Chapter 6.
Merging Spatial Data Representations

Spatial databases come in a variety of flavors, employing nearly as many different data representations as there are uses for the data. In cartographic systems, for example, vector databases support the development of customized map products and analytic functions such as the measuring of distance and area. Systems that focus on the processing and display of imagery depend on raster data structures. For systems like civil engineering packages that provide three-dimensional visualization, triangulated irregular networks (TIN) and constructive solid geometry (CSG) models represent terrain and other objects. Past work has produced numerous data structures that effectively support one or more of the various applications of spatial data. Many of these are summarized in surveys such as [NW79, SW88, Cla90, Nie90, EH91, Gat91, RK91, WH91].

Yet as Guenther and Buchmann observed [GB90], one of the challenges still confronting spatial database builders is how to integrate these different representations. Because different spatial data representations are designed to serve different purposes, relying on only one representation precludes the ability to support some functions. As noted by Nievergelt [Nie90], failing to take advantage of the rich spatial structures can adversely affect performance of spatial operations.

Rather than force all data to conform to one data representation, methods are needed for merging data representations in real time. These methods should allow applications to make use of all capabilities without requiring the casual user to be cognizant of any differences in data representation.

The objective of the work described in this chapter is to develop techniques for doing just that. Numerous applications require these three-dimensional models to be integrated with cultural information in vector form and pictorial information in raster form. In response to this need I developed techniques for merging my adaptive hierarchical triangulations with vector and raster data. Although the focus of this discussion is on cartographic systems, other applications using spatial data may use similar structures to achieve their particular goals.

My approach is to exploit the advantages of each data structure, designing algorithms that perform well in reality as well as asymptotically. Building upon past research in the areas of computational geometry and geographic information systems, I have developed novel techniques for rapidly merging a hierarchical triangulation with raster representations, other triangulations, and vector features. The primary theoretical contribution of this chapter is the introduction of my polygonal sweep method. This technique can significantly reduce the time it takes to find all polygonal patches inside or crossing an area boundary without restrictions.

Together, these techniques form the basis of a toolkit for merging two- and three-dimensional spatial data representations in real time. My hope is that this will begin to open a new world of possibilities to those applications that have had to settle for one data representation at the expense of another.

6.1. Background

Much is already known about the data representations that I selected for merging. In this section I provide a brief overview of some of these known quantities. First I review characteristics of the two-dimensional data being merged with the adaptive hierarchical triangulation. Then I discuss past work on merging data. Finally I outline past work on spatial operations done in the areas of computational geometry and geographic information systems.

6.1.1. 2D spatial data representations

Users and builders of geographic information systems (GIS) have argued for well over a decade over which of two representations is best suited for GIS: vector or raster [NW79, Oos89, Goo91]. Yet as three-dimensional visualization and analysis gain importance in spatial applications, still another structure has gained acceptance in the world of GIS: the triangulated irregular network (TIN). The truth of the matter is that all three structures have undeniable advantages when it comes to the applications that they were designed to accommodate. TIN structures have already been discussed at length in earlier chapters of this dissertation. Characteristics, advantages, and disadvantages of vector and raster data representations are discussed below.

6.1.1.1. Vector

Of the three data structures that I am merging, vector representations are most commonly used in current geographic information systems (GIS). Systems relying on this data representation — including ARC/INFO, GRASS, GeoVision, and Genamap — typically emphasize analytical and map-creation capabilities [Moo85, GIS89].

Vector feature representations use topological graph structures — composed of nodes, edges, and faces — to depict spatial entities and their relationships. Features that reference this topological graph are attributed with spatial (e.g. height, width) and non-spatial (e.g. name, surface material) characteristics.

This representation has several advantages. First, it is well suited for analytical functions such as graph problems (network flow, single-source shortest path), topological relationships (adjacency, inclusion), and algebraic functions (distance, area). Second, it is affine transformation invariant, representing absolute ground coordinates and elevation values. Third, they efficiently describe graphic entities such as points, lines and areas.

Yet the simplicity that makes vector feature representations ideal for two-dimensional map applications also makes them cumbersome for other applications. For example, displays created from these features cannot easily represent greater pictorial detail or higher dimensionality without great difficulty. They are cumbersome at best in applications such as pictorial display, image processing, and three-dimensional modeling.

Digital vector data sources include the U.S. Geological Survey's Digital Line Graphs (USGS DLG), Defense Mapping Agency's Digital Feature Analysis Data (DMA DFAD), and Digital Chart of the World (DCW). Yet many groups still get their vector data by tracing it off of charts and images, so data entry is still a major part of most vector GIS.

6.1.1.2. Raster

Raster data partitions a picture plane or other surface into a regular grid of pixels or cells, each of which is assigned a value. This value may represent nearly anything including color, intensity, elevation, or surface material code. Raster data representations form the foundation of those GIS that emphasize image processing capabilities applied to photographs and digitized maps. Erdas, MapBox, and ER Mapper, for example, are raster-based GIS [GIS89].

