Multiresolution Tiling

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

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, based on ideas from multiresolution analysis and wavelets, requires $O(n)$ space and appears to require $O(n \log n)$ time for average inputs, compared to the $O(n^2)$ space and $O(n^2 \log n)$ time required by the optimizing algorithm of Fuchs, Kedem and Uselton [4]. The results computed by our algorithm are in many cases nearly the same as those of the 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 provides an efficient mechanism for data compression by discarding wavelet coefficients smaller than a threshold value during reconstruction. The amount of detail lost can be controlled by appropriate choice of the threshold value. The use of lower resolution approximations to the original contours yields 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 such as those shown in Figure 1 is an important problem in medical and biological research, geology, and other areas. The problem can be broken into several subproblems [8], one of which, the *tiling problem*, is the subject of this paper.

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 polygon and a vertex of the other as the remaining faces. In Figure 2, the upper left shows a pair of contours and and the lower right shows a solution to the tiling problem for those contours. To be a valid solution to the tiling problem , the polyhedron constructed must be simple. O'Rourke and Subramanian [9] have shown that such a solution is not always possible. They demonstrated that if the shapes of the contours differ sufficiently, no simple polyhedron can be constructed subject to the above restrictions. In practice, adjacent contours are usually fairly similar in shape but there are exceptions. Consider the pair of contours shown in Figure 1, representing adjacent slices through the cerebral cortex of the human brain. Notice that the shapes of the contours differ dramatically. In such cases, the shape differences may be great enough that no simple polyhedral tiling can be constructed within the standard definition of the tiling problem.

Figure 1: A pair of contours obtained from the cerebral cortex of the human brain. The contours contain 128 (closed dots) and 114 (open dots) vertices.

Numerous methods have been devised to solve the tiling problem. A method that computes a tiling optimal with respect to a certain class of goal functions was devised by Keppel [6], and later improved by Fuchs, Kedem and Uselton [4]. The goal function assigns a cost to each triangle of the tiling, and minimizes the total cost over the triangles in the tiling. In part because of the computational cost of their algorithm, numerous other methods have been devised that do not perform a global optimization. A discussion of some of the methods can be found in [8].

This paper describes a new method for solving the tiling problem that represents a significant improvement in both space and time when compared to the algorithm of Fuchs, Kedem and Uselton [4]. 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. In many cases, this performance is acceptable. However, when the number of vertices in a contour is large, the performance of the optimizing algorithm becomes unacceptable, particularly in an interactive environment. Contours containing 1000 vertices or more are encountered in actual data sets. With current hardware and sufficient memory, the optimizing algorithm takes approximately 2 minutes to construct a tiling from a pair of con-

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

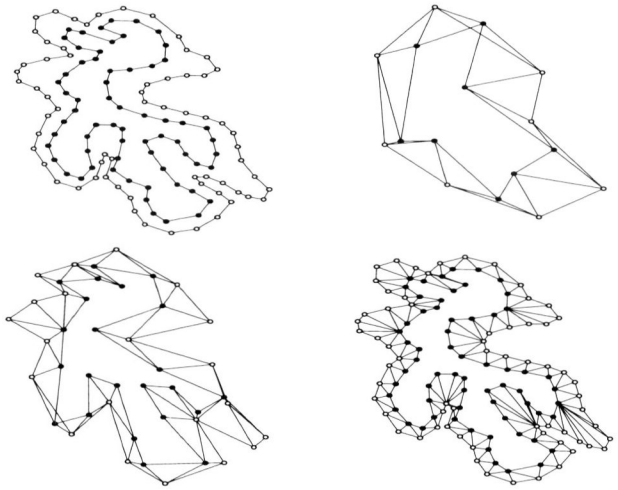

Figure 2: The main steps of the multiresolution tiling algorithm. **Upper Left:** Input contours. **Upper Right:** Tiled base-case. **Lower Left:** Intermediate stage of single-wavelet reconstruction. **Lower Right:** Final tiling.

tours each with 1000 vertices. With insufficient memory, the time required increases to more than 30 minutes, a problem we encountered when attempting to tile a pair of 1000 vertex contours on a "normally configured" DECstation 5000/125 with 20 megabytes memory. The multiresolution tiling algorithm presented here takes about 1 second to compute a tiling for the same input, on the same machine.

