Multiresolution Curves

Adam Finkelstein and David H. Salesin
Department of Computer Science and Engineering
University of Washington
Seattle, Washington 98195

April 1994

Technical Report 94-01-06b

Multiresolution Curves

Adam Finkelstein *David H. Salesin*

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

Abstract

We describe a multiresolution curve representation, based on wavelets, that conveniently supports a variety of operations: smoothing a curve; editing the overall form of a curve while preserving its details; and approximating a curve within any given error tolerance for scan conversion. We present methods to support continuous levels of smoothing as well as direct manipulation of an arbitrary portion of the curve; the control points, as well as the discrete nature of the underlying hierarchical representation, can be hidden from the user. The multiresolution representation requires no extra storage beyond that of the original control points, and the algorithms using the representation are both simple and fast.

CR Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling — Curve, Surface, Solid, and Object Representations; I.3.6 [Computer Graphics]: Methodology and Techniques — Interaction Techniques.

Additional Key Words: curve compression, curve editing, curve fitting, curve smoothing, direct manipulation, scan conversion, wavelets.

1 Introduction

A good representation for curves should allow for flexible editing, smoothing, and scan conversion. In particular, a representation for curves should support:

- the ability to change the overall "sweep" of a curve while maintaining its fine details, or "character" (Figure 3);
- the ability to change a curve's "character" without affecting its overall "sweep" (Figure 6);
- the ability to edit a curve at any continuous level of detail, allowing an arbitrary portion of the curve to be affected through direct manipulation (Figure 4);
- continuous levels of smoothing, in which undesirable features are removed from a curve (Figure 2);
- curve approximation, or "fitting," within a guaranteed maximum error tolerance, for scan conversion and other applications (Figures 8 and 9).

In this paper, we show how a *multiresolution* curve representation can provide a single, unified framework for addressing all of these issues. It requires no extra storage beyond that of the original m control points, and the algorithms that use it are both simple and fast, typically linear in m.

There are many applications of multiresolution curves, including computer-aided design, in which cross-sectional curves are frequently used in the specification of surfaces; keyframe animation, in which curves are used to control parameter interpolation; 3D modeling and animation, in which "backbone" curves are manipulated to specify object deformations; graphic design, in which curves are used to describe regions of constant color or texture; font design, in which curves represent the outlines of characters; and pen-and-ink illustration, in which curves are the basic elements of the finished piece. In all of these situations, the editing, smoothing, and approximation techniques we describe can be powerful tools.

1.1 Related work

Some of the algorithms supported by multiresolution curves are completely new, to our knowledge, such as the ability to edit a curve at any continuous level of detail, and the ability to change a curve's character without affecting its overall sweep. However, the majority of applications described in this paper have already been addressed in one form or another. Although the algorithms we describe compare favorably, in and of themselves, with most of this previous work, it is the convenience with which the multiresolution representation supports such a wide variety of operations that makes it so useful. Here we survey some of these previous techniques.

Forsey and Bartels [13] employ hierarchical B-splines to address the problem of editing the overall form of a surface while maintaining its details. Their original formulation requires the user to design an explicit hierarchy into the model. In later work [14], they describe a method for recursively fitting a hierarchical surface to a set of data by first fitting a coarse approximation and then refining in areas where the residual is large. This construction is similar in spirit to the filter bank process used in multiresolution analysis, as described in Section 2.1. One significant difference is that in their formulation there are an infinite number of possible representations for the same surface, whereas the multiresolution curve representation is unique for a given shape. Fowler [15] and Witkin and Welch [28] also describe methods in which editing can be performed over narrower or broader regions of a surface; however, in neither of these works is there an attempt to preserve the higher-resolution detail beneath the edited region.

Curve and surface smoothing algorithms that minimize various energy norms have also been studied; these are surveyed in Hoschek and Lasser [16]. One example is the work of Celniker and Gossard [7], in which a fairness functional is applied to hand-drawn curves, as well as to surfaces. The method we describe is really a least-squares type of smoothing, which is much simpler but supports continuous levels of smoothing that behaves quite reasonably and intuitively in practice.

Many schemes for approximating curves within specified error tolerances have also been explored [2, 20, 23, 27]. Most of this research has centered on various forms of knot removal for representing curves efficiently with non-uniform B-splines. In this paper, we look at the very practical concern of producing a small number of Bézier segments that approximate the curve well, since these segments are the standard representation for curves in PostScript [1], the most common page description language. Our requirements are also somewhat different than those of most previous curve-fitting methods. In particular, for our application of scan conversion we do not require any particular continuity constraints for the approximating curve. Relaxing this condition allows for potentially much higher compression rates.

1.2 Overview

The next section discusses the theory of multiresolution analysis, and develops a multiresolution representation for B-spline curves. Sections 3, 4, and 5 describe how this representation can be used to support efficient smoothing,

editing, and scan conversion. Finally, Section 6 suggests some areas for future research. The details of the multiresolution curve formulation can be found in the appendices.

2 Theory of multiresolution curves

In this section, we discuss the theory of wavelets and multiresolution analysis, and we show how it can be applied to representing endpoint-interpolating B-spline curves.

2.1 Wavelets and multiresolution analysis

Wavelets are a simple mathematical tool that have found a wide variety of applications in recent years, including signal analysis [22], image processing [11], and numerical analysis [6]. In this section, we sketch the basic ideas behind wavelets and multiresolution analysis. Rather than presenting the classical multiresolution analysis developed by Mallat [22], we present here a slightly generalized version of the theory, following Lounsbery *et al.* [19], that is more convenient for our application of representing open curves.[1]

Consider a discrete signal C^n, expressed as a column vector of samples $[c_1^n, \ldots, c_m^n]^{\mathrm{T}}$. In our application, the samples c_i^n could be thought of as a curve's control points in $\mathbb{R}^2$.

