Multiresolution Tiling

David Meyers
Department of Computer Science and Engineering
University of Washington
Seattle, Washington 98195

Technical Report No. 93-12-02
December 1993

Abstract

This paper describes an efficient method for constructing a tiling between a pair of planar contours. The problem is of interest in a number of domains, including medical imaging, biological research and geological reconstructions. Our method requires $O(n)$ space and appears to require $O(n\,log(n))$ time for average inputs, compared to the $O(n^2\,log(n))$ time and $O(n^2)$ space required by the optimizing algorithm of Fuchs, Kedem and Uselton [3]. The results computed by our algorithm are in many cases very nearly the same as those of their optimizing algorithm, but at a small fraction of the computational cost. The performance improvement makes the algorithm usable for large contours in an interactive system. The use of multiresolution analysis also allows for tiling after partial reconstruction of the original contours from a low-resolution version, with controlled loss of detail. The use of lower resolution approximations to the original contours can result in significant savings in the time required to display a reconstructed object, and in the space required to store it.

Key Words: Surface reconstruction, Tiling, Meshes, Multiresolution analysis, Wavelets.

1 Introduction

Reconstruction of surfaces from a set of planar contours is an important problem in medical and biological research, geology, and other areas. The problem can be broken into several subproblems [6], one of which, the *tiling problem*, is the subject of this paper.

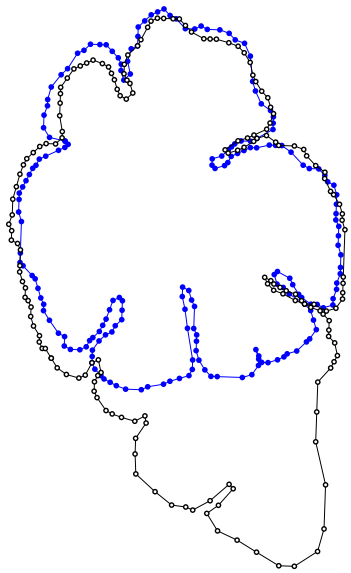

Figure 1: A pair of contours obtained from the cerebral cortex of the human brain. The contours contain 195 (closed dots) and 172 (open dots) vertices. The spacing between the contours is 1.3 mm and the larger of the contours has a short dimension of 4.6 cm and long dimension of 7.5 cm.

A solution to the tiling problem constructs a polyhedron from two planar polygons, using the polygons as two of the faces of the polyhedron, and triangles constructed from an edge of one

[1] This work was supported in part by the National Science Foundation under grant DMS-9103002

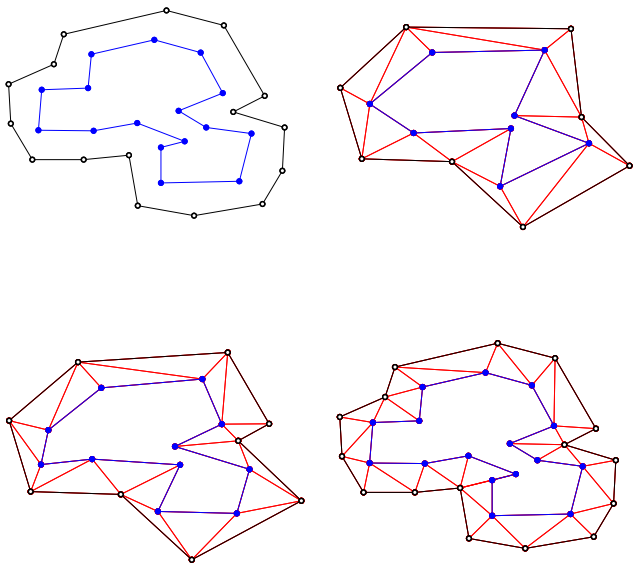

Figure 2: The main steps of the multiresolution tiling algorithm. **Upper Left:** Input contours. **Upper Right:** Tiled base case. **Lower Left:** Intermediate stage of sorted single wavelet reconstruction. **Lower Right:** Final tiling.

polygon and a vertex of the other as the remaining faces. To be a valid solution, the polyhedron constructed must be simple. Figure 2**a** and **d** show a pair of contours and a solution to the tiling problem for those contours. It has been shown by O'Rourke and Subramanian [7] that such a solution is not always possible. They show that if the shape of the polygons differs sufficiently, no simple polyhedron can be constructed subject to the above restrictions. In practice, adjacent contours are usually fairly similar in shape. There are exceptions. Consider the pair of contours shown in Figure 1. They represent adjacent slices through the cerebral cortex of the human brain. Notice that the shape of the contours differs dramatically. In cases such as the one shown in Figure 1, the shape difference can be large enough that no simple polyhedral tiling can be constructed within the definition of [7].

Numerous methods have been devised to solve the tiling problem. A method which computes a tiling optimal with respect to a goal function was devised by Keppel [4], and later improved by Fuchs, Kedem and Uselton [3]. The goal function assigns a cost to each triangle of the tiling, and minimizes the sum over the triangles in the tiling. In part because of the computational cost of their algorithm, numerous other methods have been devised which do not perform a global optimization. A discussion of many of the methods can be found in [6].

This paper describes a new method for solving the tiling problem which represents a significant improvement in speed and space required over the algorithm of [3]. Their algorithm requires $O(n^2 log(n))$ time and $O(n^2)$ space to construct a tiling for a pair of contours each of size n (if the contours are of different size, replace n^2 with $n * m$). In many cases, this performance is acceptable. However, when the number of vertices in a contour is large, the performance of the algorithm of [3] becomes unacceptable, particularly in an interactive environment. Contours containing 1000 vertices are encountered in actual data sets. With current hardware and sufficient memory, their algorithm takes approximately 2 minutes to construct a tiling from a pair of contours each with 1000 vertices. With insufficient memory, the time required increases to more than 30 minutes, something we noticed when attempting to tile 1000 vertex contours on a "normally configured" DECstation 5000/125 with 20 mbytes memory. The multiresolution tiling algorithm presented here takes about 1 second to compute a tiling for the same input, on the "normally configured" machine.