As Figure 3 shows, even the optimizing algorithm can construct unacceptable tilings. For that reason, 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-optimizing methods. The algorithm we describe uses methods from multiresolution analysis to reduce the size of the contours, then uses the optimizing tiling algorithm of Fuchs, Kedem and Uselton [4] to construct an optimal tiling for the reduced problem size, and finally uses multiresolution reconstruction and local optimization to construct a final 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. The reader is referred to [2] and [7] for a more detailed treatment. In the following, the notation x^n is used to denote a discrete signal consisting of 2^n samples $(x_0^n, \ldots, x_{2^n-1}^n)$.

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

Figure 3: Tilings computed by **Upper:** Our single-wavelet algorithm and **Lower:** The method of Fuchs, Kedem and Uselton, for the contours in Figure 1. Neither result would be considered acceptable by a trained anatomist.

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 *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 according to:

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

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 $c^0, d^0, \ldots, d^{n-1}$.

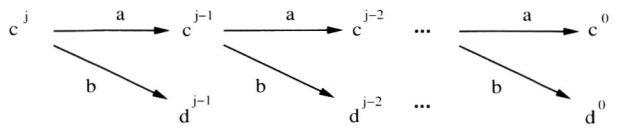

Figure 4: Filter-Bank

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 (or *support*).

The multiresolution analysis framework developed by Mallat [7] 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:

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

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 [3]. 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 $\psi(x)$ as

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

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 for 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 [3] 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 periodic 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.

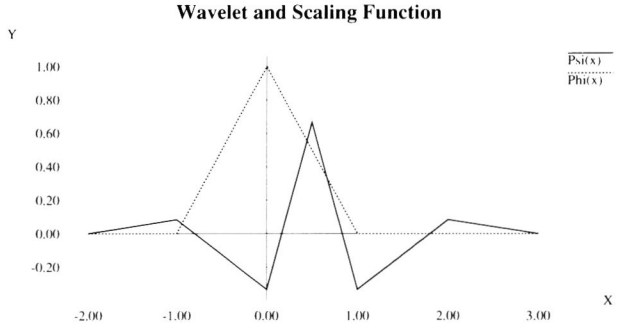

Figure 5: The single knot wavelet $\psi(x)$ and linear B-spline scaling function $\phi(x)$.

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

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

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

approximations to the original contours (Figure 2). These low resolution contours are tiled using the optimizing method of Fuchs, Kedem and Uselton [4]. 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 globally optimal with respect to the goal function used for computing the low-resolution tiling, but it can be computed much faster, particularly for contours with many vertices. 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. In such situations, it is often the case that neither method produces a result acceptable to a trained human user (see Figure 3).

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 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 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, in Section 4 we present experimental data to support the assertion that for the average case it is very nearly a constant time operation to optimize a tiling locally after adding a vertex and edge or moving a vertex (Figure 8, Figure 10).

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:

1. Place the original contour edges into a priority queue based on their length.
2. Remove and bisect the longest edge in the queue by adding a new vertex at its midpoint.
3. Insert the two new edges into the queue.
4. 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, discussed below. Local optimization of the tiling is done after each step of the reconstruction.

3.2.1 Filter-Bank Method

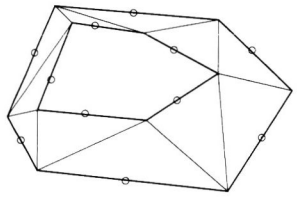

Figure 6: A tiling example, illustrating vertex and edge addition during reconstruction. Newly added vertices are open circles, newly added edges connect to a vertex of the opposite contour.

Computing a tiling using the filter-bank method involves the following steps: First, use the filter-bank to decompose each contour into a low-resolution version. Next, compute a tiling for this pair of low-resolution contours using the optimizing tiling algorithm. Finally, reconstruct the original contours by repeating the following steps for each level of the filter-bank:

1. 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.

2. 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 (see Figure 6).
3. Place all the old cross edges into a list of "suspect" edges.
4. Locally optimize the tiling as described in Section 3.3.
5. Repeat until the original resolution is reached (Requires $n - m$ iterations for a contour of 2^n vertices and a low-resolution contour of 2^m vertices).