Suppose we wish to create a low-resolution version C^{n-1} of C^n with a fewer number of samples m'. The standard approach for creating the m' samples of C^{n-1} is to use some form of filtering and downsampling on the m samples of C^n. This process can be expressed as a matrix equation

$$C^{n-1} \quad = \quad A^n C^n \tag{1}$$

where A^n is an $m' \times m$ matrix.

Since C^{n-1} contains fewer samples than C^n, it is intuitively clear that some amount of detail is lost in this filtering process. If A^n is appropriately chosen, it is possible to capture the lost detail as another signal D^{n-1}, computed by

$$D^{n-1} \quad = \quad B^n C^n \tag{2}$$

where B^n is an $(m - m') \times m$ matrix, which is related to matrix A^n. The pair of matrices A^n and B^n are called *analysis filters*. The process of splitting a signal C^n into a low-resolution version C^{n-1} and detail D^{n-1} is called *decomposition*.

If A^n and B^n are chosen correctly, then the original signal C^n can be recovered from C^{n-1} and D^{n-1} by using another pair of matrices P^n and Q^n, called *synthesis filters*, as follows:

$$C^n \quad = \quad P^n C^{n-1} \; + \; Q^n D^{n-1} \tag{3}$$

Recovering C^n from C^{n-1} and D^{n-1} is called *reconstruction*.

Note that the procedure for splitting C^n into a low-resolution part C^{n-1} and a detail part D^{n-1} can be applied recursively to the new signal C^{n-1}. Thus, the original signal can be expressed as a hierarchy of lower-resolution signals $C^0, \ldots, C^{n-1}$ and details $D^0, \ldots, D^{n-1}$, as shown in Figure 1. This recursive process is known as a *filter bank*.

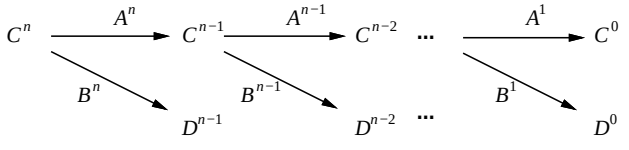


Figure 1: *The filter bank.*

Since the original signal C^n can be recovered from the sequence C^0, D^0, D^1, ..., D^{n-1}, this sequence can be thought of as a transform of the original signal, known as a *wavelet transform*. Note that the total size of the transform C^0, D^0, ..., D^{n-1} is the same as that of the original signal C^n, so no extra storage is required.

Wavelet transforms have a number of properties that make them attractive for signal processing. First, if the filters A^j, B^j, P^j, and Q^j are constructed to be sparse, then the filter bank operation can be performed very quickly — often in $O(m)$ time. Second, for many of the signals encountered in practice, a large percentage of the entries in the wavelet transform are negligible. Wavelet compression methods can therefore approximate the original set of samples in C^n by storing only the significant coefficients of the wavelet transform. Impressive compression ratios have been reported for univariate signals as well as for images [11].

As suggested by the treatment above, all that is needed for performing a wavelet transform is an appropriate set of analysis and synthesis filters A^j, B^j, P^j, and Q^j. To see how to construct these filters, we associate with each signal C^n a function $f^n(u)$ with $u \in [0, 1]$ given by

$$f^n(u) \quad = \quad \Phi^n(u)\, C^n \tag{4}$$

where $\Phi^n(u)$ is a row matrix of basis functions $[\phi_1^n(u), \ldots, \phi_m^n(u)]$, called *scaling functions*. In our application, for example, the scaling functions are the endpoint-interpolating B-splines basis functions, in which case the function $f^n(u)$ would be an endpoint-interpolating B-spline curve.[2]

The scaling functions are required to be *refinable*; that is, for all j in $[1, n]$ there must exist a matrix P^j such that

$$\Phi^{j-1} \quad = \quad \Phi^j P^j \tag{5}$$

In other words, each scaling function at level $j-1$ must be expressible as a linear combination of "finer" scaling functions at level j. As suggested by the notation, the refinement matrix in equation (5) turns out to be the same as the synthesis filter P^j.

Next, let V^j be the linear space spanned by the set of scaling functions Φ^j. The refinement condition on Φ^j implies that these linear spaces are nested: $V^0 \subset V^1 \subset \cdots \subset V^n$. Choosing an inner product for the basis functions in V^j allows us to define W^j as the *orthogonal complement* of V^j in V^{j+1}, that is, the space W^j whose basis functions $\Psi^j = [\psi_1^j(u), \ldots, \psi_{m-m'}^j(u)]$ are such that Φ^j and Ψ^j together form a basis for V^{j+1}, and every $\psi_i^j(u)$ is orthogonal to every $\phi_i^j(u)$ under the chosen inner product. The basis functions $\psi_i^j(u)$ are called *wavelets*.

We can now construct the synthesis filter Q^j as the matrix that satisfies

$$\Psi^{j-1} \quad = \quad \Phi^j Q^j \tag{6}$$

Equations (5) and (6) can be expressed as a single equation by concatenating the matrices together:

$$\left[\, \Phi^{j-1} \;\middle|\; \Psi^{j-1} \,\right] \quad = \quad \Phi^j \left[\, P^j \;\middle|\; Q^j \,\right] \tag{7}$$

Finally, the analysis filters A^j and B^j are formed by the matrices satisfying the inverse relation:

$$\left[\, \Phi^{j-1} \;\middle|\; \Psi^{j-1} \,\right] \left[\frac{A^j}{B^j}\right] \quad = \quad \Phi^j \tag{8}$$

Note that $\left[\, P^j \;\middle|\; Q^j \,\right]$ and $\left[\, A^j \;\middle|\; B^j \,\right]^{\mathrm{T}}$ are both square matrices. Thus,

$$\left[\frac{A^j}{B^j}\right] \quad = \quad \left[\, P^j \;\middle|\; Q^j \,\right]^{-1} \tag{9}$$

from which it is easy to prove a number of useful identities:

$$\begin{aligned} A^j Q^j \;&=\; B^j P^j \;&=\; \mathbf{0} \\ A^j P^j \;&=\; B^j Q^j \;&=\; P^j A^j \;+\; Q^j B^j \;=\; \mathbf{1} \end{aligned} \tag{10}$$

where $\mathbf{0}$ and $\mathbf{1}$ are the matrix of zeros and the identity matrix, respectively.