Raster systems boast numerous advantages due to their regularity. Image processing functions, for example, rely on this regular representation. Displaying raster structures on raster displays requires little or no transformation. Because raster cells may contain any type of value, raster structures are also used to store surface models. The advantage of this is that the regularly spaced elevation posts have implied ground positions which need not be stored.

Yet this regularity is also the source of several disadvantages. By their nature raster structures are resolution- and transformation-dependent. This leads to aliasing and lost information when resolutions are too coarse, and unnecessarily large data volumes when resolutions are too fine. In addition, raster structures

do not lend themselves as well to graph, topological, or algebraic analysis functions as vector representations do.

Digital raster data sources include DMA Digital Terrain Elevation Data (DTED), USGS Digital Elevation Models (DEM), and a wide variety of scanned charts and images (e.g. SPOT and Landsat satellite images).

6.1.2. Past work on merging data structures

Although some work has already been done on the merging of data structures, much of this focuses on merging data using similar structures. Raster structures may be merged together trivially using boolean operations or algebraic and image processing functions such as averaging [Pra78]. In his PhD thesis, van Oosteram discusses merging vector representations using reactive data structures [Oos89]. Others have described methods for merging data represented by quadtrees [HS79, SW88a, HS91]. Guibas and Seidel [GS87] invented a clever method for merging two triangulations, which is discussed further below.

Some work has also been done to merge vector and raster structures together. Gupta et al. describe an extended object-oriented data model for large image bases [GWJ91] that defines four different viewing planes showing the inter-relationships among vector and raster data structures. Zhou and Garner are investigating methods of merging data from a vector GIS (ARC/INFO) with an in-house raster database containing spatial index links [ZG91]. Yet in most cases this is achieved by translating vector to raster information, or raster to vector [LB87, GS89, ESRI90, Li93]. Clarke [Cla90] provides a thorough survey of map data structure transformations and the issues involved.

Little work has been done to merge two-dimensional and three-dimensional data structures. Most algorithms in this area focus on texture-mapping (e.g. [W83]). Using the filtering capabilities of multi-level structures for the merging of two- and three-dimensional data is a virtually unexplored area.

6.1.3. Spatial relationships

Computational geometry provides a wealth of algorithms and data structures devoted to various merging-type problems. Many of these ideas are applicable to the objectives of this research: finding relationships between different spatial entities. I outline some of the techniques most relevant to this topic below.

6.1.2.1. Inclusion problems

The point location problem may be stated as follows. Given a partition of the plane, find the region R that contains a point P. The problem of merging a triangulated model with point data in a vector representation is a special case of this problem.

Most algorithms in this area focus on defining planar partitions to aid the search. For the most part, these facilitate searches taking O(log n) time. Several of these are outlined in a paper by Preparata [Pre90]. Slab methods further partition the plane with horizontal lines passing through the vertices of the original partition. The slab containing the point may be found using two binary searches [DL76], one binary search on a trapezoid tree [Pre81], or a sweep line [ST86]. Separating-chain methods [LP77, EGS86] generate chains from edges already in

the planar partition. Triangulation refinement [Kir83], which inspired the work described in chapter 3.1, uses a hierarchy of irregular triangulations to locate points. In addition, partition trees [Ede87] and their extensions [GOS88] may also be used to organize the space for improved performance.

6.1.2.2. Intersection problems

Sweep methods are commonly employed to rapidly find intersections among structures as well as determine point locations. The idea is that once the spatial entities have been sorted in some manner — an O(n log n) operation — the sweep line may be used to examine them in linear time. The original sweep algorithm [NP82] uses a straight line which passes over the data in question. A list of edges/regions intersecting the sweep line is maintained at all times. Edelsbrunner and Guibas extended this idea by introducing the topological sweep line [EG89] which derives benefits from its ability to bend. For example, this idea was used for reciprocal search [GS87]. Sweep lines have even been used to spatially access vector data in a GIS [KBS91].

6.1.2.3. Map generalizations

In the process of making maps, cartographers generalize features by eliminating details, simplifying shapes, and combining or displacing features that are very close to one another [Cla90]. As map scales decrease, more generalization is required. This is analogous to the zoom operation discussed in the previous chapter. With the advent of computer cartography, numerous algorithms generalizing

features have been developed. Among these are algorithms for generating and manipulating hierarchical levels of detail.

Just as the triangulation hierarchy improves performance of triangle manipulations, hierarchies of other cartographic generalizations may also be used to improve the performance of merging algorithms. Numerous hierarchical curve representations have been proposed. For example, the strip tree [Bal81], curve tree [GW90], and BLG-tree [Oos89] are binary trees that retain information for bounding each generalized curve segment. The multi-scale line tree [JA86] contains levels of detail meeting accuracy constraints, similar to the adaptive triangulation hierarchy. Area generalizations have also been described, such as the TR*-tree [SK91] and the reactive tree [Oos89].