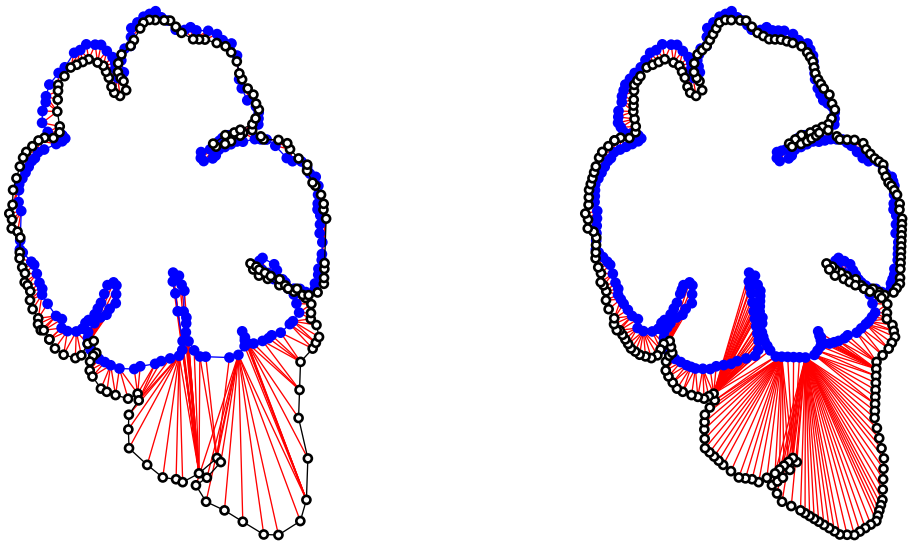

Figure 3: Tilings computed by the method of **Left:** Fuchs, Kedem and Uselton and **Right:** our sorted single wavelet algorithm, for the contours in Figure 1.

Since even the "optimal" algorithm of [3] can construct unacceptable tilings (Figure 3), user interaction is a necessary part of a system for reconstructing surfaces from a set of contours. In an interactive system, delays of the magnitude encountered with the optimizing algorithm are unacceptable, and have led to the use of faster, non-optimal methods. The algorithm we describe uses methods from multiresolution analysis to reduce the size of the contours, then uses the optimizing tiling algorithm of [3] to construct a tiling for the reduced problem size, and finally uses multires-

olution reconstruction and local optimization to construct a tiling. Our algorithm uses $O(n)$ space and what appears to be $O(n\,log(n))$ time. Although we do not prove this time bound, we show experimental results that support it.

2 Multiresolution Analysis

This section provides a brief introduction to multiresolution analysis and wavelets. For a more detailed treatment than is possible here, the reader is referred to [1] and [5]. In the following, the notation x^n is used to denote a discrete signal consisting of 2^n samples $(x_0^n, \ldots, x_{2^n-1}^n)$.

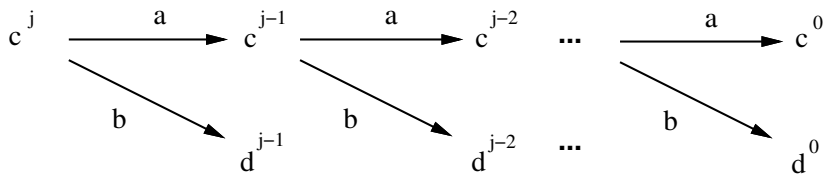

Figure 4: Filter Bank

Consider a discrete signal c^n, and let $a = \{\ldots, a_{-1}, a_0, a_1, \ldots\}$ denote a discrete low-pass filter. A low-resolution version of c^n can be obtained by convolving c^n with a (treating c^n as a periodic function), followed by selecting every other sample (often called *downsampling*). More formally c^{n-1} is obtained from c^n by

$$c_i^{n-1} = \sum_l a_{l-2i} c_l^n$$

Clearly, some detail is lost in this filtering process, since c^{n-1} contains half as many samples as c^n. If a is appropriately chosen, this lost detail can be captured as a detail signal d^{n-1}:

$$d_i^{n-1} = \sum_l b_{l-2i} c_l^n$$

where the filters a and $b = \{\ldots, b_{-1}, b_0, b_1, \ldots\}$ are called *conjugate mirror filters* or *analysis filters*. The original signal can be recovered from c^{n-1} and d^{n-1} by convolution with a pair of *synthesis filters* p and q:

$$c_i^n = \sum_l [p_{i-2l} c_l^{n-1} + q_{i-2l} d_l^{n-1}]$$

The process of computing c^{n-1} and d^{n-1} from c^n is known as *decomposition* and its inverse, recovering c^n from c^{n-1} and d^{n-1} is known as *reconstruction*. The decomposition process can be applied

recursively to c^{n-1} to form c^{n-2} and d^{n-2} and so on, forming a *filter bank*, illustrated in Figure 4. The result of applying such a filter bank to a signal c^n is the set of sequences $c^0, d^0, \ldots, d^{n-1}$.

Since the original signal can be recovered by the set of sequences $c^0, d^0, \ldots, d^{n-1}$, they can be thought of as a transform of the original signal, sometimes referred to as a *wavelet transform*. Note that the number of elements in the wavelet transform is the same as in the original sequence c^n. Use of the filter bank outlined above makes it possible to compute the wavelet transformation in linear time if the analysis and synthesis filters are of finite width.