[1] The more general theory described here differs from Mallat's original formulation by relaxing his condition that the basis functions must be translates and scales of one another.

[2] For simplicity of notation, we often omit the explicit dependence on u when writing f^n and Φ^n.

2.2 Multiresolution endpoint-interpolating B-splines

In our application, we build a multiresolution analysis for B-spline curves. In this paper, we restrict our attention to the common case of cubic B-splines defined on a knot sequence that is uniformly spaced everywhere except at its ends, where its knots have multiplicity 4. Such B-splines are commonly referred to as *endpoint-interpolating* cubic B-splines. These curves are discussed in detail in many texts on computer-aided design [4, 12, 16].

The multiresolution framework described in Section 2.1 is very general. To construct our *multiresolution curves* from endpoint-interpolating cubic B-splines, we need to make several choices, as enumerated below:

1. *Choose the scaling functions $\Phi^j(u)$ for all j in $[0, n]$.*
   This choice determines the synthesis filters P^j. For each level j, we would like a basis for the endpoint-interpolating cubic B-spline curves with 2^j interior segments. The basis functions for these curves are the $2^j + 3$ endpoint-interpolating cubic B-splines, which are refinable, as required by equation (5).
2. *Select an inner product for any two functions f and g in V^j.*
   This choice determines the orthogonal complement spaces W^j. We use the standard form $\langle f\ ,\ g\rangle = \int f(u)g(u)du$.
3. *Select a set of wavelets $\Psi^j(u)$ that span W^j.*
   This choice determines the synthesis filters Q^j. Together, the synthesis filters P^j and Q^j determine the analysis filters A^j and B^j by equation (9). We use the set of 2^j *minimally-supported* functions that span W^j.

Appendix A contains more details on the specific wavelets we use and their derivation. A similar construction has also been independently proposed by Chui and Quak [9]. Note that multiresolution constructions can be built for other types of splines as well, such as uniform B-splines [8], and non-uniform B-splines with arbitrary knot sequences [21]. A recent construction applicable to subdivision surfaces is discussed by Lounsbery *et al.* [19].

Note that because both the scaling functions and wavelets in our construction have compact support, the synthesis filters P^j and Q^j have a banded structure, allowing reconstruction in $O(m)$ time. However, a potential weakness of our construction is that the analysis filters A^j and B^j are dense, which would seem to imply an $O(m^2)$-time decomposition algorithm. Fortunately, there is a clever trick, due to Quak and Weyrich [25], for performing the decomposition in linear time. The implementation of their algorithm is described in Appendix B.

3 Smoothing

In this section, we address the following problem: *Given a curve with m control points C, construct a best least-squares-error approximating curve with m' control points C', where $m' < m$.* Here, we will assume that both curves are endpoint-interpolating uniform B-spline curves.

The multiresolution analysis framework allows this problem to be solved trivially, for certain values of m and m'. Assume for the moment that $m = 2^j + 3$ and $m' = 2^{j'} + 3$ for some nonnegative integers $j' < j$. Then the control points C' of the approximating curve are given by

$$C' \quad = \quad A^{j'+1} A^{j'+2} \cdots A^j C$$

In other words, we simply run the decomposition algorithm, as described by equation (1), until a curve with just m' control points is reached. Note that this process can be performed at interactive speeds for hundreds of control points using the linear-time algorithm described in Appendix B.

One notable aspect of the multiresolution curve representation is its discrete nature. Thus, in our application it is easy to construct approximating curves with 4, 5, 7, 11, or any $2^j + 3$ control points efficiently, for any *integer level* j. However, there is no obvious way to quickly construct curves that have "levels" of smoothness in between.

The best solution we have found is to define a *fractional-level* curve $f^{j+t}(u)$ for some $0 \le t \le 1$ in terms of a linear interpolation between its two nearest integer-level curves $f^j(u)$ and $f^{j+1}(u)$, as follows:

$$\begin{aligned} f^{j+t}(u) &= (1-t)\, f^j(u) \ + \ t\, f^{j+1}(u) \\ &= (1-t)\, \Phi^j(u)\, C^j \ + \ t\, \Phi^{j+1}(u)\, C^{j+1} \end{aligned} \qquad (11)$$

These fractional-level curves allow for continuous levels of smoothing. In our application a user can move a control slider and see the curve transform continuously from its smoothest (4 control point) form, up to its finest (m control point) version. Some fractional-level curves are shown in Figure 2.

Figure 2: *Smoothing a curve continuously. From left to right: the original curve at level 8.0, and smoother versions at levels 5.4 and 3.1.*

4 Editing

Suppose we have a curve C^n and all of its low-resolution and detail parts $C^0, \ldots, C^{n-1}$ and $D^0, \ldots, D^{n-1}$. Multiresolution analysis allows for two very different kinds of curve editing. If we modify some low-resolution version C^j and then add back in the detail $D^j, D^{j+1}, \ldots, D^{n-1}$, we will have modified the overall sweep of the curve (Figure 3). On the other hand, if we modify the set of detail functions D^j, D^{j+1}, .., D^{n-1} but leave the low-resolution versions $C^0, \ldots, C^j$ intact, we will have modified the character of the curve, without affecting its overall sweep (Figure 6). These two types of editing are explored more fully below.

