Volume xxx, (1995) number yyy pp. 000–000

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 [1]. 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.

Keywords: Surface reconstruction; contours; tiling; meshes; multiresolution analysis; wavelets

1. Introduction

Reconstruction of surfaces from a set of planar contours such as those shown in Figure 2 is an important problem in medical and biological research, geology, and other areas. The problem can be broken into several subproblems [2], 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 1, 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 [3] 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 2, 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.

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 [4], and later improved by Fuchs, Kedem and Uselton [1]. We will refer to their algorithm as the *optimizing algorithm*. 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 the optimizing al-

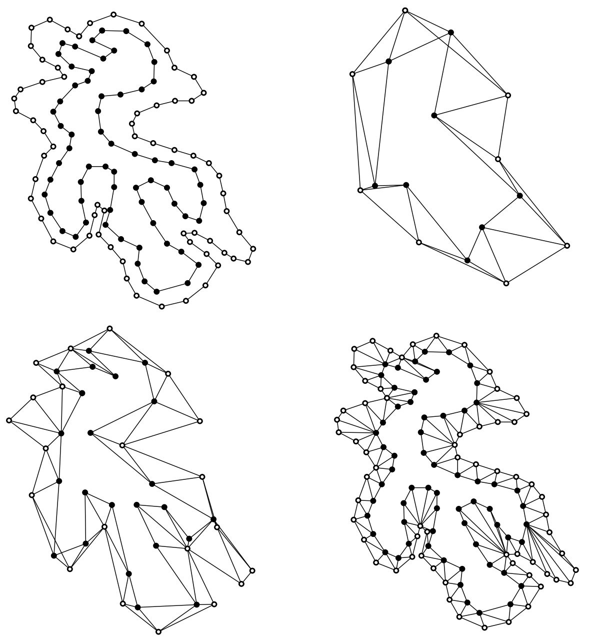

Figure 1: *The main steps of the multiresolution tiling algorithm.* **Upper Left:** *Input contours.* **Upper Right:** *Tiled base-case.* **Lower Left:** *Intermediate stage of single-wavelet reconstruction.* **Lower Right:** *Final tiling.*

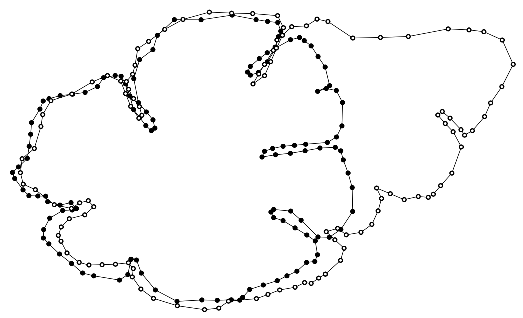

Figure 2: *A pair of contours obtained from the cerebral cortex of the human brain. The contours contain 128 (closed dots) and 114 (open dots) vertices respectively.*

gorithm, numerous other methods have been devised that do not perform a global optimization. A discussion of some of the methods can be found in Meyers, Skinner and Sloan [2].

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 optimizing algorithm. That 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 contours 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.

The contours of Figure 2 represent a severe test of a tiling algorithm. Each of the contours has several indentations along its perimeter that represent the infoldings or *sulci* of the cerebral cortex. An anatomist would require these indentations to be matched in a reconstruction so that the sulci maintain continuity.

As Figure 3 shows, even the optimizing algorithm can construct unacceptable tilings. Notice that some of the sulci are not matched in the tilings produced by the optimizing algorithm and our multiresolution algorithm. 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 algorithm 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 an introduction to multiresolution analysis and wavelets. The reader is referred to Chui [5] and Mallat [6] for a more detailed treatment. We first illustrate the basic idea behind wavelet analysis using as an example a one-dimensional piecewise linear function of a parameter t. We then show how

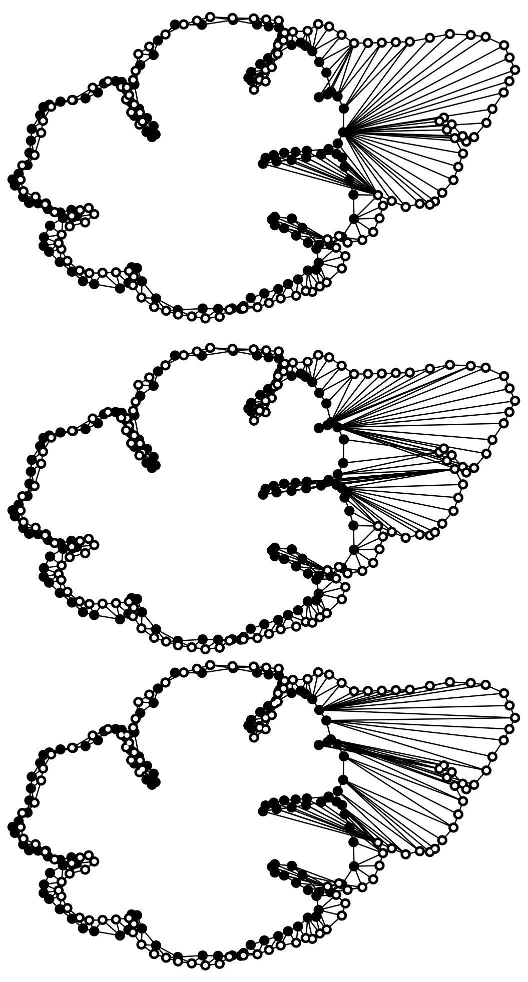

Figure 3: *Tilings computed by* **Upper:** *Our single-wavelet algorithm and* **Center:** *The method of Fuchs, Kedem and Uselton, for the contours in Figure 2. Neither result would be considered acceptable by a trained anatomist.* **Lower:** *an interactively modified tiling in which the sulci are matched from one section to the next.*

wavelet analysis can be used to find least-squares best low-resolution approximations of contours.