6.2. Raster/TIN

Merging a triangulated surface with a raster representation is a special case of the more generalized problem of combining two surface meshes. When raster data represents an image, this type of merge can be used to paint that image on the surface. Raster data merging may also be used to attribute the triangulated surface patches with classification categories or additional height information. Like digital elevation models, many raster representations may be more efficiently represented as irregular triangulated networks (TIN). Therefore both capabilities are discussed in this section.

6.2.1. Problem definition

Essentially two problems are considered here:

- Given a triangulation and a raster image covering the same area as the triangulation, map the raster image pixels to all of the triangles at a specified level of detail.
- Given two triangulations of the same area, find the overlay. This is a triangulation that includes all vertices and edges from both of the original two triangulations.

6.2.2. Approach

An obvious approach to merging raster and TIN data is to use a conventional plane sweep algorithm [NP82]. A horizontal sweep line travels in incremental steps from top to bottom, each step corresponding to a scanline in the raster image. Initially the cut of the sweep line contains all triangle edges and vertices intersecting the top scanline. Pixels are transferred to the triangles intersected by the sweep line in a left-to-right order. The sweep line then sweeps down to the next scanline, updating the list of intersected triangles as necessary, and the process is repeated. The cut of the sweep line changes each time a triangle vertex is encountered. This is easily checked by stepping through a list of the vertices sorted top to bottom, left to right.

Yet some raster data use many pixels to represent large flat regions that could be generalized by a few large polygons. If the raster image may be segmented into a few very large polygons, it is best to use a variation of the area

merge described later in 6.5. If, however, the regions are better represented by a TIN — or if the raster data has already been triangulated — a better approach is to use a variation of Guibas and Seidel's reciprocal search method [GS87].

Informally, the reciprocal search problem finds all pairs (b, g) of objects such that $b \in B$, $g \in G$, and bpg for some relation p. In our case, B and G represent the two triangulations to be merged, and p is the overlay operation. Guibas and Seidel describe a solution to this problem based on topological line sweep. Their method finds the overlay by sweeping over each vertex in the overlay, where a vertex either originates from B and/or G, or is formed by the intersection of two edges, one each from B and G.

Figure 6.1 shows an example where the topological sweep line has advanced past an intersection vertex. Here, solid lines represent one triangulation while dashed lines represent the other. The heavy line is the topological sweep line.

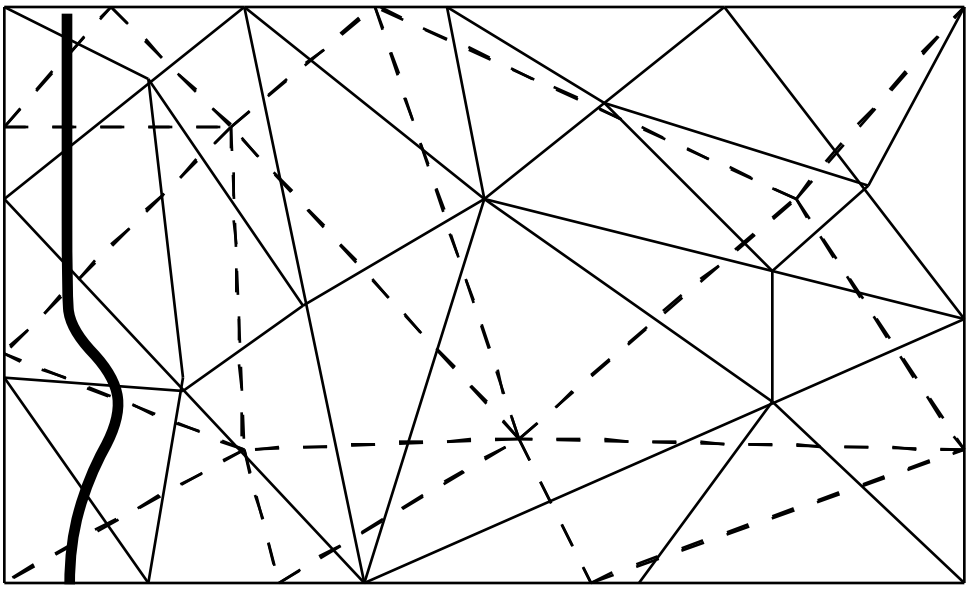

Figure 6.1. A topological line sweep bends to discover one intersection at a time

Further enhancement is required to ensure that the resulting overlay is a triangulation. I propose two possible approaches to this. The first, and most obvious, approach is to find the overlay polygons, then triangulate all those that need it using some polygon triangulation method as described in chapter 2. The trouble with this approach is that it will significantly worsen the time complexity of the algorithm.