4.1 Editing the sweep

Editing the sweep of a curve at an integer level of the wavelet transform is simple. Let C^n be the control points of the original curve $f^n(u)$, let C^j be a low-resolution version of C^n, and let $\widehat{C}^j$ be an edited version of C^j, given by $\widehat{C}^j = C^j + \Delta C^j$. The edited version of the highest-resolution curve $\widehat{C}^n = C^n + \Delta C^n$ can be computed through reconstruction:

$$\begin{aligned} \widehat{C}^n &= C^n \ + \ \Delta C^n \\ &= C^n \ + \ P^n P^{n-1} \cdots P^{j+1}\, \Delta C^j \end{aligned}$$

Note that editing the sweep of the curve at lower levels of smoothing j affects larger portions of the high-resolution curve $f^n(u)$. At the lowest level, when $j = 0$, the entire curve is affected; at the highest level, when $j = n$, only the narrow portion influenced by one original control point is affected. The kind of flexibility that this multiresolution editing allows is suggested in Figures 3 and 4.

In addition to editing at integer levels of resolution, it is natural to ascribe meaning to editing at fractional levels as well. We would like the portion of the curve affected when editing at fractional level $j + t$ to interpolate the portions affected at levels j and $j + 1$. Thus, as t increases from 0 to 1, the portion affected should gradually narrow down from that of level j to that of level $j + 1$, as demonstrated in the lower part of Figure 4.

Consider a fractional-level curve $f^{j+t}(u)$ given by equation (11). Let C^{j+t} be the set of control points associated with this curve; that is,

$$f^{j+t}(u) \quad = \quad \Phi^{j+1}(u)\, C^{j+t} \qquad (12)$$

We can obtain an expression for C^{j+t} by equating the right-hand sides of equations (11) and (12), and then applying equations (5) and (3):

$$\begin{aligned} C^{j+t} &= (1-t)\, P^{j+1}\, C^j \ + \ t\, C^{j+1} \\ &= P^{j+1}\, C^j \ + \ t Q^{j+1}\, D^j \end{aligned}$$

Suppose now that one of the control points c_i^{j+t} is modified by the user. In order to allow the portion of the curve affected to depend on t in the manner described above, the system will have to automatically move some of the nearby control points when c_i^{j+t} is modified. The distance that each of these control points is moved is inversely proportional to t: for example,

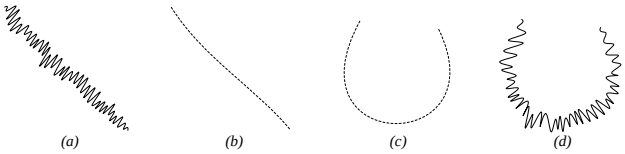


Figure 3: *Changing the overall sweep of a curve without affecting its character. Given the original curve (a), the system extracts the overall sweep (b). If the user modifies the sweep (c), the system can reapply the detail (d).*

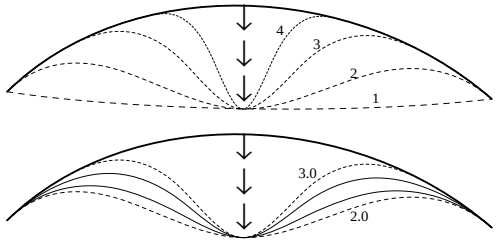


Figure 4: *The middle of the dark curve is pulled. Upper: Editing at integer levels 1, 2, 3, and 4. Lower: Editing at fractional levels between 2.0 and 3.0.*

when t is near 0, the control points in C^{j+t} are moved in conjunction so that the overall effect approaches that of editing a single control point at level j; when $t = 1$, the nearby control points are not moved at all, since the modified curve should correspond to moving just a single control point at level $j+1$.

Let ΔC^{j+t} be a vector describing how each control point of the fractional-level curve is modified: the i-th entry of ΔC^{j+t} is the user's change to the i-th control point; the other entries reflect the computed movements of the other control points. Rather than solving for ΔC^{j+t} explicitly, our approach will be to break this vector into two components, a vector ΔC^j of changes to the control points at level j, and a vector ΔD^j of changes to the wavelet coefficients at level j:

$$\Delta C^{j+t} \quad = \quad P^{j+1}\,\Delta C^j \;+\; t\,Q^{j+1}\,\Delta D^j \tag{13}$$

Next, define $\Delta \acute{C}^{j+t}$ to be the user's change to the control points at level $j+t$, that is, a vector whose i-th entry is Δc_i^{j+t}, and whose other entries are 0. Define also a new vector $\Delta \acute{C}^j$ as a change to control points at level j necessary to make the modified control point c_i^{j+t} move to its new position. We choose the vector that is 0 everywhere, except for one or two entries, depending on the index i of the modified control point. By examining the i-th row of the refinement matrix P^{j+1}, we can determine whether the modified control point is maximally influenced by either *one* control point c_k^{j+1} or *two* control points c_k^{j+1} and c_{k+1}^{j+1} at level $j+1$. In the former case, we set $\Delta \acute{c}_k^j$ to be $\Delta c_i^{j+t} / P_{i,k}^{j+1}$. In the latter case, we set $\Delta \acute{c}_k^j$ and $\Delta \acute{c}_{k+1}^j$ to be $\Delta c_i^{j+t} / 2P_{i,k}^{j+1}$.

Note that applying either change alone, $\Delta \acute{C}^{j+t}$ or $\Delta \acute{C}^j$, would cause the selected control point to move to its new position; however, the latter change would cause a larger portion of the curve to move. In order to have a "breadth" of change that gradually decreases as t goes from 0 to 1, we can interpolate between these two vectors, using some interpolation function $g(t)$:

$$\Delta C^{j+t} \quad = \quad (1 - g(t))\,P^{j+1}\,\Delta \acute{C}^j \;+\; g(t)\,\Delta \acute{C}^{j+t} \tag{14}$$

Thus, ΔC^{j+t} will still move the selected control point to its new position, and it will also now control the "breadth" of change as a function of t.