A sequence of 2^n real-valued samples can be represented by a piecewise-linear function $f^n(t)$ where the samples are considered to be taken at equally spaced integer *knot values* of the parameter t. In the following discussion, the notation c^n will be used to denote a sequence of 2^n samples, $(c_0^n, \ldots, c_{2^n-1}^n)$ representing the values of a piecewise linear function at the integer values of t.

If we are given a sequence of samples c^n, we can obtain an approximating sequence c^{n-1} with half as many samples by replacing c^n with a sequence found by convolving with a filter kernel $a = \{\ldots, a_{-1}, a_0, a_1, \ldots\}$ and then selecting every other element of the new sequence, a process known as *downsampling.* The sequence c^n is treated as a periodic function during convolution. Appropriate choice of the kernel a is an important consideration in such a process since its properties will determine the quality of the the low-resolution sequence. An important property is the ability to recover the detail lost when converting to the approximating sequence. If we can find a kernel $b = \{\ldots, b_{-1}, b_0, b_1, \ldots\}$ with the appropriate characteristics, we can obtain a detail sequence d^{n-1} by convolution of c^n with b followed by downsampling. If a and b are well chosen, it is possible to reconstruct the original sequence from c^{n-1} and d^{n-1} using two additional filters p^{n-1} and q^{n-1}. Multiresolution analysis makes use of such kernels to find a series of detail sequences and a low-resolution approximation of the original sequence.

Let us now take a more formal view of the process. Given c^n, we find c^{n-1} by convolution and downsampling with kernel a according to:

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

Similarly, d^{n-1} is obtained by convolution and downsampling with kernel b:

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

The elements of the sequence c^{n-1} replace the elements of c^n that have even values of t, and the elements of d^{n-1} can be thought of as capturing the lost detail.

The kernels a and b are often called *analysis filters.* If the analysis filters have been chosen appropriately, we can find a pair of kernels p and q (called *synthesis filters*) that will allow us to reconstruct c^n from c^{n-1} and d^{n-1}:

$$c_i^n = \sum_l [p_{i-2l} c_l^{n-1} + q_{i-2l} d_l^{n-1}]. \tag{3}$$

The analysis filters allow us to transform a sequence into two new sequences: a low-resolution approximation to the original, and a detail sequence; a process often called *decomposition.* The synthesis filters allow us to reconstruct the original sequence from the low-resolution sequence and the detail sequence, a process known as *reconstruction.* Decomposition can be applied to the low-resolution sequence c^{n-1}, to obtain c^{n-2} and d^{n-2}, and can be continued recursively until we obtain c^0 and a series of detail sequences $d^0, d^1, \ldots, d^{n-2}, d^{n-1}$, as illustrated by Figure 4. This

$$\begin{tikzcd}
c^{j} \arrow[r, "a"] \arrow[dr, "b"'] & c^{j-1} \arrow[r, "a"] \arrow[dr, "b"'] & c^{j-2} & \cdots \arrow[r, "a"] \arrow[dr, "b"'] & c^{0} \\
& d^{j-1} & d^{j-2} & \cdots & d^{0}
\end{tikzcd}$$

Figure 4: *Filter Bank*

set of sequences has the same number of elements as the original sequence c^n, and is often called the *wavelet transform* of that sequence. The elements of the detail sequences are often called *wavelet coefficients*. At any stage of the decomposition process, the sequence c^{n-j} gives the value of $c(t)$, for the *knots* at $t = k * 2^j$, $k \in \{0, \ldots, 2^{n-j} - 1\}$. The elements of the sequence d^{n-j} capture the detail lost when knots at $t = ((k * 2^j) + 2^{j-1})$ were removed during decomposition.

Because of the reversibility of the decomposition and reconstruction processes, the original sequence can be recovered from the wavelet transform. Therefore, no information is lost in going from one representation to the other. The process of recursively applying the analysis filters to a sequence is often called a *filter bank*, and is illustrated by Figure 4. The figure suggests an efficient algorithm for finding a wavelet transformation, called the *filter bank algorithm*. If the filters a, b, c, and d are all of finite width (or *support*), then the filter bank algorithm can be used to convert between the original sequence and wavelet transform in linear time.

Thus far, we have postulated the existence of suitable synthesis and analysis filters, but have said nothing about how they might be chosen. That is the subject of the next section.

2.1. Scaling Functions and Wavelets

In the previous section, we described how a set of analysis and synthesis filters can be used to represent a function at several levels of detail. We now examine the relationship between these discrete filters using the multiresolution analysis framework developed by Mallat [6].

The key idea is that a function $f^j(t)$ can be associated with a sequence c^j by thinking of the elements of c^j as the coefficients of a series of *scaling functions* $\phi(t)$, that are identical except for translation by an integer value:

$$f^j(t) = \sum_k c^j_k \phi(2^j t - k)$$

A scaling function must have the property that it is *refinable*. To be refinable, there must exist an unique sequence of coefficients $p = \{\ldots, p_{-1}, p_0, p_1, \ldots\}$ that allow $\phi(t)$ to be represented in terms of $\phi(2t)$:

$$\phi(t) = \sum_k p_k \phi(2t - k). \tag{4}$$

Therefore, a refinable function can be represented by the integer *translates* of a *scaled* version of itself. The refinement coefficients for the scaling function $\phi(t)$ are identically the synthesis filter p. If we consider the nature of the functions that can be represented by the translates of a scaling function, we find that there are sequences that can be represented exactly by the translates of $\phi(2t)$ that cannot be exactly represented by translates of $\phi(t)$. This leads us to the concept of a set of nested spaces spanned by integer translates of the scaling function at its different resolution levels. In more formal terms, we can define a set of linear spaces V^j given a scaling function $\phi(t)$:

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

Each of the spaces V^j is spanned by the integer translates of the appropriately scaled version of the scaling function: $\phi(2^j t)$. By that definition, the translates of $\phi(2^j t)$ form a basis for V^j. Since by definition a scaling function is refinable, we know that the spaces are nested:

$$V^0 \subset V^1 \subset V^2 \cdots .$$

Recall that the *orthogonal complement* of space B in space A is defined by

$$B^{\perp} = \{q \in A \mid \langle q, f \rangle = 0, \forall f \in B\}.$$

If we consider a pair of spaces V^j and V^{j+1} from this infinite set, we can define a space W^j to be the orthogonal complement of V^j in V^{j+1} so that:

$$V^{j+1} = V^j \oplus W^j.$$

Mallat defines a *wavelet* to be a function $\psi(x)$ such that the integer translates of $\psi(2^j x)$ form a basis for W^j. Together V^j and W^j span the space V^{j+1}. This property matches nicely with the idea of finding an approximation of a function and capturing the lost detail presented in the previous section. For a function that is an element of space V^{j+1}, we can find a lower-resolution approximation in V^j and capture the lost detail in W^j. This notion is stated more formally for scaling function $\phi(t)$ and wavelet $\psi(t)$ by the following *decomposition relation*:

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

The analysis filters a and b described in the previous section are identical to the coefficients that make the decomposition relation hold for the chosen wavelet and scaling function.

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

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

2.2. Multiresolution Analysis of Contours

The filter bank decomposition illustrated in Figure 4 can be applied to decomposition of a contour represented as a planar polygon by treating the x and y coordinates of its vertices separately, computing a decomposition for each sequence. The contour is treated as a pair of functions $x(t)$ and $y(t)$, periodic with respect to t over an interval equal to the number of vertices in the contour. The parameter t is allowed to range over the interval $[0, n)$ for a contour of n vertices. Each vertex is viewed as a knot, and has an integer value of t. Points on the contour between knots are linearly interpolated between pairs of knots, so that $x(t)$ and $y(t)$ are piecewise linear functions of t over a series of n knots with uniform knot spacing.

When using multiresolution analysis, it is important to choose the scaling function and wavelet appropriately. Since contours represented by planar polygons are piecewise linear functions, a scaling function and wavelet should be chosen that can easily represent piecewise linear functions. If it were desirable to represent contours as piecewise smooth functions, it would be desirable to choose a wavelet basis that could easily represent such a representation.

2.2.1. Choice of Scaling Function and Wavelet

For the multiresolution analysis of contours, we use the linear B-spline (or *hat function*) as the scaling function $\phi(t)$. The linear B-spline is a piecewise linear function defined by

$$\phi(t) = \begin{cases} t+1 & t \in [-1, 0] \\ -t+1 & t \in (0, 1] \\ 0 & \text{otherwise.} \end{cases}$$

With this choice of $\phi(t)$, we can determine the values of the synthesis filter p, which must make the relation of equation 4 hold. For the linear B-spline,

$$\phi(t) = \frac{1}{2}\phi(2t-1) + \phi(2t) + \frac{1}{2}\phi(2t+1).$$

Therefore the analysis filter p has non-zero values only for $k \in \{-1, 0, 1\}$.

For the wavelet function $\psi(t)$ we use a *single-knot wavelet* similar to that described by DeRose, Lounsbery and Warren [7]. As implied by its name, the single-knot wavelet adds only one new knot when added to an existing function using the filter bank. That property is useful when multiresolution analysis is used for data compression.

To obtain $\psi(t)$, we first define $\overline{\phi}(t)$ to be the orthogonal projection of $\phi(2t-1) \in V^1$ into V^0. Since $\overline{\phi}$ is an element of V^0 there must exists coefficients α_j such that

$$\overline{\phi}(t) = \sum_j \alpha_j \phi(t - j). \tag{5}$$

The wavelet $\psi(t)$ is defined in terms of $\phi(2t-1)$ and $\overline{\phi}(t)$ as

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

It is not difficult to show that $\psi(t)$ is a wavelet since its integer translates span W^0.

The above construction of $\psi(t)$ has an unfortunate consequence: when so defined, $\psi(t)$ has infinite support. Analysis and reconstruction are linear-time operations if the filters a, b, p, and q have finite support, but that is only possible if $\psi(t)$ has finite support. For our purposes, it is sufficient to modify the definition of $\psi(t)$ slightly so that its support is finite. The modified version of $\psi(t)$ is no longer a true wavelet; however, by appropriately choosing the number of non-zero terms in the projection of $\phi(2t-1)$ into V^0, orthogonality can be approached as closely as desired. In Appendix A we show how to find values for the sequence α in Equation 5 for finite support using a constrained least squares method.

Since $\psi(t)$ by our modified definition is no longer orthogonal to V^0, the low-resolution contours found are no longer the least squares best approximations to the original. They are, however, the best approximation subject to the constraints imposed, and they have worked well in practice. It is possible to define a wavelet that is orthogonal to V^0 and also of finite support. Chui [5] defines a minimally supported wavelet for the linear B-spline, but the synthesis filters a and b in his construction are of infinite extent, and must be truncated for implementation. Consequently, the minimally supported wavelet does not allow exact reconstruction. Chui shows an upper limit for the magnitude of the error as a function of the number of non-zero terms in the truncated analysis filters. We sacrifice orthogonality but maintain exact reconstruction and can approach orthogonality as closely as desired. In contrast, Chui sacrifices exact reconstruction, but shows that it can be approached as closely as desired.