The multiresolution analysis framework developed by Mallat [5] provides a particularly convenient framework for understanding the relationship between the analysis and synthesis filters mentioned above. Rather than starting with the filters, Mallat's idea was to associate a function $f^j(x)$ with each sequence c^j according to

$$f^j(x) = \sum_k c_k^j \phi(2^j x - k)$$

where $\phi(x)$ is a function that Mallat called a *scaling function*. Scaling functions are required to be *refinable*; that is, there must exist unique coefficients $\ldots, p_{-1}, p_0, p_1, \ldots$ such that

$$\phi(x) = \sum_k p_k \phi(2x - k)$$

As suggested by the notation, the refinement coefficients turn out to form the synthesis filter p. More formally, given a scaling function $\phi(x)$, Mallat defines an infinite set of linear spaces V^j by

$$V^j = Span(\phi(2^j - k) \mid k \in \{\ldots, -1, 0, 1, \ldots\})$$

The fact that $\phi(x)$ is refinable implies that these spaces are nested: $V^0 \subset V^1 \subset V^2 \cdots$.

By definition, the translates of the scaling function $\phi(2^j(x))$ form a basis for V^j. Let W^j denote the orthogonal complement of V^j in V^{j+1}. A wavelet is a function $\psi(x)$ with the property that translates of $\psi(2^j x)$ form a basis for W^j.

The analysis filters are formed by the coefficient sequences that make the following relation hold:

$$\phi(2x - l) = \sum_k [a_{l-2k}\phi(x - k) + b_{l-2k}\psi(x - k)]$$

Finally, the synthesis filter q is defined to be the sequence satisfying

$$\psi(x) = \sum_k q_k \phi(2x - k)$$

For multiresolution analysis of contours, we use the linear B-spline (or "*hat function*") as the scaling function $\phi(x)$. For the wavelet function $\psi(x)$ we use the "single knot wavelet" of DeRose, Lounsbery and Warren [2]. To obtain $\psi(x)$, first define $\overline{\phi}(x)$ to be the projection of $\phi(2x-1) \in V^1$ into V^0. Then define $\psi(x)$

$$\psi(x) = \phi(2x-1) - \overline{\phi}(x).$$

This definition of $\psi(x)$ has an unfortunate consequence: $\psi(x)$ has infinite support. For our purposes, it is sufficient to slightly modify the definition of $\psi(x)$ so that the support is finite. We do that by solving the projection of $\phi(2x-1)$ into V^0 for a limited number of non-zero terms. This modification has the consequence that $\psi(x)$ is no longer orthogonal to V^0. One implication of this loss of orthogonality is that the sequence c^{n-1} is no longer the best least squares approximation to c^n (see [2] for more detail). By appropriately choosing the number of non-zero terms in the projection of $\phi(2x-1)$ into V^0, orthogonality can be approached as closely as desired. Another approach to the problem of infinite support is to truncate an infinite sequence. That approach maintains orthogonality but sacrifices the ability to exactly reconstruct because of errors introduced by truncation. We choose to sacrifice orthogonality in favor of exact reconstruction.

To apply wavelet analysis to contours, we treat a contour as a sequence of knots with equally spaced values of a parameter t. Each knot has associated values of x and y. We apply the wavelet transform to x and y independently, each with respect to the parameter t.

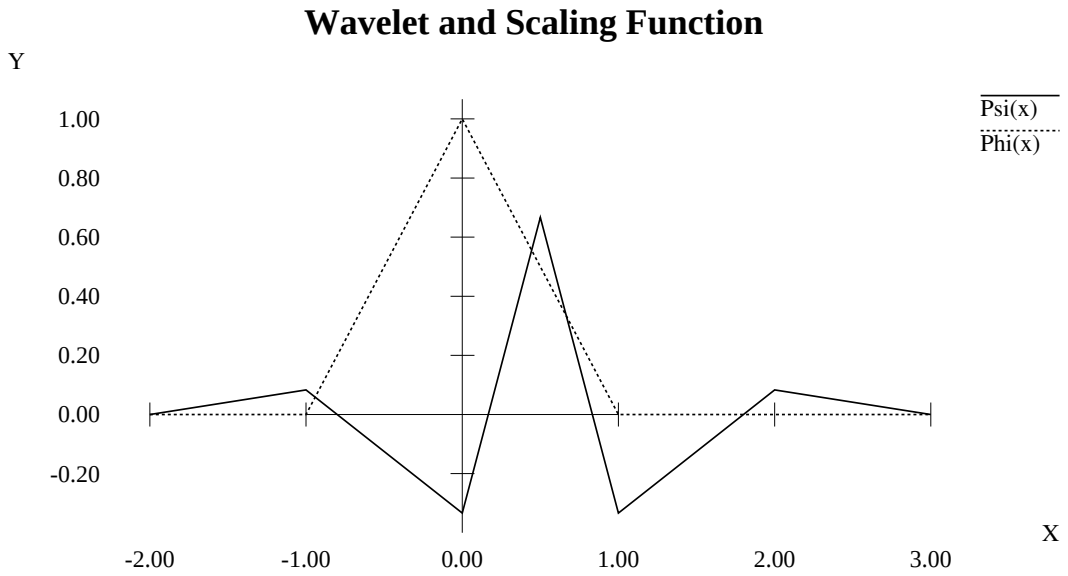

Figure 5: The single knot wavelet and linear B-spline scaling function.

i	-4	-3	-2	-1	0	1	2	3	4	5
a_i	1/24	-1/12	-1/8	1/3	2/3	1/3	-1/8	-1/12	1/24	0
b_i	0	0	0	0	-1/2	1	-1/2	0	0	0
p_i	0	0	0	1/2	1	1/2	0	0	0	0
q_i	0	1/24	1/12	-1/8	-1/3	2/3	-1/3	-1/8	1/12	1/24

Table 1: The non-zero terms of the analysis filters a and b and the synthesis filters p and q

The functions $\phi(x)$ and $\psi(x)$ are plotted in Figure 5. Table 1 shows the non-zero terms of analysis filters a and b and synthesis filters p and q.