Finally, equating the right-hand sides of equations (13) and (14), multiplying with either A^{j+1} or B^{j+1}, and employing the identities (10) yields the two expressions we need:

$$\begin{aligned} \Delta C^j \quad &= \quad (1 - g(t))\,\Delta \acute{C}^j \;+\; g(t)\,A^{j+1}\,\Delta \acute{C}^{j+t} \\ \Delta D^j \quad &= \quad \frac{g(t)}{t}\,B^{j+1}\,\Delta \acute{C}^{j+t} \end{aligned} \tag{15}$$

We now have the choice of any function $g(t)$ that allows ΔD^j to increase monotonically from 0 to 1. The function $g(t) := t^2$ is an obvious choice that we have found to work well in practice.

The changes to the high-resolution control points are then reconstructed using a straightforward application of equation (3):

$$\Delta C^n \quad = \quad P^n\,P^{n-1}\cdots P^{j+2}\,(P^{j+1}\,\Delta C^j \;+\; Q^{j+1}\,\Delta D^j) \tag{16}$$

The fractional-level editing defined here works quite well in practice. Varying the editing level continuously gives a smooth and intuitive kind of change in the region of the curve affected, as suggested by Figure 4. Because the algorithmic complexity is just $O(m)$, the update is easily performed at interactive rates, even for curves with hundreds of control points.

4.1.1 Editing with direct manipulation

The fractional-level editing described above can be easily extended to accommodate *direct manipulation*, in which the user tugs on the smoothed curve directly rather than on its defining control points [3, 13, 15, 18]. To use direct manipulation when editing at level $j+t$, we make use of the pseudo-inverse of the scaling functions at levels j and $j+1$.

More precisely, suppose the user drags a point of the curve $f^{j+t}(u_0)$ to a new position $f^{j+t}(u_0) + \delta$. We can compute the least-squares change to the control points $\Delta \acute{C}^j$ and $\Delta \acute{C}^{j+t}$ at levels j and $j+t$ using the pseudo-inverses $(\Phi^j)^+$ and $(\Phi^{j+1})^+$ as follows:

$$\begin{aligned} \Delta \acute{C}^j \quad &= \quad (\Phi^j(u_0))^+\,\delta \\ \Delta \acute{C}^{j+t} \quad &= \quad (\Phi^{j+1}(u_0))^+\,\delta \end{aligned} \tag{17}$$

These two equations should be interpreted as applying to each dimension x and y separately. That is, δ should be a scalar (say, the change in x), and the left-hand side and the pseudo-inverses should both be column-matrices of scalars. The modified control points of the highest-resolution curve can then be computed in the same fashion outlined for control-point manipulation, by applying equations (15) and (16).

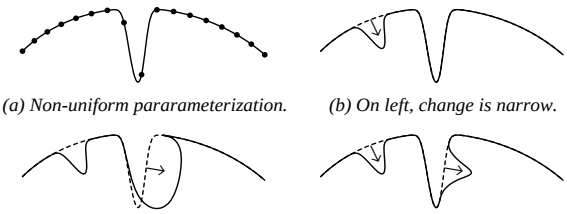
Note that the first step of the construction, equation (17), can be computed in constant time, since for cubic B-splines at most four of the entries of each pseudo-inverse are non-zero. The issue of finding the parameter value u_0 at which the curve passes closest to the selection point is a well-studied problem in root-finding, which can be handled in a number of ways [27]. In our implementation, we scan-convert the curve once to find its parameter value at every illuminated pixel. This approach is easy to implement, and appears to provide a good trade-off between speed and accuracy for an interactive system.

For some applications, it may be more intuitive to drag on the high-resolution curve directly, rather than on the smoothed version of the curve. In this case, even when the curve's display resolution is at its highest level, it may still be useful to be able to tug on the curve at a lower editing resolution. In this way, varying levels of detail on the curve can be manipulated by dragging a single point: as the editing resolution is lowered, more and more of the curve is affected. This type of control can be supported quite easily by setting δ to be the change in the high-resolution curve at the dragged point $f^n(u_0)$, and using the same equations (17) above.

4.1.2 Editing a desired portion of the curve

One difficulty with curve manipulation methods is that their effect often depends on the parameterization of the curve, which does not necessarily correspond to the curve's geometric embedding in an intuitive fashion. The manipulation that we have described so far suffers from this same difficulty: dragging at a particular (possibly fractional) level $\ell = j + t$ on different points along the curve will not necessarily affect constant-length portions of the curve. However, we can use the multiresolution editing control to compensate for this defect in direct manipulation, as follows (Figure 5).

Let h be a parameter, specified by the user, that describes the desired length of the editable portion of the curve. The parameter h can be specified using any type of physical units, such as screen pixels, inches, or percentage of



(a) Non-uniform pararameterization. *(b) On left, change is narrow.*

(c) Near middle, change is broad. *(d) Equal-breadth changes.*

Figure 5: *Curve (a) has a parameterization that is non-uniform with respect to its length. Direct manipulation on the left part of the curve (b) affects a much smaller fraction of the curve than does direct manipulation at the same level in the middle (c). The last figure (d) shows that a specified fraction of the curve can be edited, with the system determining the appropriate editing level.*

the overall curve length. The system computes an appropriate editing level ℓ that will affect a portion of the curve of about h units in length, centered at the point $f^n(u_0)$ being dragged.

