visibility maps of such points/segments.

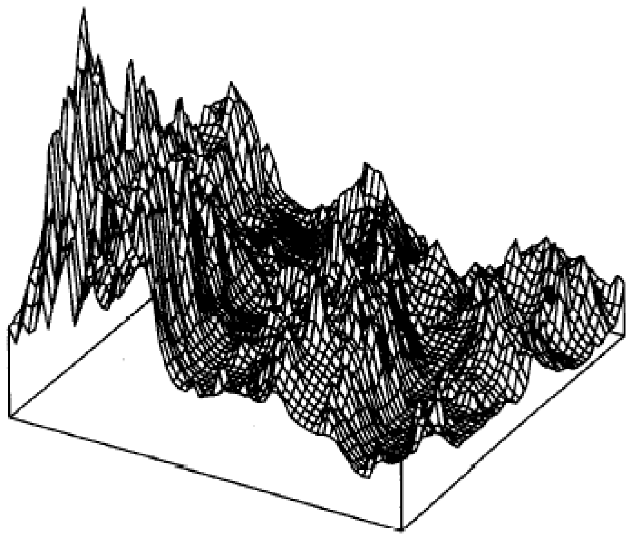
What if the observation point moves?

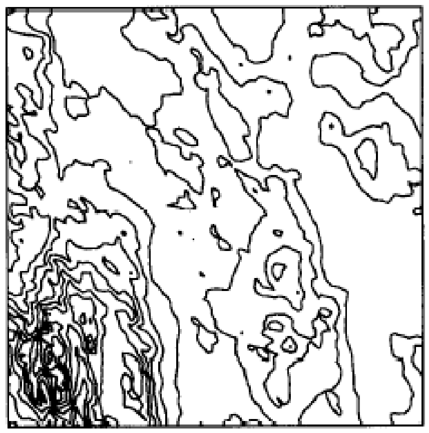
Visibility map construction algorithms

The visibility map of an observation point for a TIN with *n* triangles has a worst-case space complexity of $O(n^2)$. There are algorithms with an $O(n^2)$ running time, which are thus optimal in the worst case. Te worst case usually does not occur in practice, hence it is useful to try and find output-sensitive algorithms: algorithms whose running time not only depends on the number of input triangles, but also on *k*, the complexity of the output. Such algorithms will be faster when *k* is small. For a TIN of *n* triangles, the visibility map from an observation point camn be computed in $O(n\ \alpha(n) \log n + k \log n)$ time, where *k* denotes the complexity of the visibility map [5].

Contour lines

Given an elevation *h*, the *contour lines* at height h are the intersection of the surface with the plane of equation $z = h$. For a TIN, the contour lines are a collection of closed or open polygonal chains. The projections of several contours (corresponding to a sequence of height values) onto the xy-plane form a *contour map*. The contour lines are determined by computing the intersection segments of the triangles query with the horizontal planes at the given height. Since that any triangle contains at most one line segment that is part of these contour lines, to determine the contour lines on a TIN it suffices to examine every triangle once and see if it contributes to the contour lines.





Perspective view of a terrain and contour map of it.

A traversal that visits every triangle once is like a depth-first search in the graph dual to the TIN. For a TIN with *n* vertices and, hence, *O(n)* edges and triangles, depth-first search through all the triangles can be done in *O(n)* time, if a mark bit is available in every object to see if it has been visited before. So one can compute all contour lines of a given elevation in *O(n)* time: traverse the whole TIN and for every triangle, determine if it contains a segment of the contour lines. If so, report it. One shortcoming of this algorithm is that it gives the segments of the contour lines in an arbitrary order.

When the connectedness of these line segments is relevant, the connected components of line segments in the contour line, the contour components, can also be obtained with a query time *O(log n+k)*, where *k* is the total number of segments retrieved [6].

Path on terrains

An *Euclidean shortest path* between s and t is defined to be a path with minimum Euclidean length among all posible path joining s and t that line on the surface of the TIN. For a TIN in which each triangle has an associated positive weight indicating the cost of travel in that triangle (for example depending on slope), a *weighted shortest path* between s and t is defined to be a path with minimum cost among all posible path joining s and t that line on the surface of the TIN. The cost of the path is the sum of the lengths of all segments multiplied by the corresponding triangle weight.