The filter-bank method is easy to implement and reconstructing contours from their low-resolution versions requires only linear time. The cost of locally optimizing the tiling at each level of the filter bank reconstruction determines the overall cost of the algorithm. We have collected experimental results by using this algorithm to construct tilings for contours obtained from the human brain. These data suggest that optimization after addition of one vertex and edge to the tiling (Figure 10) requires approximately constant time; the overall cost of the filter-bank tiling method therefore appears to be $O(n \log n)$ (Figure 8).

3.2.2 Single-Wavelet Method

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.

Adding wavelets in decreasing 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 a better tiling should result, 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 the method of Section 3.2.1.

The initial steps in computing a tiling using the single-wavelet method are the same as those in the filter-bank method.

Figure 7 illustrates addition of a wavelet to a one-dimensional function $f(t)$. For a two-dimensional contour, the x and y coordinates of a vertex are modified respectively 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:

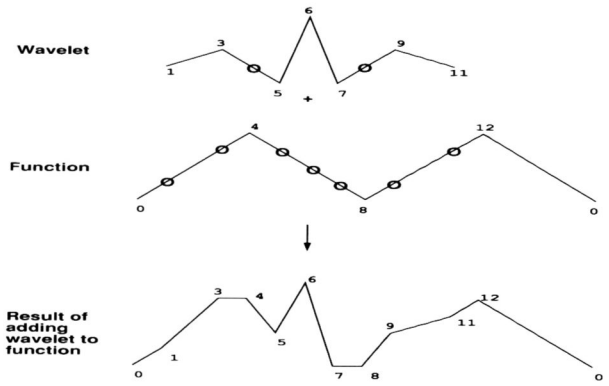

Figure 7: Illustration of Single-Wavelet reconstruction in one dimension. The wavelet has intrinsic knots at t values of 1, 3, 5, 6, 7, 9, 11. The function initially has knots with t values 0, 4, 8, 12. After addition of the wavelet, the function will have knots at t values 0, 1, 3, 4, 5, 6, 7, 8, 9, 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.

1. 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.
2. Merge the set of knots intrinsic to the wavelet and the set of knots present in the region of the contour influenced by the wavelet so that the wavelet and contour knot vectors match. After this step, both the wavelet and the region of the contour influenced by the wavelet contain the union of the intrinsic wavelet knots and the knots originally present in the region of the contour influenced by the wavelet.
3. Interpolate values for any newly inserted knots of either the wavelet or the contour. Values for knots inserted into the contour are computed by linear interpolation. Values for knots inserted into the wavelet are computed by linear interpolation after the intrinsic knot values have been scaled by the wavelet coefficient values.
4. 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.
5. Place all edges incident on any vertex influenced by addition of the wavelet onto a list of suspect edges.
6. Locally optimize the tiling by the method described in Section 3.3.
7. 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 method, 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 gain the benefits associated with adding wavelets in sorted order. Because sorting requires $O(n \log n)$ 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*, examining them to determine if the local geometry allows an edge swap, and if it does, swapping the edge orientation if doing so reduces the goal function. 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 that 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 so are placed onto the suspects list.

The initialization of the list of suspect edges for the single-wavelet reconstruction differs from that used in filter-bank reconstruction. Single-wavelet reconstruction adds a variable 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. That is not a good strategy, since any of the vertices within the range of the wavelet may have moved. The strategy we use is to insert into the suspects list all edges incident on any vertex within the range of the added wavelet. 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 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 (see Fig 10). These data imply an expected performance for the filter-bank reconstruction of $O(n)$ and for the single-wavelet reconstruction of $O(n \log n)$. Since addition of vertices to the original contour 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 single-wavelet methods.

3.4 Choice of Base-case Size

The *base-case* is a pair of low-resolution contours computed by performing a filter-bank decomposition of the original contours. An optimal tiling is computed for the base-case using the algorithm of Fuchs, Kedem and Uselton [4] 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 [4]. However, the quality of tiling constructed is likely to depend on how different the shape of the base-case is from that of the original contours. 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 the tradeoff between acceptable tiling result and execution time. In a non-interactive environment, a base-case size of 64 seems to work well (Figure 9). For contours of that size, the execution times of the Fuchs algorithm and the sorted single-wavelet algorithm are approximately equal (see Figure 8). 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", in the Department of Biological Structure, at the University of Washington. In those data, contour size ranges from 20 to over 1000 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 8, 9, and 10.