A second approach is to generate the triangles as the sweep line advances. The cut of the sweep line is supplemented with information about each of the regions it passes through, where each region is bounded by two edges of the cut and possibly one other edge that the topological sweep line has already advanced past. Each time the sweep line advances past some vertex, all regions bounded by an edge that uses that vertex is examined. If the region has two bounding vertices that have already been swept over, then a new edge is added to form a triangle with the new vertex. This triangle is effectively cut off from the region, and is added to the list of output triangles. Regions are then updated accordingly. The advantage of this strategy is that the overhead for producing triangles is reduced to a constant overhead, with negligible impact on the asymptotic running time. The disadvantage is that this approach is not rotation invariant: results depend on the order in which vertices are advanced over.

6.2.3. Analysis

The traditional sweep line method for merging raster data with a triangulation benefits from the explicit adjacency ordering of triangles within the hierarchy. Given that N is the number of triangles in the triangulation, it takes O(N log

N) time to sort the triangle vertices (events). Therefore if M is the number of pixels in the raster structure, merging raster data with a triangulation takes O(M + N log N) time.

Guibas and Seidel show that their reciprocal search method can find the overlay of two triangulations in O(M' + N + K) time where M' is the number of triangles covering M pixels in the original raster image and K is the size of the result. If triangles are generated as the sweep line advances, this merely adds a constant overhead to each of the K output triangles with no effect on the asymptotic run time. If M' is significantly smaller than M and the raster image is triangulated beforehand, then merging two triangulations (rather than a raster image with a triangulation) could significantly improve performance of the merging operation.

6.3. Point

Finding the triangle that corresponds to a point on a surface is the most fundamental of the triangle/vector merging problems. Placing a three-dimensional point feature (e.g. a tree or building) in a perspective view of a landscape requires this capability. For many analysis functions such as line-of-sight and cross-country movement — which involve finding a path between two points — point location is an important initial step. Point location is also used in the solutions to the remaining triangle/vector merging problems.

6.3.1. Problem definition

The spatial component of a point feature is represented as a single geographically referenced point on a map. The problem of merging a point feature with the triangulated surface has two aspects:

- Find the triangle — at a specified level of detail — that the point lies within.
- Considering a point feature's width, length, and angular orientation, find the triangle(s) covered by that point feature's footprint.

6.3.2. Approach

The obvious approach is to take advantage of the triangulation hierarchy's structure and use top-down search. For a regular hierarchy — where each non-leaf node has a fixed number of children and O(log N) levels of detail — this produces a reasonable search time of O(log N) for a model with N triangles and O(log N) levels of detail. This is as good as the point location algorithms described in the computational geometry literature. Yet the adaptive hierarchical triangulation has a fixed number of levels of detail and variable numbers of children at each node. Therefore in the worst case, a top-down search may take O(N) time.

I propose an alternative point location strategy that approaches constant time in the average case. My approach is to use the query point's ground position to calculate an index for a triangle that the point is either inside of or near.

Imagine a regular grid overlaying the triangulation. Any point on the surface will correspond to a single grid cell, indexed in constant time using the ground position. I associate each grid cell with one triangle so that the triangle may be indexed in constant time given the grid cell. Although a rectangle may intersect more than one triangle, this puts us in the correct general location: if the point is not in that triangle, it is in a nearby triangle.

Figure 6.2 shows an example of my indexing strategy. Here, the number of rows/columns in the grid is m ($m^2 \leq$ N) over a triangulation of N triangles (4 in the example). Given that (x,y) is the position of the point and g is the height and width of each grid cell, the triangle index is

$((x/g) \times m) + (y/g) + 1$.

An alternative strategy is to store a pointer for each grid cell referencing the triangle that covers the largest portion of the cell. This approach is preferable in cases where triangle sizes are extremely variable.