We estimate ℓ as follows. For each integer-level editing resolution j, let $h^j(u_0)$ denote the length of $f^n(u)$ affected by editing the curve at the point $f^n(u_0)$. The length $h^j(u_0)$ is easily estimated by scan-converting the curve $f^n(u)$ to determine the approximate lengths of its polynomial segments, and then summing over the lengths of the segments affected when editing the curve at level j and parameter position u_0. Next, define j_- and j_+ to be, respectively, the smallest and largest values of j for which $h^{j-}(u_0) \geq h \geq h^{j+}(u_0)$. To choose the editing level ℓ, we use linear interpolation between these two bounding levels j_- and j_+:

$$\ell = \frac{h - h^{j+}}{h^{j-} - h^{j+}}$$

Finally, by representing ℓ in terms of an integer level j and fractional offset t, we can again apply equation (17), followed by equations (15) and (16), as before. Though in general this construction does not *precisely* cover the desired portion h, in practice it yields an intuitive and meaningful control. Figure 5 demonstrates this type of editing for a curve with an extremely non-uniform geometric embedding.

4.2 Editing the character of the curve

Another form of editing that is naturally supported by multiresolution curves is one of editing the character of a curve, without affecting its overall sweep. Let C^n be the control points of a curve, and let $C^0, \ldots, C^{n-1}$, $D^0, \ldots, D^{n-1}$ denote the components of its multiresolution decomposition. Editing the character of the curve is simply a matter of replacing the existing set of detail functions $D^j, \ldots, D^{n-1}$ with some new set $\widehat{D}^j, \ldots, \widehat{D}^{n-1}$, and reconstructing.

With this approach, we have been able to develop a "curve character library" that contains different detail functions, which can be interchangeably applied to any set of curves. The detail functions in the library have been extracted from hand-drawn strokes; other (for example, procedural) methods of generating detail functions are also possible. Figure 6 demonstrates how the character of curves in an illustration can be modified with the same (or different) detail styles. The interactive illustration system used to create this figure is described in a separate paper [26].

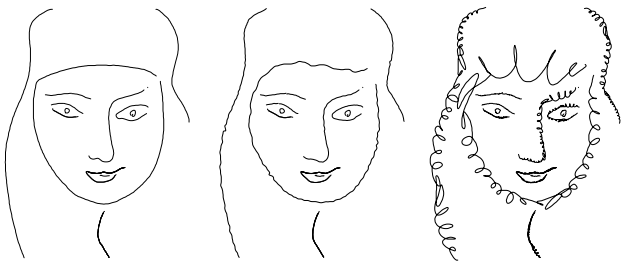
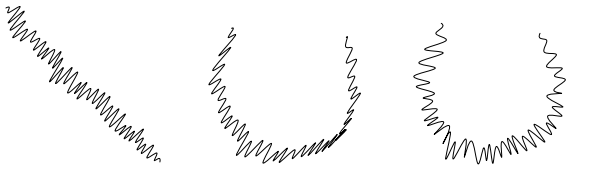


Figure 6: *Changing the character of a curve without affecting its sweep.*

4.3 Orientation of detail

A parametric curve in two dimensions is most naturally represented as two separate functions, one in x and one in y: $f(u) = (f_x(u), f_y(u))$. Thus, it seems reasonable to represent both the control points C^j and detail functions D^j using matrices with separate columns for x and y. However, encoding the detail functions in this manner embeds all of the detail of the curve in a particular xy-orientation. As demonstrated in Figure 7, this representation does not always provide the most intuitive control when editing the sweep of the curve.



Original curve. *Fixed xy-orientation.* *Detail relative to the tangent.*

Figure 7: *Editing the sweep of a curve using a fixed xy-orientation of detail versus orientation relative to the tangent of the curve.*

As an alternative, we employ a method similar to that of Forsey and Bartels [13] for representing detail with respect to the tangent and normal to the curve at a coarser level. Specifically, for computing the reference frame for orienting a detail coefficient d_i^j, we use the tangent and normal of the curve $f^{j-1}(u_0)$ at a parameter position u_0 corresponding to the maximum value of the wavelet $\psi_i^j(u)$. Note that the curve $f(u)$ is no longer a simple linear combination of the scaling functions Φ^0 and wavelets Ψ^j; instead, a change of coordinates must be performed at each level of reconstruction for the wavelet coefficients D^j. However, this process is linear in the number of control points, so it does not increase the computational complexity of the algorithm.

We have experimented with both normalized and unnormalized versions of the reference frame; the two alternative versions yield different but equally reasonable behavior. Figure 6 uses the unnormalized tangents whereas the rest of the figures in this paper use normalized tangents.

5 Scan conversion and curve compression

Using "curve character libraries" and other multiresolution editing features, it is easy to create very complex curves with hundreds or potentially thousands of control points. In many cases (such as in this paper), these curves are printed in a very small form. Conventional scan conversion methods that use all the complexity of these curves are wasteful, both in terms of the network traffic to send such large files to the printer, and in terms of the processing time required by the printer to render curves of many control points within a few square pixels. We therefore explore a form of curve compression that is suitable for the purposes of scan conversion. The algorithm requires an approximate curve to have a guaranteed error tolerance, in terms of printer pixels, from the original curve. However, it does not require any particular continuity constraints, as are usually required in data-fitting applications.

As discussed in Section 3, the simple removal of wavelet coefficients can be used to achieve a least-squares, or L^2, error metric between an original curve and its approximate versions. However, for scan conversion, an L^2 error metric is not very useful for measuring the degree of approximation: an approximate curve $\widetilde{f}(u)$ can be arbitrarily far from an original curve $f^n(u)$ and still achieve a particular L^2 error bound, as long as it deviates from the original over a small enough segment. In order to scan convert a curve to some guaranteed precision—measured, say, in terms of maximum deviation in printer pixels—we need to use an L^∞ norm on the error. There are many ways to achieve such a bound. The method described here is a simple and fast one, although methods with higher compression ratios are certainly possible.