2.2.2. Determining Filter Sequence Values

The values of the synthesis filter q and the analysis filter a depend on the values of the sequence α. Once the sequence α has been determined, the values of the

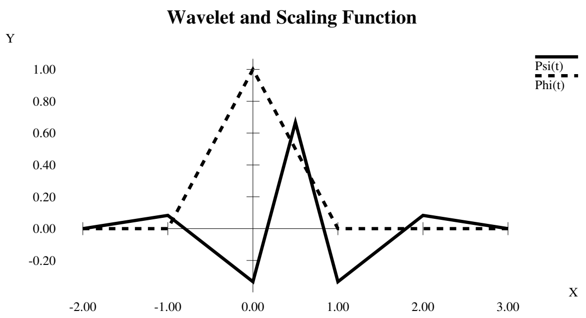

Figure 5: *The single-knot wavelet $\psi(t)$ and linear B-spline scaling function $\phi(t)$.*

synthesis filter q are

$$q_k = \begin{cases} -\alpha_{\frac{k}{2}} & k \text{ even} \\ -\frac{1}{2}(\alpha_{\frac{k-1}{2}} + \alpha_{\frac{k+1}{2}}) & k \text{ odd}, k \neq 1 \\ 1 - \frac{1}{2}\alpha_0 - \frac{1}{2}\alpha_1 & k = 1. \end{cases} \tag{6}$$

and the values of the analysis filter a are

$$a_{-2k} = \begin{cases} -\frac{1}{2}(\alpha_k + \alpha_{k+1}) & k \neq 0 \\ 1 - \frac{1}{2}(\alpha_0 + \alpha_1) & k = 0 \end{cases} \tag{7}$$

$$a_{1-2k} = \alpha_k. \tag{8}$$

Using these relationships, the filters a and q can be found to any number of non-zero terms by solving for the required number of non-zero terms of the sequence α. The derivations of Equations 6, 7, and 8 are found in Appendix A.

To apply wavelet analysis to contours, we treat a polygonal contour as a periodic sequence of *knots* with equally spaced integer values of a parameter t, each with associated values $x(t)$ and $y(t)$. To find low-resolution contours, the wavelet transform is applied to $x(t)$ and $y(t)$ independently. The original number of vertices in a contour determines the space V^j into which the highest resolution representation of the contour falls.

The functions $\phi(t)$ and $\psi(t)$ are plotted in Figure 5; Table 1 shows the non-zero terms of analysis filters a and b and synthesis filters p and q when α contains 4 non-zero terms.

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 (see Figure 1). These low resolution contours are tiled using the optimizing algorithm.

i	a_i	b_i	p_i	q_i
-4	1/28	0	0	0
-3	-1/14	0	0	1/28
-2	-1/8	0	0	1/14
-1	9/28	0	1/2	-1/8
0	19/28	-1/2	1	-9/28
1	9/28	1	1/2	19/28
2	-1/8	-1/2	0	-9/28
3	-1/14	0	0	-1/8
4	1/28	0	0	1/14
5	0	0	0	1/28

Table 1: *The non-zero terms of the analysis filters a and b and the synthesis filters p and q found when α contains 4 non-zero terms.*

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 contours' shapes are very different. In such situations, it is often the case that neither method produces an acceptable result (see Figure 3).

If the multiresolution algorithm is to achieve an overall speedup, its reconstruction and local optimization processes must be fast. If addition of one wavelet coefficient to the reconstruction requires as much as $O(n)$ time, then the overall reconstruction process will require $O(n^2)$ time. To achieve the desired speedup, it is necessary that the time to add a single wavelet during reconstruction be a nearly constant-time operation. Addition of a wavelet coefficient to a contour can be done in constant time using the filter bank algorithm. To show an improvement in time complexity over the optimizing algorithm, it is necessary to demonstrate that the additional time required for the addition of edges and local optimization of the tiling is sufficiently small. A proof of upper bound for this process is difficult because 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. It is conceivable that addition of each wavelet could cause enough perturbation for a cascade of swaps to occur. If such situations are common, $O(n)$ time could be required for local optimization after addition of each wavelet, resulting in an overall $O(n^2)$ time complexity. Although we offer no proof of an upper bound, in Section 4 we present experimental data to support the assertion that for the average case, local optimization after addition of a wavelet is very nearly a constant time operation (see Figure 8 and 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 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

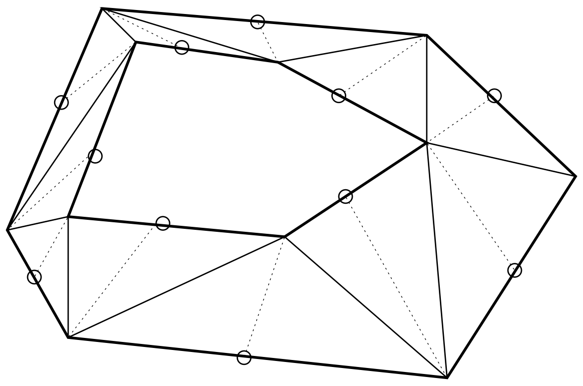

Figure 6: *Illustration of vertex and edge addition during filter bank reconstruction reconstruction. Newly added vertices are open circles, newly added edges are shown as dotted lines and connect each newly added vertex to a vertex of the opposite contour. In a real situation the number of vertices of each contour would be a power of 2, the original vertices would be moved to new locations, and the newly added vertices would not strictly bisect original edges.*

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, construct a tiling for 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 splits 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. The newly added vertices do not in general bisect an original edge, since their locations are determined by the magnitude of the wavelet coefficients involved. Rather, each edge is split into two, and all original vertices of both contours may be moved from their original positions.
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 (see Figure 10) requires approximately constant time; the overall cost of the filter bank tiling method therefore appears to be $O(n \log n)$ (see 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, regardless of the resolution level from which they were obtained, and to avoid using a wavelet if the magnitude of its coefficient 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 two cases of the addition of a single 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:

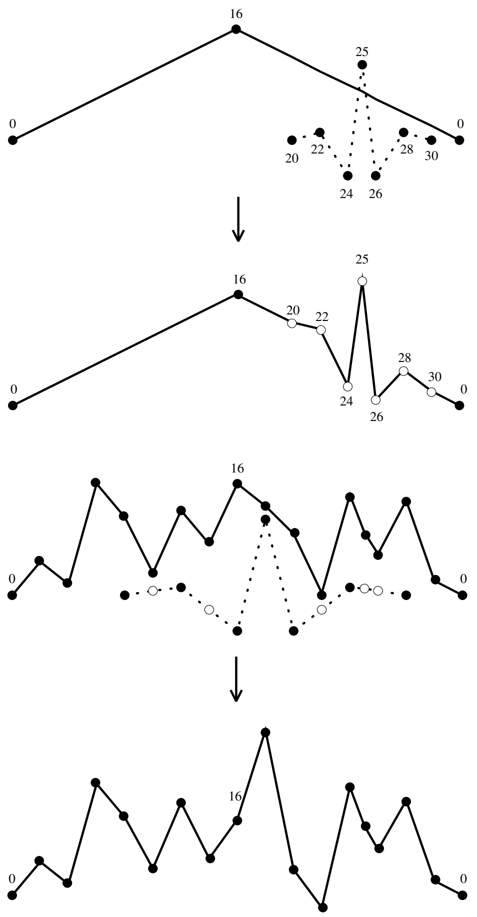

Figure 7: *Illustration of single-wavelet reconstruction in one dimension.* **Upper:** *Addition of a fine detail wavelet to a coarse detail function. All intrinsic wavelet knots are added to the function, shown by white circles.* **Lower:** *Addition of a coarse detail wavelet to a function already at a fine level of detail. All intrinsic wavelet knots are already present in the function. Knots indicated by white circles must be added to wavelet before knot vectors are merged.*

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 Figure 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 optimizing algorithm 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 the optimizing algorithm. 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

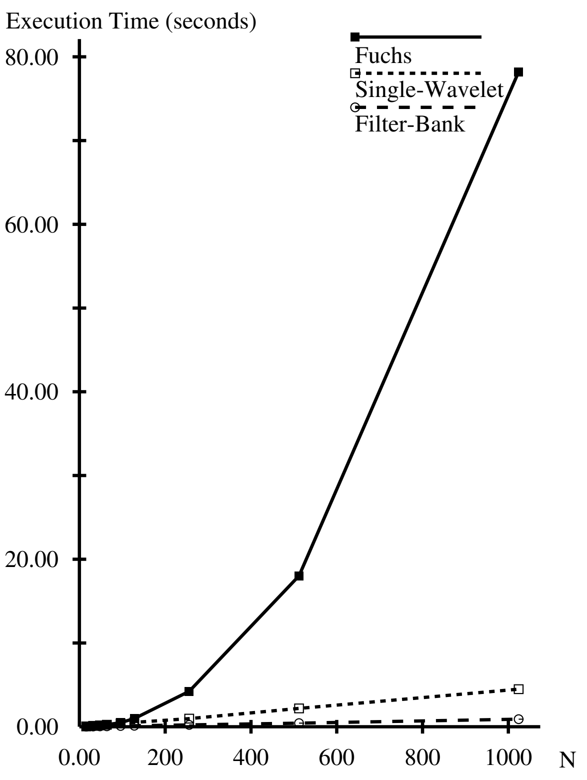

Figure 8: *Execution time versus N for the Fuchs, Kedem, Uselton algorithm and the filter bank and single-wavelet multiresolution algorithms.*

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 optimizing algorithm and the sorted single-wavelet algorithm are approximately equal (see Figure 8). Contours containing 64 or fewer vertices can be tiled using the optimizing 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 optimizing 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 optimizing algorithm. The results of these tests are shown in Figures 8, 9, and 10.

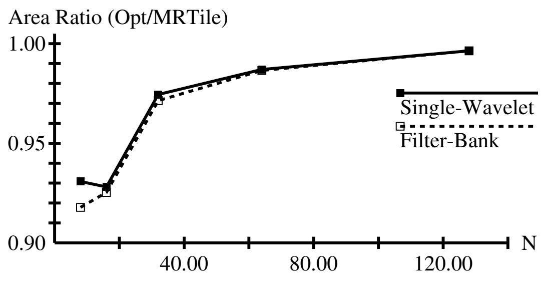

Figure 9: *Tiling quality as a function of base-case size for the contours of Figure 2.*

Figure 8 shows the timing results obtained for each of the optimizing, 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 optimizing algorithm. The optimizing 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 2. Notice that larger base-case size 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 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 2 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

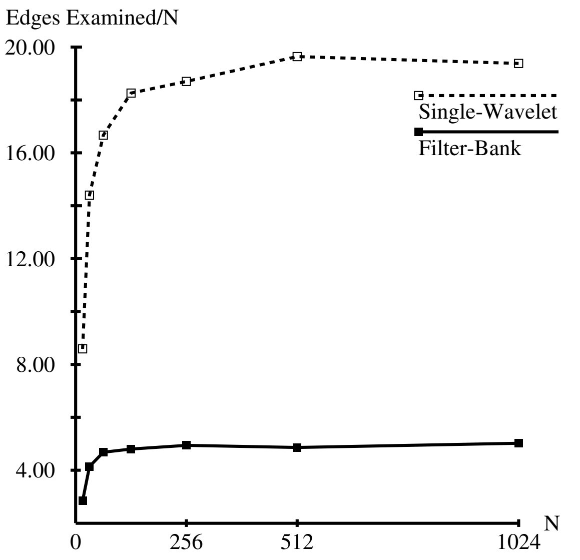

Figure 10: *Number of edges examined per vertex during the optimization process for contours ranging in size from 16 to 1024 vertices by the filter bank and single-wavelet reconstruction methods. A base-case size of 8 was used.*