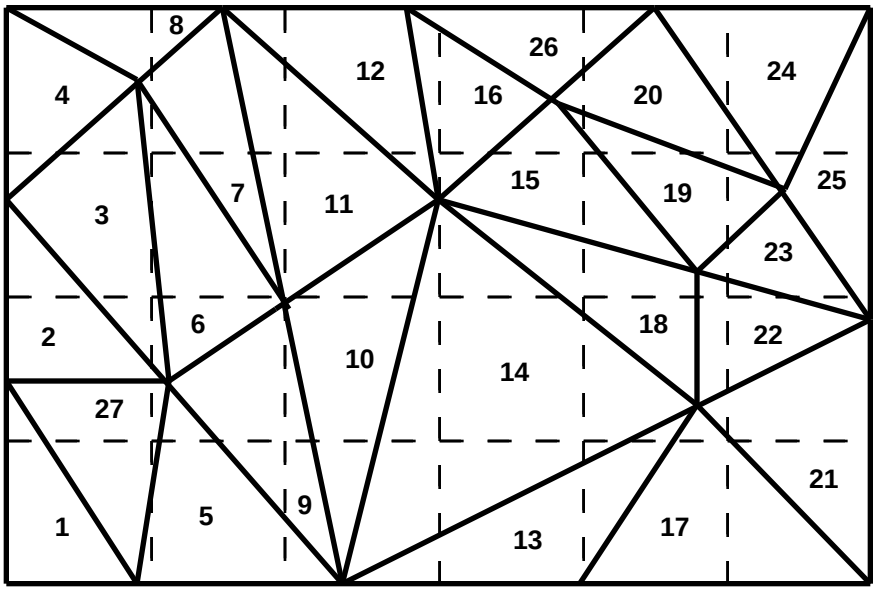

Figure 6.2. Point indexing strategy

After finding a triangle in constant time, I then determine whether the point is actually in that triangle and, if not, which neighbor it is in. There are three edges to compare the point to when determining whether the point is inside a triangle. Fortunately, this testing may be done rather efficiently. If all triangles are stored with their edges in counter-clockwise order then a point is inside a triangle if and only if it lies to the left of all edges of the triangle. This may be found with two determinant calculations [Pav82]. The determinant produces a positive value if the point (*x*,*y*) lies to the left of the edge from (*x1*,*y1*) to (*x2*,*y2*) , and a negative value if it lies to the right. The absolute value of this is distance *d* (from the point to the edge) times distance *n* (from (*x1*,*y1*) to (*x2*,*y2*)). Summing the determinants with respect to all three edges produces a value equivalent to twice the area of the triangle:

$$\det\begin{vmatrix} x1 & x2 & x3 \\ y1 & y2 & y3 \\ 1 & 1 & 1 \end{vmatrix} = \det\begin{vmatrix} x & x1 & x2 \\ y & y1 & y2 \\ 1 & 1 & 1 \end{vmatrix} + \det\begin{vmatrix} x & x2 & x3 \\ y & y2 & y3 \\ 1 & 1 & 1 \end{vmatrix} + \det\begin{vmatrix} x & x3 & x1 \\ y & y3 & y1 \\ 1 & 1 & 1 \end{vmatrix}$$

Therefore, if the area of the triangle is known, then only two determinants need be found. If both values are positive and their sum is less than or equal to twice the area, then the point lies within or on the boundary of that triangle. Otherwise, these two determinants imply which of six neighbors should be searched next, as shown in 6.3.

If a point feature's spatial dimensions are also to be considered in the point location problem, all additional triangles covered by its footprint may be found with the same determinant formula.

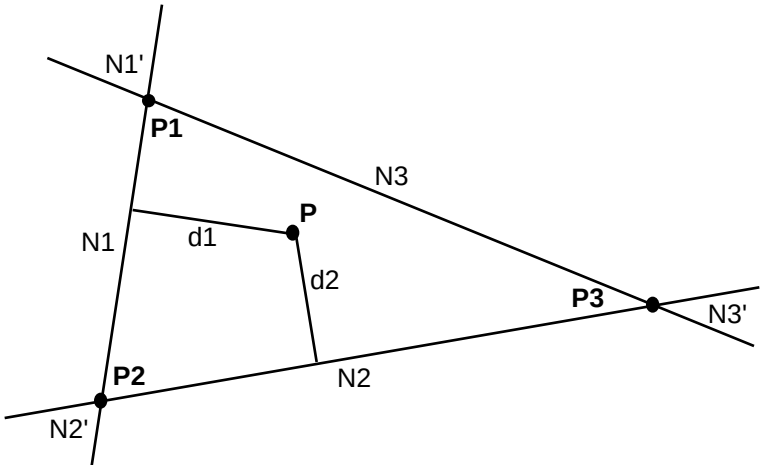

Figure 6.3. Using the determinant form reveals whether a point is inside a triangle and, if not, which neighbor to search next

6.3.3. Analysis

The following additional information is required to support this point location strategy:

- Store a single value (twice the triangle area) for each triangle, to determine inclusion with two edge comparisons. If storage space is more precious than processing time, three determinant calculations could be performed instead, negating the need for this additional information.
- Store one value for each level of detail, to be used as a divisor calculating the grid square that a point lies within. An alternative strategy is to store two values for each level of detail instead, for separate horizontal and vertical position calculations.
- If no triangle is referenced by more than one grid cell, then the triangles may be ordered such that the index of a triangle is equivalent to the index of the grid cell that refers to it. No additional information is

required. If, however, the mapping of grid cells to triangles is many-to-one, then explicit indices must be stored, one for each grid cell.

With certain restrictions imposed, this strategy may be shown to have a constant execution time on average. I use an amortized analysis, considering M uniformly distributed point location queries for a triangulation that is covered by a grid with M cells, so that each query point lands in a different grid cell. I assume that the grid has been defined such that no cell contains more than one triangle vertex. I also assume that for grid cells that contain no triangle vertices, the average number of triangles covered by a cell is $\leq K$, where K is some constant. These restrictions are easily met by adjusting the resolution of the grid.