Our choice of ϕ and ψ was motivated by the fact that a polygon of n vertices can be considered to be a piecewise linear function defined on a set of n knots. Thus, the properties of the ϕ and ψ we chose are well suited to the nature of the data with which we are working.

3 Multiresolution Tiling

Multiresolution analysis motivates a new approach to solving the tiling problem. The first step is to reduce the size of the problem by using multiresolution analysis to find low resolution approximations to the original contours (Figure 2). These low resolution contours are tiled using the method of [3]. Detail is then added to the low resolution tiling by adding wavelets, inserting edges at newly added vertices, and improving the tiling by local edge swapping. The resulting tiling is no longer guaranteed to be optimal with respect to the goal function used for computing the low resolution tiling, but can be computed much faster, particularly for contours with large n. Since the tiling begins with an optimized base case, and maintains local optimality, the final tiling constructed is often very nearly identical to that computed by the optimizing algorithm. Significant differences between the methods occur most often in areas where the pair of contours have very different shape, for example, when one contour has an indentation in an area that the other does not.

To achieve an overall speedup, the reconstruction and local optimization process must be fast. If addition of a single wavelet coefficient to the reconstruction requires as much as $O(n)$ time, the overall process will require $O(n^2)$ time. Since addition of a single wavelet coefficient to a contour can be done in constant time using the filter bank algorithm, it is only necessary to demonstrate that the additional time required for the addition of edges and local optimization of the tiling is

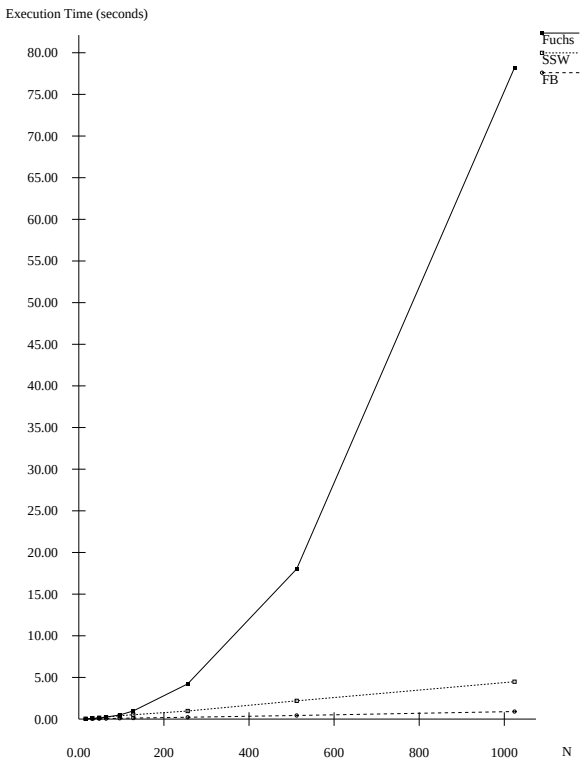

Figure 6: Execution time versus N for the Fuchs, Kedem, Uselton algorithm and the Filter bank (FB) and sorted single wavelet (SSW) multiresolution algorithms.

sufficiently small. It is possible to imagine a situation in which insertion of an edge or movement of a vertex could alter a local configuration so that a previously undesirable edge swap becomes desirable. That edge swap could conceivably allow a "cascade" of previously unswappable edges to become swappable. If such situations are common, it could take $O(n)$ time to optimize locally after addition of an edge, or movement of a vertex. Although we offer no proof of an upper bound for this process, we present data to support the assertion that for the average case it is very nearly a constant time operation to optimize a tiling locally after local modification by adding a vertex and edge or moving a vertex (Figure 6, Table 2).

3.1 Contour Decomposition

Decomposition of a contour into a set of wavelet coefficients and a lower resolution contour is done using the filter bank method described in Section 2. If the number of vertices in a contour is not a power of 2, we add vertices using the following method:

- Place the original contour edges into a priority queue based on their length.
- Remove and bisect the longest edge in the queue by adding a new vertex at its midpoint.
- Insert the two new edges into the queue.
- Repeat until the required number of vertices have been added.

Since the number of vertices in a contour is at most doubled by this process, no more than a constant factor of 2 is added to the overall complexity of computing a tiling for the resulting contour. With appropriate choice of priority queue implementation, this addition of vertices requires at most $O(n\,log(n))$ time for a contour of n vertices.

3.2 Reconstruction

The reconstruction of a contour from its low resolution version can be done using several different methods. The filter bank algorithm described in Section 2 is one. It is easy to implement, and reconstruction of a contour from its low-resolution version requires $O(n)$ time. Another method is to reconstruct by adding wavelet coefficients one at a time. This method is not as easy to implement as the filter bank algorithm, and the reconstruction of the original contour from its low-resolution version requires $O(n\,log(n))$ time, but it has some advantages over the filter bank approach, which will be discussed below. Local optimization of the tiling is done after each step of the reconstruction of the contours from their low-resolution approximations.

3.2.1 Filter Bank

Computing a tiling using the filter bank method involves the following steps: First, use the filter bank to decompose each contour to a low-resolution version. Next compute a tiling for this pair of low-resolution contours using the optimizing tiling method of [3]. Finally, reconstruct the original contours by repeating the following steps for each level of the filter bank (see Figure 7):

- Construct a new polygon for each contour using one level of the filter bank. This bisects each edge of both contours, thereby introducing a new vertex into each triangle of the tiling from the lower resolution level, so that the former triangles are now quadrilaterals, with three vertices on one of the contours and the fourth on the opposite contour.