Figure 8 shows the timing results obtained for each of the Fuchs, Filter-Bank, and Single Wavelet algorithms using a base-case size of 8. For n = 1024 the Filter-Bank algorithm is 70 times faster than the Fuchs algorithm. The Fuchs algorithm takes nearly 80 seconds of CPU time, while the Filter-Bank 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 size improves tiling quality (measured as the ratio between the cost of the optimal tiling and the

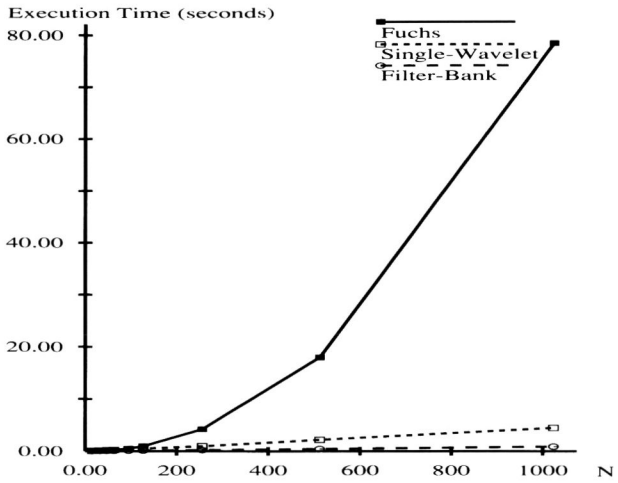

Figure 8: Execution time versus N for the Fuchs, Kedem, Uselton algorithm and the filter-bank and single-wavelet multiresolution algorithms.

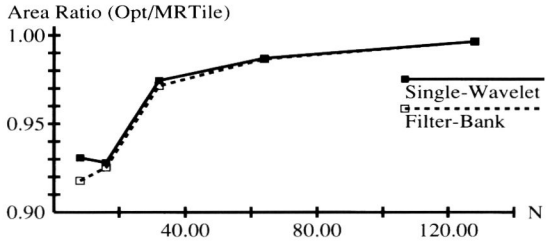

Figure 9: Tiling quality as a function of base-case size for the contours of Figure 1.

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 10 shows the number of edges examined during the local optimization phase of reconstruction for the single-wavelet and filter-bank reconstruction methods. Contour size ranges from 16 to 1024 vertices. After an 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 per contour vertex during local optimization.

The contours shown in Figure 1 represent a difficult instance of the tiling problem, obtained from the human cerebral cortex. A trained anatomist would recognize that each of the 7 indentations on each contour should be linked to a corresponding indentation on the other contour by edges at their inner extrema. Figure 3 shows tilings produced for those contours by the optimizing algorithm and by the multiresolution algorithm. Note that there are areas in both tilings that may not be acceptable according to the criterion that the indentations should be linked. The lower tiling, computed by the optimizing algorithm, con-

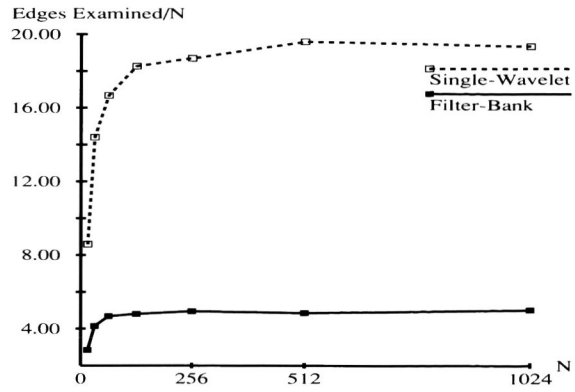

Figure 10: Number of edges examined per vertex during the optimization process for contours ranging in size from 16 to 1024 vertices by the filter-bank and single-wavelet reconstruction methods. A base-case size of 8 was used.

nects the long indentation on the right side of the smaller contour to the center of the edge of an indentation on the larger contour, which probably is not what happens in the real object. In the other tiling, computed by our single-wavelet algorithm, the indentation in the smaller contour is connected to an indentation of the larger contour, but it is unclear whether or not the "correct" connection has been found. Simply put, the "correct" tiling in this region is ambiguous, and depends on the nature of the material from which the contours were derived. No algorithm is likely to yield results always acceptable to a trained human user. In this case, the multiresolution algorithm connected 6 of 7 indentations, compared to 5 of 7 connected by the optimizing algorithm.