As shown in chapter 3.2, the number of triangles T in a triangulation of $V = V_{in} + V_{out}$ vertices is

$$T = 2V - V_{out} - 2.$$

Applying this to Euler's formula for a planar graph yields

$$\begin{aligned} E &= V + T - 1 \\ &= V + 2V - V_{out} - 2 - 1 \\ &= 3V - V_{out} - 3. \end{aligned}$$

Because each edge connects exactly 2 vertices, the average number of edges — and triangles — adjacent to a vertex is less than 6.

Now imagine a grid cell containing one vertex on the triangulation with 6 adjacencies. In the worst case, 5 of those triangles will need to be examined before the correct triangle (containing the query point) is found. Therefore I assign an amortized cost of 5. Furthermore, I constrain K (limit on the average number of triangles covered by a grid cell not containing a vertex) to be 5.

With these conditions in place, the total amortized cost of M point location queries — 5M — is an upper bound on the total actual cost. The average search time is therefore O(1).

6.4. Line

Lineal features are often associated with features on the earth's surface, such as rivers and roads. Merged with a triangulated surface, these lineal features may be painted on the surface as it is, or used to mark places where the feature needs to be carved out more carefully. Merging lines with TINs may also be used to assign weights to paths based on the surface slope and/or material, or do topographic consistency checking (i.e. asking questions such as "is the river running uphill?").

6.4.1. Problem definition

Line features are homeomorphs composed of successive points implicitly connected by straight line segments (edges). Merging lineal features with triangulated surface models translates to the following problems:

- Find the triangles that the line travels through.
- Partition the line such that each resulting segment represents a homeomorph inside exactly one triangle.
- Given that the line feature has an associated width, find all triangles intersected by the widened line.

6.4.2. Approach

The obvious approach is to find the triangle that the starting point lies in, then travel along all M edges of the line feature. Each time the line enters a new triangle, record that triangle and/or partition the line appropriately. If the line travels through N triangles, this will take O(N+M) time.

The trouble with this approach is that if the line feature contains a great deal of detail, the algorithm could spend a lot of time wandering around inside a single triangle. As with the triangulations, performance of this operation may be improved by introducing a hierarchy of generalizations for the line feature.

I propose to do the following. Consider the multi-scale line tree hierarchy [JA86] where each level of detail is approximated by line segments linking the most critical points for that level. At the coarsest level, this is a single line segment connecting the two endpoints of the homeomorph. Each generalized line

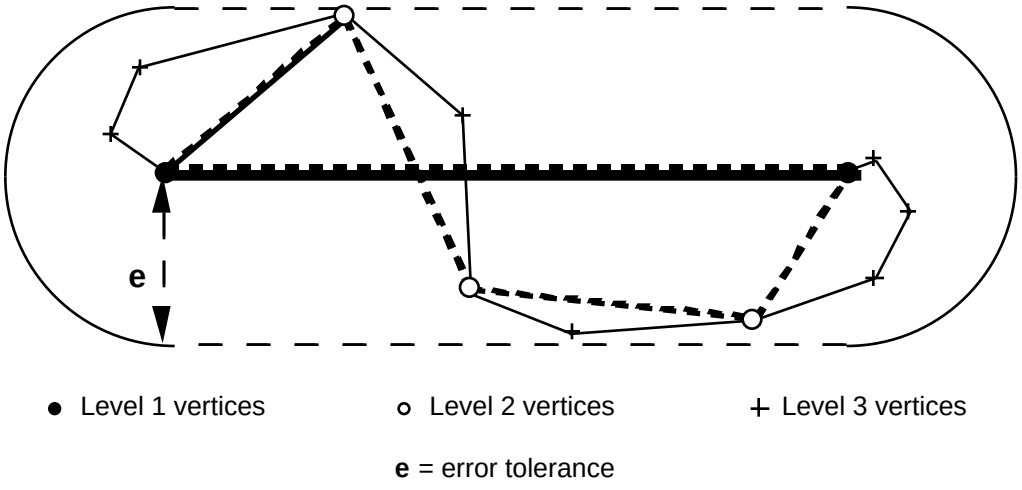

Figure 6.4. In a multi-scale line tree the error *e* at each level of detail defines a polygon that wholly contains the portion of the line being generalized

segment has an associated error tolerance e_i which guarantees the goodness of fit at level of detail i. If one were to inscribe a polygon around all points within e_i of the line segment, one would get a convex polygon that completely contains the portion of the homeomorph that the line segment is meant to generalize. This is illustrated in figure 6.4.