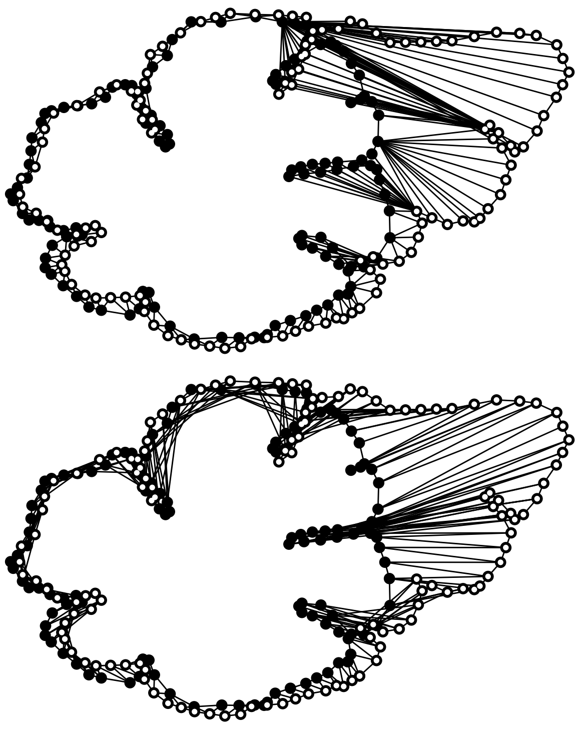

Figure 11: *Tilings computed by* **Upper:** *The algorithm of Christiansen and Sederberg and* **Lower:** *The algorithm of Ganapathy and Dennehy, for the contours in Figure 2. Compare to the tilings shown in Figure 3.*

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, connects 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 2 using the linear-time "greedy" methods of Ganapathy and Dennehy [8] and of Christiansen and Sederberg [9]. Both methods construct a tiling beginning with an edge assumed to be a good starting point. They then 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 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 top may be adequate. The low-resolution version requires

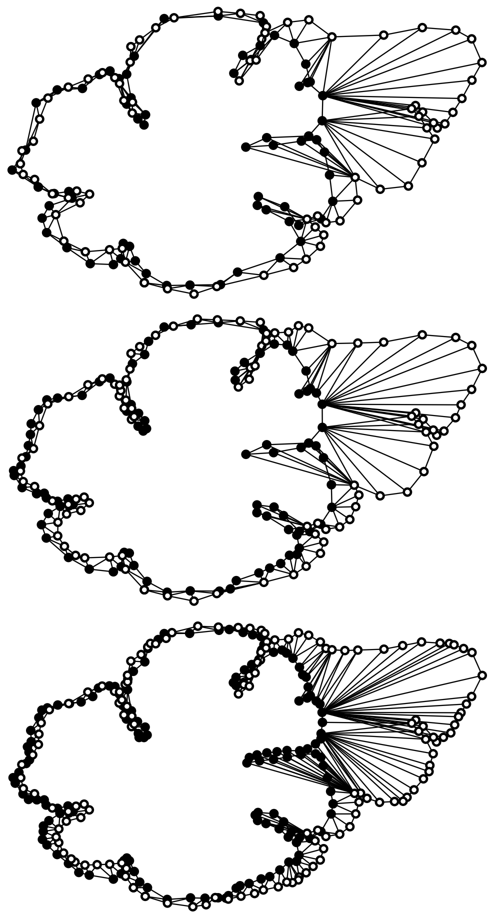

Figure 12: *Tilings of the contours in Figure 2 using the single-wavelet algorithm with threshold factors of* **Top:** *0.005,* **Center:** *0.0025, and* **Bottom:** *0.001. The threshold factor multiplied by the magnitude of the largest wavelet coefficient determines the magnitude of the smallest coefficient used.*

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 optimizing algorithm for solving the tiling problem, that of Fuchs, Kedem and Uselton [1].

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.

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. Fuchs, Z. Kedem, and S. Uselton, "Optimal surface reconstruction from planar contours", *Communications of the ACM*, **20**(10), pp. 693–702 (1977).
2. D. Meyers, S. Skinner, and K. Sloan, "Surfaces from contours", *ACM Transactions on Graphics*, **11**(3), pp. 228–258 (1992).
3. J. O'Rourke and V. Subramanian, "On reconstructing polyhedra from parallel slices", Tech. Rep. TR # 008, Smith College Department of Computer Science, Northampton, MA 01063, (June 20, 1991).
4. E. Keppel, "Approximating complex surfaces by triangulation of contour lines", *IBM J. Res. Develop.*, **19**, pp. 2–11 (1975).
5. C. K. Chui, *An Introduction To Wavelets.* Academic Press, Inc., (1992).
6. S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation.", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**(7), pp. 674–693 (1989).
7. T. D. DeRose, M. Lounsbery, and J. Warren, "Multiresolution analysis for surfaces of arbitrary topological type", Tech. Rep. 93-10-05, University of Washington, Dept. of Computer Science and Engineering, (1993).

8. S. Ganapathy and T. Dennehy, "A new general triangulation method for planar contours", *Computer Graphics*, **16**(3), pp. 69–75 (1982).

9. H. Christiansen and T. Sederberg, "Conversion of complex contour line definitions into polygonal element mosaics", *Computer Graphics*, **12**(2), pp. 187–192 (1978).

10. G. H. Golub and C. F. Van Loan, *Matrix Computations*, ch. 12, pp. 566–567. The Johns Hopkins University Press, second ed., (1989).

Appendix A: Wavelets, Scaling Functions, and Filters

This appendix presents the derivation of the single-knot wavelet and shows how the analysis and synthesis filters are obtained for the linear B-spline scaling function and single-knot wavelet.