We computed tilings for the contours shown in Figure 1 using the linear-time "greedy" methods of Ganapathy and Dennehy [5] and of Christiansen and Sederberg [1]. Both methods construct a tiling beginning from a start vertex on each contour. They sequentially advance along either the upper or lower contour, connecting the current vertex on one contour to the next vertex on the other. The Christiansen-Sederberg algorithm attempts to minimize the sum of edge lengths by always selecting the shorter of the two possible edges at each step. The Ganapathy-Dennehy algorithm always selects the edge that minimizes the difference in normalized arc length traversed between the upper and lower contours. Figure 11 shows the results. Each of the algorithms gets "confused" by a local configuration that is not well modeled by its heuristic. The resulting tilings are significantly worse than those of either the optimizing or multiresolution algorithm.

Figure 12 shows a series of reconstructions of single contours using the single-wavelet multiresolution method. In each tiling, coefficients smaller than a threshold value were discarded. The number of vertices in the contours decreases significantly, while the overall shapes of the contours retain much of the original detail. For many purposes, the resolution of the tiling shown on the right may be adequate. The low-resolution version requires significantly less space to store, and less time to display.

Figure 11: Tilings computed by **Upper:** The algorithm of Christiansen and Sederberg and **Lower:** The algorithm of Ganapathy and Dennehy, for the contours in Figure 1. Compare to the tilings shown in Figure 3.

5 Conclusions

We have described a multiresolution approach to improving the performance of a well-known optimizing algorithm for solving the tiling problem, that of Fuchs, Kedem and Uselton [4].

A problem with all known tiling algorithms is that they can produce unacceptable tilings. For that reason, a practical system for reconstructing surfaces from contours must be interactive. The computational cost of the optimizing algorithm has caused implementors of practical systems to use linear-time "greedy" methods. The method we present in this paper is dramatically faster than the optimizing algorithm. Though it does not guarantee a globally optimal tiling, in many cases the tilings it produces 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, but for which both methods produce results superior to those of linear-time "greedy" methods. The multiresolution algorithm represents an improvement in quality over the greedy methods, and is fast enough for interactive use, even with contours containing well over 1000 vertices.

Multiresolution tiling provides a fast way to produce tilings at reduced resolution, resulting in significant savings both in time required to display a reconstruction and in the space required to store it.

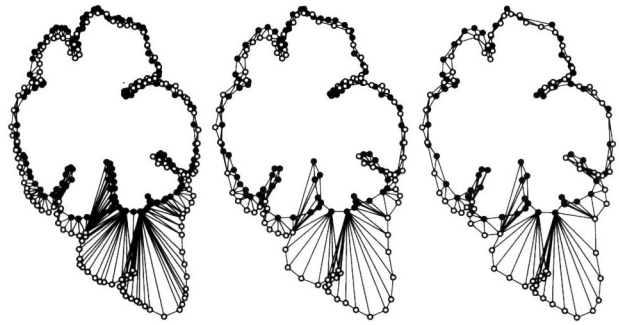

Figure 12: Tilings of the contours in Figure 1 using the single-wavelet algorithm with threshold values of **Left:** 0.001, **Center:** 0.0025, and **Right:** 0.005. The threshold value multiplied by the magnitude of the largest wavelet coefficient determines the magnitude of the smallest coefficient used.

6 Acknowledgements

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

References

[1] H.N. Christiansen and T.W. Sederberg. Conversion of complex contour line definitions into polygonal element mosaics. *Computer Graphics*, 12(2):187–192, August 1978.

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

[3] 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.

[4] 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.

[5] S. Ganapathy and T.G. Dennehy. A new general triangulation method for planar contours. *Computer Graphics*, 16(3):69–75, July 1982.

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

[7] 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.

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

[9] 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.