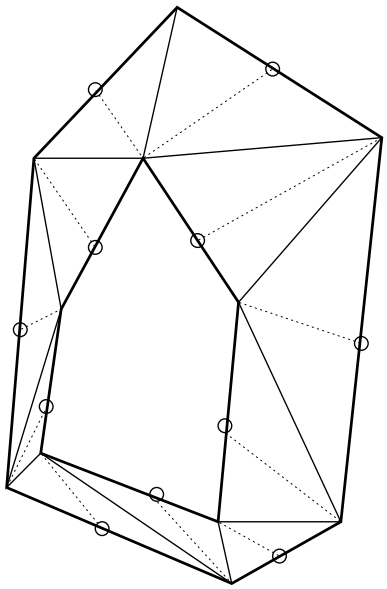

Figure 7: A tiling example, illustrating vertex and edge addition during reconstruction. Newly added vertices are open circles, newly added edges are dotted lines.

- For each new vertex added to a contour, construct an edge from that vertex to the quadrilateral vertex on the other contour, splitting the quadrilateral into 2 triangles.
- Place all the old cross edges into a list of suspect edges.
- Locally optimize the tiling as described in Section 3.3.
- Repeat until the original resolution is reached (For a contour of 2^n vertices and a low resolution contour of 2^m vertices where $m < n$, this will require $n - m$ iterations).

The filter bank method is easily implemented and the reconstruction of contours from their low-resolution versions only requires linear time. The amount of time required for local optimization of the tiling at each level of the filter bank reconstruction determines the overall cost of the algorithm. We have collected experimental results generated by using this algorithm to construct tilings for contours obtained from the human brain (Figure 6). These data suggest that the optimization step requires approximately constant time to add one vertex and edge to the tiling (Table 2), with the result that the overall cost of the filter bank tiling method is $O(n\,log(n))$ or better.

3.2.2 Single Wavelet

The filter bank reconstruction process doubles the resolution of each contour at each step, and requires that wavelet coefficients be added in the inverse of the order they were found during analysis. By adding wavelet coefficients one at a time, it is possible to use them in any desired order, and to avoid using them if their magnitude is below some threshold value. It is particularly useful to reconstruct by adding the wavelet coefficients in decreasing order of their magnitude.

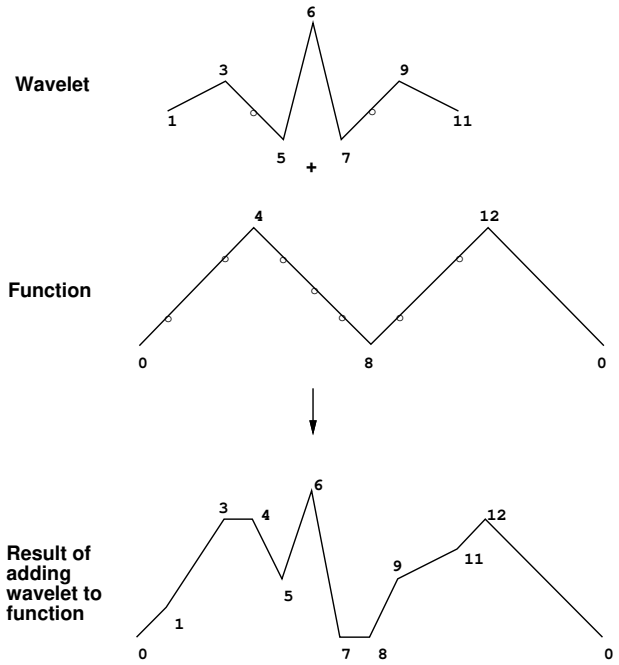

Figure 8: Illustration of Single Wavelet reconstruction in one dimension. The wavelet has intrinsic knots at t values of 1, 3, 5, 6, 7, 9, 11. The function initially has knots with t values 0, 4, 8, 12. After addition of the wavelet, the function will have knots at t values 0, 1, 3, 4, 5, 6, 7, 8, 9, 11, 12. Open circles indicate knots added to the function and wavelet for which values must be interpolated before adding the wavelet to the function.

Adding wavelets in sorted order has two benefits. First, it allows for data compression. Reconstruction using only wavelets with coefficient magnitude larger than some threshold value can reduce the number of vertices in a contour while preserving as much detailed structure as is consistent with the reduced number of vertices. Second, reconstruction by addition of wavelets in order of decreasing magnitude causes the contours to approach their original shape as rapidly as possible. Intuitively, it seems plausible that this should produce a better tiling because the local optimization process operates on a close approximation of the final shape as early as possible. In practice, this

approach seems to produce a better tiling than reconstruction using single wavelet reconstruction in the order generated by filterbank decomposition.

The initial steps in computing a tiling using the single wavelet method are the same as those in the filter bank method. Figure 8 illustrates the process of adding a wavelet to a one-dimensional function $f(t)$. For a two-dimensional contour, both the x and y coordinates of a vertex are modified by the x and y components of the wavelet coefficient. Starting from a tiled pair of low-resolution contours, the single wavelet method proceeds as follows:

- Select a wavelet to add. The method we use is to alternate contours at each iteration, and use the wavelets in descending order of the magnitude of the vector formed by their x and y coefficients.
- Determine the region of the contour influenced by the wavelet. Add knots to the contour as necessary. Any knots intrinsic to the wavelet but not already present in the contour must be added to the contour, and values for their x and y scaling function coefficients must be computed. The set of knots in the contour becomes the union of the set of knots intrinsic to the wavelet and the knots previously in the contour. If any knots are added to the contour at this step, edges must be added to split quadrilateral or larger faces into triangles (see Figure 7).
- Add knots to the wavelet if necessary. Any knots present in the region of the contour influenced by the wavelet but not intrinsic to the wavelet must be added to the wavelet, and values interpolated for x and y for the added knots. The x and y values of the intrinsic knots are computed by multiplying the wavelet function values at those knots by the wavelet coefficients. Values for knots added to the wavelet because they are in the contour are obtained by interpolating between the values at the intrinsic knots. The wavelet contains the union of its intrinsic knots and all knots of the contour within its range of influence after this step.
- Update the positions of the vertices affected by the wavelet by adding the values of x and y at the wavelet knots to the corresponding x and y values of the contour knots at each knot in the wavelet knot sequence.
- Place all edges incident on any vertex influenced by addition of the wavelet onto a list of suspect edges.