Linear B-Spline Scaling Function and Single-Knot Wavelet

The linear B-spline scaling function $\phi(t)$ is the piecewise linear "hat" function

$$\phi(t) = \begin{cases} 0 & t < -1, t > 1 \\ t+1 & -1 \le t \le 0 \\ 1-t & 0 < t \le 1. \end{cases} \tag{9}$$

We here derive a *single-knot wavelet* $\psi(t)$ for use with the linear B-spline scaling function by a construction similar to that used by DeRose, Lounsbery and Warren [7].

To obtain $\psi(t)$, we first define $\overline{\phi}(t)$ to be the orthogonal projection of $\phi(2t-1) \in V^1$ into V^0. The orthogonal projection gives a vector of coefficients α that allow us to express $\overline{\phi}(t)$ as

$$\overline{\phi}(t) = \sum_j \alpha_j \phi(t-j). \tag{10}$$

The wavelet $\psi(t)$ is defined in terms of $\phi(2t-1)$ and $\overline{\phi}(t)$ as

$$\psi(t) = \phi(2t-1) - \overline{\phi}(t). \tag{11}$$

Since $\overline{\phi}(t)$ is the orthogonal projection of $\phi(2t-1)$ into V^0, $\psi(t)$ is orthogonal to V^0. It can be shown that the integer translates $\psi(t-k)$ of $\psi(t)$ span the space W^0.

Computing $\overline{\phi}(t)$

This section describes the method used to find α, the vector of coefficients of Equation 10. We later define the analysis filter a and the synthesis filter q in terms of these coefficients.

We will make use of the following definitions:
The inner product $\langle g(t), h(t) \rangle$ is defined to be

$$\langle g(t), h(t) \rangle \equiv \int_{-\infty}^{+\infty} g(t)h(t)dt.$$

For notational convenience, we define:

$$\phi_i(t) \equiv \phi(t-i)$$

We use the notation $[\langle \phi_i(t), \phi_j(t) \rangle]$ to represent the infinite square matrix with $\langle \phi_i(t), \phi_j(t) \rangle$ as the entry in row i and column j.

The vector of coefficients α is found by orthogonal projection of $\phi(2t-1)$ into V^0. This is equivalent to solving the folowing infinite linear system for α^T:

$$[\langle \phi_i(t), \phi_j(t) \rangle]\, \alpha^T = [\langle \phi(2t-1), \phi_i(t) \rangle]\,. \tag{12}$$

where i and j range over the integers. Defined in this way, $\overline{\phi}(t)$ is the least squares best approximation of $\phi(2t-1) \in V^1$ in the space V^0. There is a problem with our definition of $\overline{\phi}(t)$: the sequence of coefficients α has infinite support; the analysis filter a and synthesis filter q will consequently have infinite support. Implementation of the filter bank algorithm requires that the filters have a finite number of non-zero terms. It is possible to modify the definition of $\overline{\phi}(t)$ so that it has finite support by limiting the square matrix $[\langle \phi_i(t), \phi_j(t) \rangle]$ to finite size. Solving Equation 12 subject to that restriction finds the vector α that makes $\overline{\phi}(t)$ the least squares best approximation to $\phi(2t-1)$ for the chosen number of non-zero terms. Unfortunately, that solution suffers from the problem that the value of the inner product $\langle \psi(t), 1 \rangle$ may be non-zero, which is not the case for infinite support. Since a constant function can be exactly represented by the scaling function $\phi(t)$, we desire that $\langle \psi(t), f(t) \rangle = 0$ when $f(t) = c$ over the support of the wavelet. With that requirement, wavelets extracted from nearly constant regions of a function will tend to have small values. That property is desirable, because these small coefficients can be eliminated during reconstruction if lossy compression is desired. To find the best α subject to the additional constraint that $\langle \psi(t), 1 \rangle = 0$, we solve a constrained least squares problem of the form

$$\min_{Bx = d} \|Ax - b\|_2$$

where

$$\begin{aligned} A &= [\langle \phi_i(t), \phi_j(t) \rangle] \\ b &= [\langle \phi(2t-1), \phi_i(t) \rangle] \\ x &= \alpha^T \\ B &= [\langle \phi_j(t), 1 \rangle] \\ d &= [\langle \phi(2t-1), 1 \rangle]\,. \end{aligned}$$

The matrix A is restricted to finite size. We solve this constrained least squares problem by the method described in Golub and Van Loan [10].

The function $\psi(t)$ obtained after modifying our definition of $\overline{\phi}(t)$ is no longer strictly orthogonal to V^0, so that $\psi(t)$ is no longer strictly a wavelet. By appropriate selection of the size of the matrix $[\langle \phi_i(t), \phi_j(t) \rangle]$, we can approach orthogonality as closely as desired.

An additional consideration when selecting the number of non-zero terms of the sequence α is that the use of an even number of non-zero α gives symmetric sequences a and q, while an odd number results in asymmetric values. We recommend use of an even number of non-zero α for that reason.

Analysis and Synthesis Filters

The values of the analysis filter a and the synthesis filter q depend on the values of the sequence α found by solving Equation 12, while the values of b and p do not. In this section, we derive the relationship between the sequence α and the filters a and q and determine the values of filters b and p. We make use of the following relationships:

$$\begin{aligned} \phi(t) &= \sum_k p_k \phi(2t-k) && (13) \\ \psi(t) &= \sum_k q_k \phi(2t-k) && (14) \\ \phi(2t) &= \sum_k [a_{-2k}\phi(t-k) + b_{-2k}\psi(t-k)] && (15) \\ \phi(2t-1) &= \sum_k [a_{1-2k}\phi(t-k) + b_{1-2k}\psi(t-k)] && (16) \\ \phi(2t-l) &= \sum_k [a_{l-2k}\phi(t-k) + b_{l-2k}\psi(t-k)] && (17) \end{aligned}$$

which are Chui's equations (1.6.2), (1.6.3), (1.6.4), (1.6.5), and (1.6.6) [5]. Equations 13 and 14 are the *two-scale relations* of the scaling function and wavelet. Equation 17 represents a composition of Equations 15 and 16, and is called the *decomposition relation.* From these relationships and the definitions of $\phi(t)$, $\overline{\phi}(t)$, and $\psi(t)$ the values of the filters a, b, p, and q can be determined as discussed below.