It is this inscribed polygon that helps to find the general solution to the problem. Because the polygon is convex — and, of course, all triangles are convex — the homeomorph may travel through a triangle if and only if its bounding error polygon overlaps that triangle. In other words, the only triangles that may be intersected by the line are those in the set of triangles $[t_1..t_n]$ covered by the ellipse defined by e .

I find the sequence of triangles intersected by such a line feature in the following manner. First, find the triangle containing the first endpoint using the point merge algorithm described above. This is the first triangle in the sequence. Second, considering the error tolerance for that edge, determine whether or not the line feature may extend outside that triangle. If not, then the entire line is within that one triangle, and I am done. Otherwise, if the next endpoint is in a neighboring triangle and the line may not pass through any other triangles, then add the second triangle to the sequence: I am done. Otherwise, I must consider the next level of detail in the lineal feature. This is repeated recursively until I’ve found all line segments inside one triangle, spanning two neighbors, or at the leaf level. If a leaf level in the line tree is reached, all triangles traversed by that edge are added to the list.

If line width is a consideration, simply add the width to the error e to extend the bounds of the ellipse.

To segment the lineal feature, one must go to the leaf level to get the actual edge that passes from one triangle to the next. This may be done with a binary search method applied to successively finer levels of detail.

6.4.3. Analysis

The following additional information is required for this algorithm:

- A multi-scale line tree stores the line generalization. Note that this structure is extremely useful for many other applications involving line features. It is better than binary representations because each level in the hierarchy represents a single generalization with a guaranteed maximum error.
- I may store either one error tolerance per level of detail, or one error tolerance per generalized line segment. The latter option is more accurate, but takes much more storage space. Note that error = 0 at the leaf level.

Finding the triangles intersected by the line in this manner requires examining all N intersected triangles, but not necessarily all M line segments. Therefore this algorithm should take O(N) time as opposed to O(N + M) time for the obvious approach. Of course if all M line segments require examining anyway, this method does incur some overhead for traversing the levels of detail. Yet because the levels of detail in a multi-scale line tree are fixed — like those in the adaptive triangulation hierarchy — this is a constant overhead. With the additional task of segmenting the lineal feature where it crosses triangle boundaries, the algorithm should take O(N log M) time.

6.5. Area

Areal features delineate regions characterized by attributes such as surface material, surface configuration, or use. Areas mapped onto a triangulation may help to determine rendering parameters for radar simulation or image synthesis, assign weighting factors for cross-country movement, or do topographic consistency checking (i.e. asking questions such as “does this lake cover a steeply sloped region?"). This merging technique is also important for multi-resolution viewing (as described in chapter 5) and combining information for landscape architecture [Erv92].

6.5.1. Problem definition

Areal features are simple polygons defined by a chain of points implicitly connected by straight line segments (edges). The merging problem for this vector type has two parts:

- Find the triangles that the area boundary intersects.
- Find all triangles inside the area.

6.5.2. Approach

The obvious approach to this problem is to use a variation of polygon edge or scan filling [Pav82] which is comparable to the traditional line sweep algorithm [Nie82]. If there are N triangles inside the area, this takes O(N) time.

For large N, it would be better to find all N triangles without having to examine each one. Performance could vastly improve with a strategy that takes advantage of the triangulation hierarchy where the following is true: if a triangle from a coarse level of detail lies entirely within the area, then because all of its children (i.e. triangles at finer levels of detail) lie within it, all of its children must also lie entirely within the area.

My solution to this problem is a polygonal sweep line, a novel variation of the topological line sweep [EG89]. Initially this sweep line corresponds to the polygonal boundary of the area being merged with the triangulations. The idea is to shrink the polygon by moving the sweep line over interior points, placing them on the outside of the sweep line. Because the sweep line passes over the finite set