- Locally optimize the tiling by the method described in Section 3.3.
- Repeat until all wavelets have been added, or until the coefficients of the remaining wavelets are below a threshold value.

In contrast to the filter bank reconstruction, reconstruction of a polygon using this single wavelet algorithm requires $O(n\,log(n))$ time. The main reason for using single wavelet reconstruction is to add wavelets in sorted order of coefficient magnitude. Because sorting already costs $O(n\,log(n))$ time, this inefficiency relative to the filter bank reconstruction is not a major cause for concern.

3.3 Local Optimization

Local optimization of the tiling after addition of a wavelet involves identifying a subset of *suspect edges* and examining them to determine if the local geometry allows an edge swap, and if so, swapping the edge orientation if doing so reduces the goal function. The initialization of the list of suspect edges for the filter bank reconstruction method differs from that used in the single wavelet reconstruction method. Only edges connecting vertices on different contours need to be considered, since contour edges cannot be swapped. The basic idea is that edges must be examined if the connectivity or geometry has changed in their immediate surroundings.

Filter bank reconstruction proceeds in levels which double the resolution of the contour at each step. Initializing a suspects list for this reconstruction method is straightforward: all edges connecting a vertex from one contour to a vertex from the other contour in the tiling from the previous level are adjacent to a newly added edge, and are placed onto the suspects list.

Single wavelet reconstruction adds an indeterminate number of vertices to a contour at each step. The number can range from 0 to 7 in our implementation. The maximum depends on the number of non-zero terms in the analysis and reconstruction filters. If no vertices are inserted during addition of a wavelet, maintenance of a suspects list based on adjacency to new edges would not place any edges into the suspects list. Since all of the vertices within the range of the wavelet may move, this is not a good strategy. The strategy we use is to insert all edges incident on any vertex within the range of the added wavelet into the suspects list. Once the suspects list has been initialized, optimization proceeds in the same manner used for filter bank reconstruction.

After the list of suspect edges has been initialized, optimization proceeds by removing an edge from the suspects list and examining it to determine whether a swap of its orientation reduces the goal function, performing the swap if it does. If a swap is performed, edges adjacent to the swapped edge are rendered suspect, and placed onto the suspects list. The optimization process terminates when the list is empty.

The amount of time required for this local optimization is critical to the complexity of our algorithm. We have not been able to prove an upper bound for the process, but data collected in tests using contours ranging in size from 16 to 1024 vertices suggest that the number of edges examined per vertex added during reconstruction is very nearly constant for contours ranging in size from 128 to 1024 vertices. These data imply an expected performance for the filter bank reconstruction of $O(n)$ and for the single wavelet reconstruction of $O(n\,log(n))$ if wavelet coefficients are added in order of decreasing magnitude. Since addition of vertices can require $O(n\,log(n))$ time, the expected complexity implied by our data is $O(n\,log(n))$ for both the filter bank and sorted single wavelet methods.

3.4 Choice of base case size

The *base case* is a pair of low-resolution contours computed by using filter bank decomposition on the original contours. An optimal tiling is computed for the base case using the algorithm of Fuchs, Kedem and Uselton [3] in step 2 of our algorithm. The size of this base case needs to be chosen to balance overall execution time and quality of the resulting tiling. Since the base case is of constant size, its tiling can be computed in constant time.

The smallest possible base case is a pair of quadrilaterals. Reducing the original contours to this size should result in the maximum speedup of the multiresolution tiling method over that of [3]. However, the quality of tiling constructed is likely to depend on how far removed the base case is from the original contour. Constructing an initial optimal tiling from a pair of contours that contain most of the key shape features of the originals is likely to produce a better final tiling than constructing the initial tiling from a base case that contains few of the shape features of the original.

One option is to allow the user to specify the base case size. In that manner the user can make