Synthesis Filter p

The filter p must make the relation of Equation 13 hold. For our choice of the linear B-spline (Equation 9) as $\phi(t)$, the two-scale relation is

$$\phi(t) = 1/2\phi(2t+1) + \phi(2t) + 1/2\phi(2t-1). \qquad (18)$$

From Equations 13 and 18 we conclude that the filter p is

$$p_k = \begin{cases} 0 & k \notin \{-1, 0, 1\} \\ 1/2 & k \in \{-1, 1\} \\ 1 & k = 0. \end{cases}$$

Synthesis Filter q

To determine the values of the filter q, Equation 11 can be rewritten by expanding $\overline{\phi}(t)$ using Equations 10 and 18 as

$$\psi(t) = \phi(2t-1) - \left[\sum_{j_{even}} \alpha_{\frac{j}{2}} \phi(2t-j) + \sum_{j_{odd}} \frac{1}{2}(\alpha_{\frac{j-1}{2}} + \alpha_{\frac{j+1}{2}})\phi(2t-j) \right]. \tag{19}$$

Comparing Equation 19 to Equation 14, we can see that the sequence q is

$$q_k = \begin{cases} -\alpha_{\frac{k}{2}} & k \text{ even} \\ -\frac{1}{2}(\alpha_{\frac{k-1}{2}} + \alpha_{\frac{k+1}{2}}) & k \text{ odd}, k \neq 1 \\ 1 - \frac{1}{2}\alpha_0 - \frac{1}{2}\alpha_1 & k = 1. \end{cases} \tag{20}$$

Analysis Filters a and b

We now find values for filters a and b. Filters a and b actually consist of two sequences each: the "even" sequences a_{-2k}, and b_{-2k} and the "odd" sequences a_{1-2k}, and b_{1-2k}. We first derive the values of the sequences a_{1-2k} and b_{1-2k}. From the definition of $\psi(t)$ in Equation 11 and substituting from Equation 10 we see that

$$\phi(2t-1) = \sum_j \alpha_j \phi(t-j) + \psi(t). \tag{21}$$

Comparing with Equation 16 we see that

$$a_{1-2k} = \alpha_k \tag{22}$$

and

$$b_{1-2k} = \begin{cases} 1 & k = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{23}$$

To derive the values of a_{-2k} and b_{-2k} we note that Equation 18 can be rearranged to

$$\phi(2t) = \phi(t) - 1/2\,[\phi(2t+1) + \phi(2t-1)],$$

which can be rewritten after some manipulation as:

$$\phi(2t) = \phi(t) - 1/2 \left[\sum_j [(\alpha_j + \alpha_{j+1})\phi(t-j)] + \psi(t+1) + \psi(t) \right]. \tag{24}$$

Comparing with Equation 15, we see that

$$a_{-2k} = \begin{cases} -\frac{1}{2}(\alpha_k + \alpha_{k+1}) & k \neq 0 \\ 1 - \frac{1}{2}(\alpha_0 + \alpha_1) & k = 0 \end{cases} \tag{25}$$

and

$$b_{-2k} = \begin{cases} -\frac{1}{2} & k \in \{-1, 0\} \\ 0 & \text{otherwise.} \end{cases} \tag{26}$$

Combining the even and odd sequences as in Equation 17, we have

$$a_{l-2k} = \begin{cases} \alpha_k & l \text{ odd} \\ -\frac{1}{2}(\alpha_k + \alpha_{k+1}) & l \text{ even},\ k \neq 0 \\ 1 - \frac{1}{2}(\alpha_0 + \alpha_1) & l \text{ even},\ k = 0 \end{cases} \tag{27}$$

and

$$b_{l-2k} = \begin{cases} 1 & l\ odd,\ k = 0 \\ 0 & l\ odd,\ k \neq 0 \\ -\frac{1}{2} & l \text{ even},\ k \in \{-1, 0\} \\ 0 & \text{otherwise.} \end{cases} \tag{28}$$

Summary

Implementation of filterbank analysis and reconstruction requires that the filters used are of finite support (finite impulse response (FIR) filters), except in special cases [5]. The single-knot wavelet as defined by Equation 11 has infinite support. It can be approximated to any desired precision as a FIR filter by appropriate specification of the dimension of the matrix $[\langle\phi_i(t), \phi_j(t)\rangle]$ of Equation 12. After specifying the dimension, the sequence α is obtained by solving a constrained least squares problem, and the values for the filters are obtained by substitution of the appropriate values into equations 25 and 20. This approach allows exact reconstruction, but the wavelet is not orthogonal to the scaling function.

Chui [5] defines a minimally supported wavelet for the linear B-spline, but filters a and b in his construction are of infinite extent, and must be truncated for implementation. Because the a and b sequences must be truncated during implementation, the minimally supported wavelet does not allow exact reconstruction. Chui shows an upper limit for the magnitude of the error as a function of the number of non-zero terms in the truncated analysis filters. We sacrifice orthogonality but maintain exact reconstruction and can approach orthogonality as closely as desired. In contrast, Chui's approach maintains orthogonality but sacrifices exact reconstruction, and he shows that exact reconstruction can be approached as closely as desired. One advantage of our approach is that exact reconstruction is possible with a relatively narrow filter, albeit at a sacrifice of orthogonality. For our purpose, orthogonality is not as important as exact reconstruction.