The problem of computing a weighted shortest path between two points s and t on a TIN arises in numerous applications in areas such as robotics, traffic control, search and rescue, water flor analysis, road design, navigation, routing, geographical information systems. Since in these applications approximate real surfaces are used, an approximate path will typically suffice.

Let n be the number of vertices of the TIN. Sharir and Schorr first proposed an $O(n^3 \log n)$ time algorithm to find the exact shortest path on a convex surface. The algorithm exploits the property that a shortest path on a terrain "unfolds" intoa straight line. Mitchell et al. have obtained an $O(n^2 \log n)$ time algorithm for general polyhedra by developing a continuous Dijkstra method for propagating the shortest path map over a surface. While Chen and Han [2] subsequently improved this to an $O(n^2)$-time algorithm. The best known algorithm for producing the optimal solution is of Kapoor. The algorithm uses a wavefront propagation method and runs in $O(n \log^2 n)$ time.

On the other hand, some other work has been done on the approximate shortest path. This has involved less computation times than the exact approaches, but it is a tradeoff with the path accuracy. In general, the term $(1+\varepsilon)$-*approx.* is usually used for evaluating the computation time for these approximate approaches, meaning that a calculated approximate path guarantees the accuracy within $(1+\varepsilon)$ times longer than the exact path. In this evaluation, smaller ε results in better accuracy.

Some practical methods for finding the approximate shortest path based on subdivided discrete graph searching ave recently been proposed. The common characteristic of these approaches is to create a certain graph, called a *pathnet*, which is needed for path computation as a pre-processing step. The $O(n \log n)$-time approach by Lanthier et al. first adds some intermediate points to edges, and discrete graph G is created by using these points and the original vertices. Then Dijkstra's algorithm is applied to G. However, this algorithm is difficult to implement.

Other problems

What about a shortest path:

- that stays below a certain elevation?
- doesn't involve large variation in elevation?

Watershed algorithms

The watershed of a river, is the set of points on a terrain which *drain* into the river, where drain is taken to mean follow the path of steepest descent.

Given a point or a segment on a river, we adress the problem of computing the watershed boundaries from a TIN for that point or segement.

Several algorithms have been proposed for finding watershed boundaries under the standard assumptions of steepest-descent flow. In particular there exist an algorithm that works in $O(n^3)$ time, where n is the number of vertices in the TIN.

Compresión and decompression of a TIN

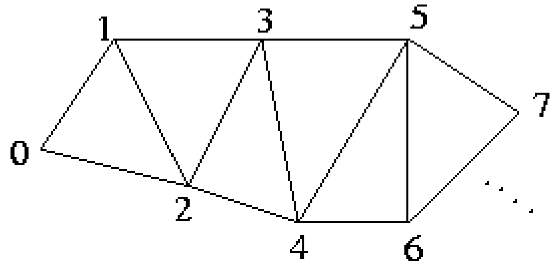
The problem of TIN compression involves two complementary tasks, which can be studied independently:

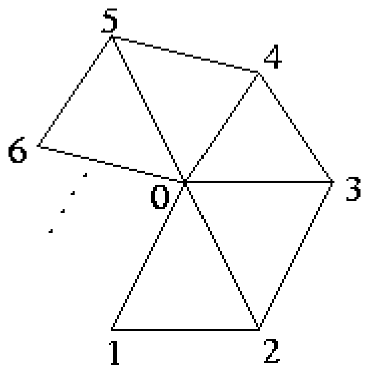
- Compression of the vertices, i.e., of the numerical information attached to each vertex of the TIN (location, elevation and, possibly, attributes).

- Compression of connectivity, i.e., of information describing the tri- angles of the TIN as triplets of vertices, and of adjacency informa tion between pairs of triangles.

Direct compression methods are aimed at efficient storage/transmission of the TIN ``as is''. Here, the goal is minimizing the number of bits needed to encode connectivity. The bitstream must be read and processed completely by a decompression algorithm, in order to obtain an explicit data structure for the TIN.