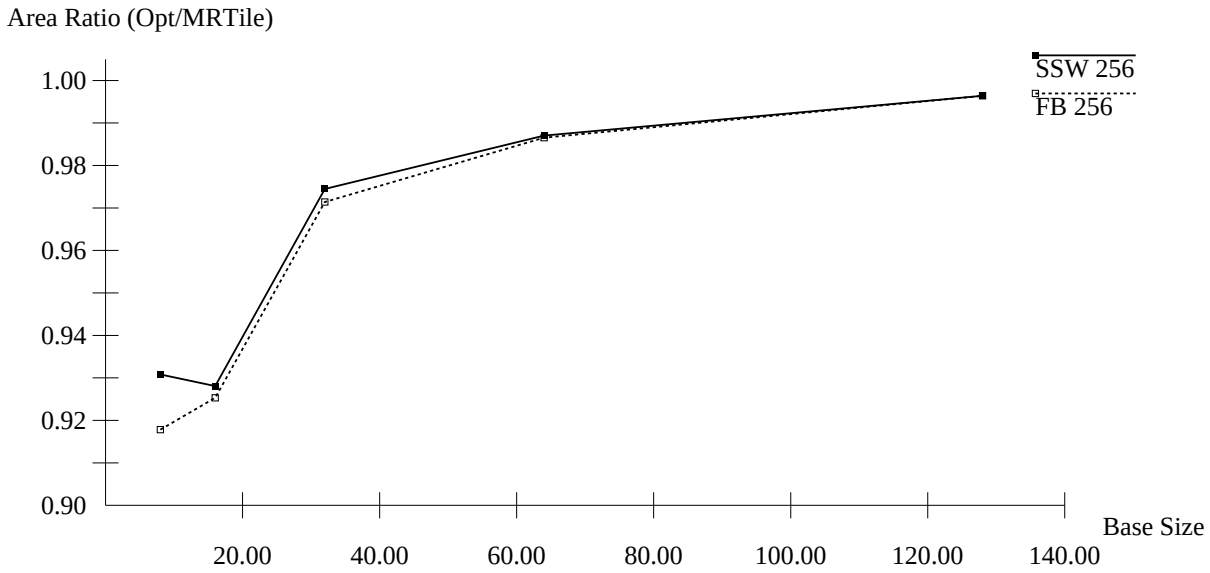

Figure 9: Tiling quality as a function of base case size for the contours of Figure 1. **SSW:** Sorted single wavelet reconstruction. **FB:** Filterbank reconstruction.

the tradeoff between acceptable tiling result and execution time. In an interactive environment, this is probably the best solution. In a non-interactive environment, a base case size of 64 seems to work well (Figure 9). For contours of that size, the execution time of the Fuchs algorithm and the sorted single wavelet algorithm are approximately equal (see Figure 6). Contours containing 64 or fewer vertices can be tiled using the Fuchs, Kedem, Uselton algorithm without significant loss of performance since a base case that size can be computed in roughly the same time it would take to reconstruct from a smaller base case.

4 Results

We have implemented both the filter bank and single wavelet reconstruction versions of the algorithm described above. To evaluate their performance we timed execution on pairs of contours obtained from the digital anatomist project, Department of Biological Structure, University of Washington. In those data, contour size ranges from 20 to over 700 vertices. Each timing run computed a tiling using the Fuchs algorithm and a tiling using one of the multiresolution methods. Various statistics were gathered by counting the number of times certain key pieces of code were executed. The resulting tilings were compared with respect to the goal function optimized by the Fuchs algorithm. The results of these tests are shown in Figures 6 and 9, and Table 2.

n	16	32	64	128	256	512	1024
FB Edges Examined	45.6	132.3	299	614	1264	2491	5136
FB Examined/n	2.84	4.14	4.68	4.80	4.94	4.86	5.02
SSW Edges Examined	275	921.3	2134	4675	9576	20113	39672
SSW Examined/n	8.59	14.40	16.67	18.26	18.70	19.64	19.38

Table 2: Number of edges examined during the optimization process for contours ranging in size from 16 to 1024 vertices by the Filter bank (FB) and Sorted single wavelet (SSW) reconstruction methods. A base case size of 8 was used.

Table 2 shows the number of edges examined during the local optimization phase of reconstruction for the sorted single wavelet and filter bank reconstruction methods. Contour size ranges from 16 to 1024 vertices. After in initial rise in the number of edges considered per contour vertex, the number per vertex remains nearly constant for contours ranging in size from 64 to 1024 vertices. These data suggest that for average inputs, a nearly constant number of edges needs to be considered when during local optimization.

Figure 6 shows the timing results obtained for each of the Fuchs, Filter Bank (FB), and Sorted Single Wavelet (SSW) algorithms using a base case size of 8. For n = 1024, with a base case size of 64 the FB algorithm is 70 times faster than the Fuchs algorithm. The Fuchs algorithm takes nearly 80 seconds of CPU time, while the FB method takes slightly over one second.

Figure 9 shows how the selection of base case size affects the quality of the tiling for the set of contours shown in Figure 1. Notice that larger base case improves tiling quality (measured as the ratio between the cost of the optimal tiling and the cost of the multiresolution tiling), and that a base case size of 64 seems to be at the point on the curve where further increase in base case size only marginally improves the final result.

Figure 3 shows the tilings produced for a pair of contours from the cerebral cortex of the human brain. The contours contain 195 and 172 vertices. The goal function value for the optimal tiling was 9.9 and that for the multi-resolution tiling was 10.7, seven percent larger than the optimum value. Notice that there are areas in both tilings that may not be acceptable. In part **a**, the tiling connects the long indentation on the lower edge of the smaller contour to the center of the edge of an indentation on the other contour, which probably is not what happens in the real object. In part **b** the indentation in the smaller contour is connected to an indentation of the larger contour, but it is unclear whether or not the tiling in that area violates the constraint that the polyhedron

produced by a tiling must be simple. Simply put, the "correct" tiling in this region is ambiguous, and depends on the nature of the material from which the contours were derived, and no algorithmic approach is likely always to yield results acceptable to a trained human user.