Let s_i^j (with $0 \leq i \leq 2^j - 1$) be a segment of the cubic B-spline curve $f^j(u)$, defined by the four control points $c_i^j, \ldots, c_{i+3}^j$. Note that each segment s_i^j corresponds to exactly two segments s_{2i}^{j+1} and s_{2i+1}^{j+1} at level $j+1$.

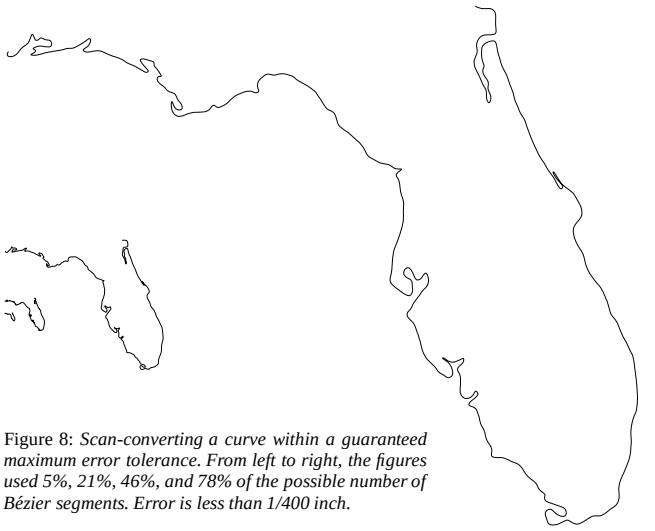


Figure 8: *Scan-converting a curve within a guaranteed maximum error tolerance. From left to right, the figures used 5%, 21%, 46%, and 78% of the possible number of Bézier segments. Error is less than 1/400 inch.*

Figure 9: *Same curves as above, but drawn at constant size.*

Our objective is to build a new approximating curve $\widetilde{f}(u)$ for $f(u)$ by choosing different segments at different levels such that $||\widetilde{f}(u) - f^n(u)||_\infty$ is less than some user-specified ϵ for all values of u.

Assume, for the moment, that we have some function *ErrBound*(s_i^j) that returns a bound on the L^∞ error incurred from using the segment s_i^j of some approximate curve $f^j(u)$ in place of the original segments of $f^n(u)$ to which it corresponds. We can scan-convert a curve to within any error tolerance ϵ by passing to the recursive routine *DrawSegment* the single segment s_0^0 corresponding to the lowest-level curve $f^0(u)$. This routine recursively divides the segment to varying levels so that the collection of segments it produces approximates the curve to within ϵ.

procedure *DrawSegment*(s_i^j):
  if *ErrBound*$(s_i^j) < \epsilon$ **then**
    Output segment s_i^j as a portion of $\widetilde{f}(u)$
  else
    DrawSegment(s_{2i}^{j+1}); *DrawSegment*(s_{2i+1}^{j+1})
  end if
end procedure

To construct the *ErrBound* routine, let M^j be the B-spline-to-Bézier-basis conversion matrix [4] for curves with $2^j + 3$ control points, and let E^j be a column vector with entries e_i^j defined by

$$E^j \quad := \quad M^j \, Q^j \, D^{j-1} \tag{18}$$

The vector E^j provides a measure of the distance that the Bézier control points migrate when reconstructing the more detailed curve at level j from the approximate curve at level $j-1$. Since Bézier curves are contained within the convex hull of their control points, the magnitudes of the entries of E^j provide conservative bounds on approximations to the curve due to truncating wavelet coefficients.

A bound δ_i^j on the L^∞ error incurred by replacing segment s_i^j with its approximation at level $j-1$ is given by

$$\delta_i^j \quad \leq \quad max_{\,i \leq k \leq i+3} \left\{ \left|\left| e_k^j \right|\right|_2 \right\} \tag{19}$$

The *ErrBound* routine can then be described recursively as follows:

procedure *ErrBound*(s_i^j):
  if $j = n$ **then**
    return 0
  else
    return $max\{$*ErrBound*$(s_{2i}^{j+1}) + \delta_{2i}^{j+1}$, *ErrBound*$(s_{2i+1}^{j+1}) + \delta_{2i+1}^{j+1}\}$
  end if
end procedure

An efficient implementation of the *ErrBound* routine would use dynamic programming or an iterative (rather than recursive) procedure to avoid recomputing error bounds. In practice, the routine is fast enough in its recursive form that we have not found this optimization to be necessary, at least for scan converting curves with hundreds of control points.

The approximate curve $\widetilde{f}(u)$ is described by a set of Bézier segments, which we use to generate a PostScript file [1]. Note that the scan-conversion algorithm, as described, produces approximate curves $\widetilde{f}(u)$ that are not even C^0 continuous where two segments of different levels abut. Since we are only concerned with the absolute error in the final set of pixels produced, relaxing the continuity of the original curve is reasonable for scan conversion. We can achieve C^0 continuity, however, without increasing the prescribed error tolerance, by simply averaging together the end control points for adjacent Bézier segments as a post-process. We have found that these C^0 curves look slightly better than the discontinuous curves; they also have a more compact representation in PostScript. Figures 8 and 9 demonstrate compression of the same curve rendered at different sizes.

6 Extensions and future work

This paper describes a multiresolution representation for endpoint-interpolating B-spline curves, and shows how this single representation supports a variety of display and editing operations in a simple and efficient manner. We believe that the operations described are very general and can be readily extended to other types of objects described by a multiresolution analysis.

There are many directions for future research, including:

Handling discontinuities. An important extension is to generalize the multiresolution curve representation and editing operations to respect discontinuities of various orders that have been intentionally placed into a curve by the designer. This extension would allow the techniques to be applied more readily to font design, among other applications. One approach is to try using the multiresolution analysis defined on non-uniform B-splines by Dæhlen and Lyche [10].