Methods proposed in the literature are based on the decomposition of a TIN into triangle strips and fans. A strip or a fan consists of a sequence of vertices: each triangle in a strip has its vertices at three consecutive positions; each triangle in a fan has its vertices at the first position of the sequence, and at two other consecutive. Generalizad triangle strips are concatenations of triangle strips and fans.





a) triangle strip; **b) triangle fan.**

Methods based on triangle strips and fans are especially convenient when applied to surface rendering. They are less suitable to be used for compressing TINs for GIS applications, since adjacency information for the triangles is not provided (except within the same strip) and must be reconstructed separately with extra processing.

Hierarchical models

The *resolution* of a TIN refers to the density of its triangles, and can be either uniform, or variable across the domain. The *accuracy* of a TIN refers to the error made in representing a terrain, and can be related to the either geometric measures (e.g., elevation, slope). A better accuracy in the approximation is generally achieved through a finer resolution, thus implying larger storage costs and higher computation times. But not all tasks within an application need necessarily the same accuracy, and, even within a single task, the required accuracy may be variable with spatial location and/or time. For instance, in landscape visualization, the accuracy needed in the various parts of a terrain depends on their distance from the viewpoint, whose position may change over time. For a close look, the resolution should be high; for a far viewpoint, the resolution could be very low.

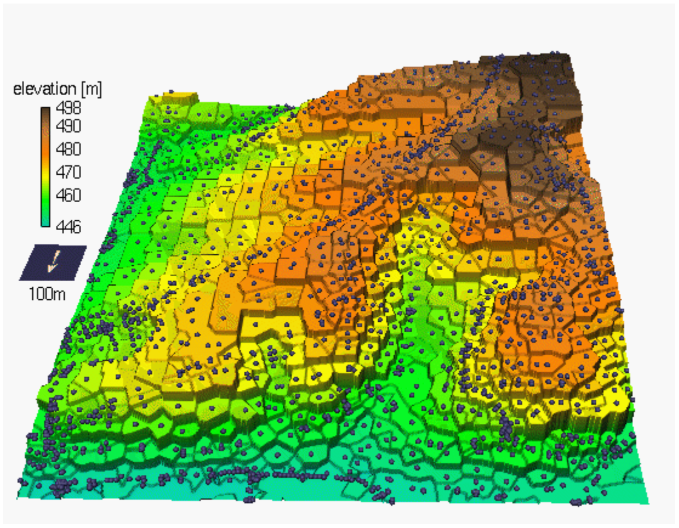
An early solution is LOD (Level of Detail). Representations with different resolutions are precomputed and stored. Then at runtime, the system chooses the resolution according to how far the viewpoint is from the terrain. View-dependent multi-resolution representation allows variant resolutions across one model, and for each viewpoint, the most economical and appropriate combination of resolutions will be computed.

A *Multi-Triangulation* (MT) is composed of a base TIN at a coarse resolution plus a partially ordered set of updates that can be applied to progressively refine the base TIN into a TIN at the full resolution. A range of TINs at intermediate resolutions can be extracted from an MT by applying a subset of its updates, while respecting the partial order.

Voronoi diagram

What is the analog of the Voronoi diagram on a terrain when:

- distance is computed on the terrain?
- distance incorporates costs for slope?



References

[1] M. Van Kreveld,
Digital Elevation models and TIN algorithms,
Algorithmic Foundations of Geographic Information Systems, number 1340 in Lecture Notes in Computer Science, pp 37-78, Springer-Verlag, 1997

[2] M. Van Kreveld,
Algorithms for triangulated terrains,

[3] M. Van Kreveld,
On Quality Path on Polyhedral Terrains,

[4] M. de Berg, M. van Kreveld, R. van Oostrum, and M. Overmars.
Simple traversal of a subdivision without extra storage,
International Journal of Geographic Information Systems, Vol. 11, pp 359-373, 1997.

[5] M. de Berg,
Visualization of TINs,
Algorithmic Foundations of Geographic Information Systems, number 1340 in Lecture Notes in Computer Science, pp 79-97, Springer-Verlag, 1997.

[6] M. van Kreveld.
Efficient methods for isoline extraction from a TIN.
International Journal of GIS, 10:523--540, 1996.

Terrains Public Software:

http://graphics.cs.uiuc.edu/~garland/software.html