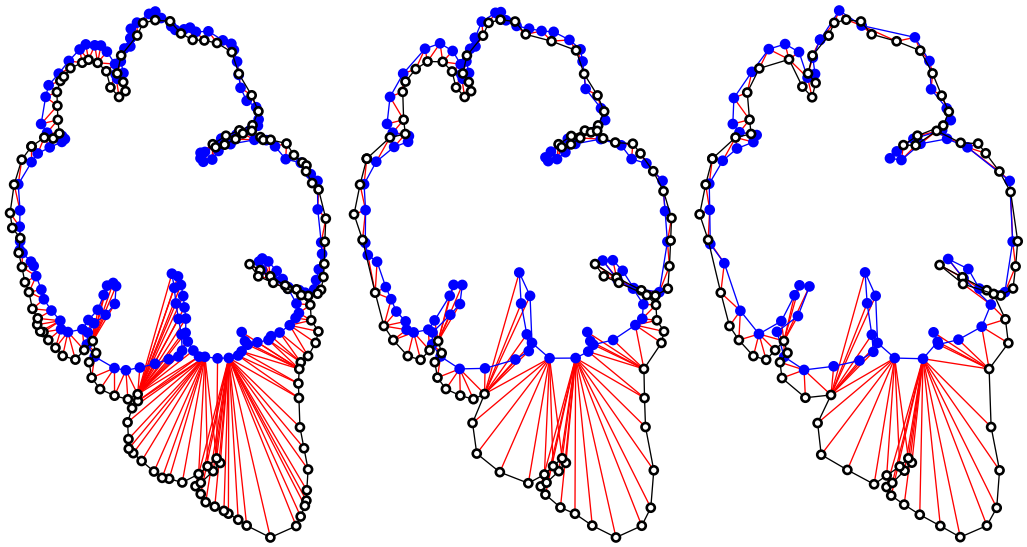

Figure 10: Tilings of the contours in Figure 1 using the sorted single wavelet algorithm with threshold values of **Left:** 0.001, **Center:** 0.0025, and **Right:** 0.005. The threshold value multiplied by the magnitude of the largest wavelet coefficient determines the magnitude of the smallest coefficient used.

Figure 10 shows a series of reconstructions of single contours using the sorted single wavelet method in which coefficients smaller than a threshold value were discarded. The number of vertices in the contours decreases significantly, while the overall shape of the contours retains much of the original detail. For many purposes, the resolution of the tiling shown in part **c** may be adequate. The low-resolution version requires significantly less space to store, and less time to display.

5 Conclusions

We have described a multiresolution approach to improving the performance of a well known algorithm for solving the tiling problem, that of Fuchs, Kedem and Uselton [3]. Their algorithm computes a tiling which is optimal with respect to a goal function that assigns a cost to each triangle in the tiling. Perhaps the most commonly used such goal function is to minimize the sum of the area of the triangles. An unfortunate problem is that this optimal algorithm requires $O(n^2 log(n))$ time and $O(n^2)$ space. In practice, these costs are too great for use in an interactive system if contours

much larger than 250 vertices are to be encountered. While testing their algorithm on contours of 1000 vertices, we found that the amount of memory required caused a machine with 20 mbytes of memory to thrash. The time required to construct a tiling was approximately 35 minutes for a pair of 1000 point contours. The method we present requires linear space, a significant improvement over the quadratic space requirement of the Fuchs algorithm. This space saving is very important if a system is to be used on computers with modest memory resources. The multiresolution method took just 1.1 seconds to compute a tiling for the same 1000 vertex contours. When run on a machine with sufficient memory to avoid thrashing (65 mbytes), the Fuchs, Kedem, Uselton algorithm took approximately 80 seconds to compute a tiling.

A problem with all known tiling algorithms is that they can produce unacceptable tilings. In fact, O'Rourke and Subramanian [7] have shown that the tiling problem is not always solvable. Problems arise when the pair of contours to be tiled differ greatly in shape. That these problems occur in real data sets can be seen by examining the pair of contours shown in Figure 1. Tilings computed for this pair of contours by the "optimal" Fuchs algorithm, and by our "Sorted Single Wavelet" algorithm are shown in Figure 3. Some areas of each tiling are probably not acceptable reconstructions of the brain tissue from which the contours were obtained. The problem is that the input is not sufficiently precise to resolve ambiguity in the problem areas, even though the spacing between the planes of the contours is only 1.3 mm.

Since unacceptable tilings can occur, a practical system for reconstructing surfaces from contours must be interactive. The computational cost of the Fuchs, Kedem, Uselton algorithm [3], both in time and space required, have caused implementors of practical systems to use other methods.

The method we present in this paper is dramatically faster than the method of [3]. It does not guarantee a globally optimal tiling, but in view of the need for user interaction in many complex cases (even when using a globally optimal algorithm), that may not be a large defect. In many cases the tilings produced by the multiresolution method are equivalent to the optimal tilings. In general, the optimal tiling differs significantly from the multiresolution results only in complex cases for which neither algorithm produces a completely acceptable result.

5.1 Future Work

When the shape of a pair of contours is significantly different, it becomes difficult or impossible to construct an acceptable tiling [7]. Some of the data sets we have used to test our methods exhibit this characteristic. User interaction therefore remains an important part of a practical system. Relaxing the specification of a solution to the tiling problem to allow inclusion of triangles in the plane of either of the sections could avoid the difficulties encountered when the shape of adjacent contours differs greatly. Detection of shape differences and their location can be used to find areas in which the tiling would be better done in the plane of a contour rather than between contours. The use of multiresolution analysis may help to make such methods fast enough to use interactively.

6 Acknowledgements

The author would like to thank Tony DeRose for the suggestion that multiple resolution analysis might be profitably applied to the tiling problem, and many helpful discussions along the way.

References

[1] Charles K. Chui. *An Introduction To Wavelets*. Academic Press, Inc., 1992.

[2] Tony D. DeRose, Michael Lounsbery, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. Technical Report 93-10-05, University of Washington, Dept. of Computer Science and Engineering, 1993.

[3] H. Fuchs, Z.M. Kedem, and S.P. Uselton. Optimal surface reconstruction from planar contours. *Communications of the ACM*, 20(10):693–702, October 1977.

[4] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM J. Res. Develop.*, 19:2–11, January 1975.

[5] Stephane Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

[6] David Meyers, Shelley Skinner, and Kenneth Sloan. Surfaces from contours. *ACM Transactions on Graphics*, 11(3):228–258, July 1992.

[7] Joseph O'Rourke and Vinita Subramanian. On reconstructing polyhedra from parallel slices. Technical Report TR # 008, Smith College Department of Computer Science, Northampton, MA 01063, June 20, 1991.