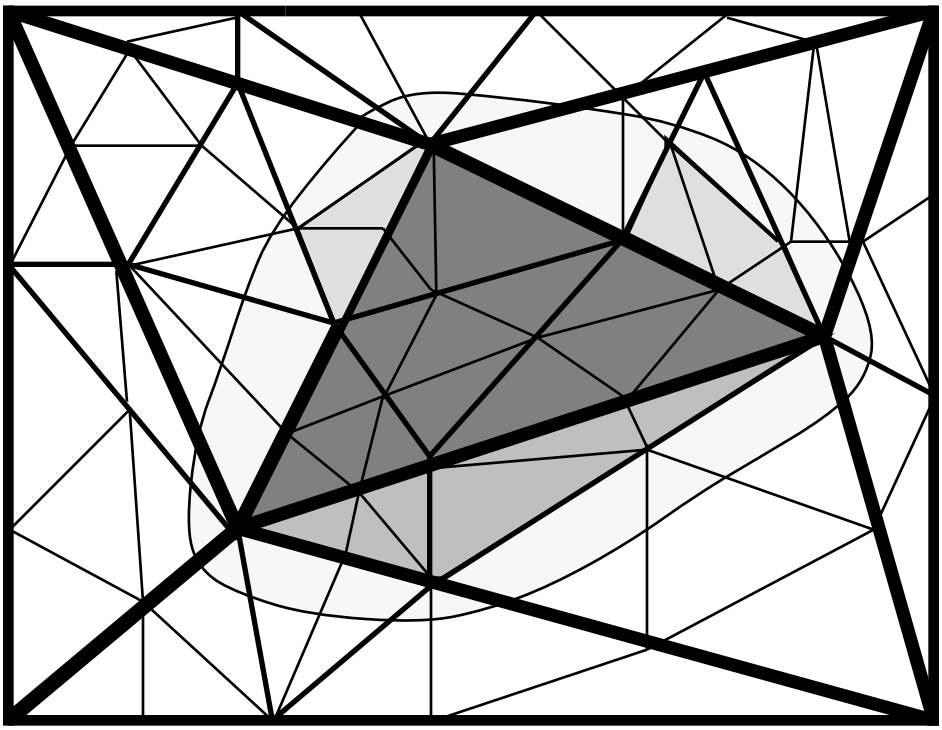

Figure 6.5. Interior triangles are found with a polygonal line sweep

of interior points only once, execution time for the algorithm is bounded by the number of interior points. If a coarser-level triangle is interior to the area, points inside of it need not be searched at all, leading to tremendous time savings.

Figure 6.5 shows an example. In this picture, darker lines represent edges in the coarser levels of detail (which, of course, persist through all finer levels of detail). Shading shows triangles intersected by or inside the area, with darker regions depicting triangles from coarser levels of detail. The polygonal sweep line need only find the points outside and bounding these larger triangles, for it is known that all vertices inside an interior triangle must also be inside the area.

My algorithm works as follows. First, find the triangles intersected by the area boundary using my line intersection strategy. Create a separate list of intersected ancestors, i.e. the coarser triangles intersected by the area boundary, found by following the chain of parent pointers.

Second, define the polygonal sweep line to be the area boundary. The cut of this sweep line comes from the sequence of intersected triangles found via line merging. Partition the polygon into m segments s_1 .. s_m such that each segment s_i passes through the sequence of k edges that converge at one vertex inside the polygon. This is illustrated in figure 6.6, where the interior of the polygon is shaded. Any triangle intersected by the sweep line that does not have a vertex inside the area must be intersected at least twice by that line, or fully contain the area. When this happens,

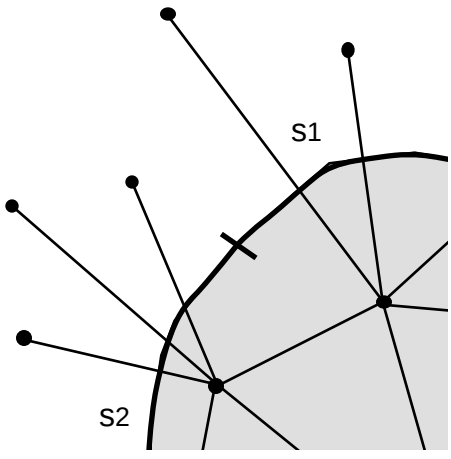

Figure 6.6. Segmenting the polygonal sweep line

the portion of the sweep line passing through that triangle is frozen in place. Note that if a triangle vertex lies on the initial sweep line, a finite number of segments may be defined at that one point.

Now the line sweeping may begin. Find an unfrozen segment s_i of the polygonal sweep line, and sweep it over an interior vertex. This moves the point to the outside of the polygon, and shrinks the polygon. This also introduces a new set of triangles, possibly requiring s_i to be split into several new segments, and possibly requiring s_{i-1} and s_{i+1} to be updated. Now examine the new set of triangles intersected by the sweep line. Any triangles in this intersection that are not on the area boundary must be inside: record those triangles in a list of interior triangles. Then examine the ancestors of those triangles. Any ancestor not on the area boundary must also be inside. Because all of its descendants must also be interior to the area, the inside of that triangle need not be searched further. Therefore that portion of the sweep line may be frozen in place. Continue until the entire sweep line is frozen — where segments have met in the same triangle or found the perimeter of the set of interior triangles.

6.5.3. Analysis

This algorithm works because it takes advantage of an adjacency ordering within the triangulation. Unlike line-thinning algorithms, the order and direction in which the vertices are swept over is irrelevant. If there are N triangle vertices inside the original area, then each vertex will be swept over by at most one segment of the polygonal sweep line. Vertices inside coarse level triangles inside the area will not be examined at all, hence the efficiency.

In the worst case, this algorithm will need to examine all N triangles anyway. Although there is the additional overhead of keeping track of ancestors, the number of ancestors is constant. Therefore this algorithm will run in O(N) time in the worst case, which is no worse than the run time of the obvious approach. In many cases this algorithm will do significantly better.