Sparse representations. Our algorithms have so far used only *complete* wavelet decompositions of the curve's original control points. However, in order to support curve editing at an arbitrarily high resolution, it would be convenient to have a mechanism in place for extending the wavelet representation to a higher level of detail in certain higher-resolution portions of the curve than in others. One such sparse representation might use pruned binary trees to keep track of the various wavelet coefficients at different levels of refinement, in a manner very similar to the one used by Berman *et al.* for representing multiresolution images [5].

Textured strokes. For illustrations, it is useful to associate other properties with curves, such as color, thickness, texture, and transparency, as demonstrated by Hsu and Lee [17]. These quantities may be considered extra dimensions in the data associated with each control point. Much of the machinery for multiresolution editing should be applicable to such curves. As a preliminary test of this idea, we have extended our curve editor with a *thickness* dimension. The thickness along the curve is governed by the thick-

Figure 10: *Two curves of varying thickness.*

nesses defined at the control points. It is possible to modify this parameter at any level of resolution, just as one edits the position of the curve. Figure 10 shows curves with varying thickness. Ultimately, we would like to combine stroke editing with multiresolution image editing [5], perhaps providing a unified framework for object-oriented ("MacDraw-like") and image-oriented ("MacPaint-like") interactive design programs.

Surfaces. Another obvious extension of these techniques is to surfaces. As a test of multiresolution surface editing, we built a surface editor that allows a user to modify a bicubic tensor-product B-spline surface [4, 12, 16] at different levels of detail. Figure 11 shows several manipulations applied to a surface over 1225 control points modeling a human face. It is worth noting that tensor-product surfaces are limited in the kinds of shapes they can model seamlessly. Lounsbery *et al.* [19] discuss a multiresolution representation for subdivision surfaces of arbitrary topology. Many of the techniques described in this paper should extend directly to their surfaces as well. In particular, fractional-level display and editing are applicable in the same way as for curves and tensor-product surfaces. In addition, the compression technique for scan-converting curves might also be used for rendering simplified versions of polyhedra within guaranteed error tolerances.

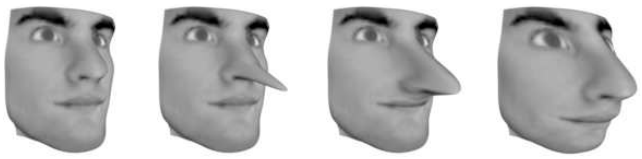


Figure 11: *Surface manipulation at different levels of detail. From left to right: original, narrow change, medium change, broad change.*

Acknowledgements

We would like to thank Tony DeRose, Ronen Barzel, and Leena-Maija Reissell for very helpful discussions during the development of these ideas, and Sean Anderson for implementing the tensor-product B-spline surface editor.

This work was supported by an NSF National Young Investigator award (CCR-9357790), by the University of Washington Graduate Research and Royalty Research Funds (75-1721 and 65-9731), and by industrial gifts from Adobe, Aldus, and Xerox.

References

[1] *PostScript Language Reference Manual*. Addison-Wesley Publishing Company, Inc., 1985.

[2] M. J. Banks and E. Cohen. Realtime spline curves from interactively sketched data. *Computer Graphics*, 24(2):99–107, 1990.

[3] R. Bartels and J. Beatty. A technique for the direct manipulation of spline curves. In *Proceedings of the 1989 Graphics Interface Conference*, pages 33–39, London, Ontario, Canada, June 1989.

[4] R. Bartels, J. Beatty, and B. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1987.

[5] D. Berman, J. Bartell, and D. Salesin. Multiresolution painting and compositing. Proceedings of SIGGRAPH 94. In *Computer Graphics*, Annual Conference Series, 1994.

[6] G. Beylkin, R. Coifman, and V. Rokhlin. Fast wavelet transforms and numerical algorithms I. *Communications on Pure and Applied Mathematics*, 44:141–183, 1991.

[7] G. Celniker and D. Gossard. Deformable curve and surface finite elements for free-form shape design. *Computer Graphics*, 25(4):257–265, July 1991.

[8] C. K. Chui. *An Introduction to Wavelets*. Academic Press, Inc., Boston, 1992.

[9] C. K. Chui and E. Quak. Wavelets on a bounded interval. In D. Braess and L. L. Schumaker, editors, *Numerical Methods in Approximation Theory*, volume 9, pages 53–75. Birkhauser Verlag, Basel, 1992.

[10] M. Dæhlen and T. Lyche. Decomposition of splines. In T. Lyche and L. L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design II*, pages 135–160. Academic Press, New York, 1992.

[11] R. DeVore, B. Jawerth, and B. Lucier. Image compression through wavelet transform coding. *IEEE Transactions on Information Theory*, 38(2):719–746, March 1992.

[12] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, third edition, 1992.

[13] D. Forsey and R. Bartels. Hierarchical B-spline refinement. *Computer Graphics*, 22(4):205–212, 1988.

[14] D. Forsey and R. Bartels. Tensor products and hierarchical fitting. In *Curves and Surfaces in Computer Vision and Graphics II, SPIE Proceedings Vol. 1610*, pages 88–96, 1991.

[15] B. Fowler. Geometric manipulation of tensor product surfaces. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, March 1992. Available as Computer Graphics, Vol. 26, No. 2.

[16] J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. A K Peters, Ltd., Wellesley, Massachusetts, third edition, 1992.

[17] S. Hsu and I. Lee. Skeletal strokes. Proceedings of SIGGRAPH 94. In *Computer Graphics*, Annual Conference Series, 1994.

[18] W. M. Hsu, J. F. Hughes, and H. Kaufman. Direct manipulation of free-form deformations. *Computer Graphics*, 26(2):177–184, 1992.

[19] M. Lounsbery, T. DeRose, and J. Warren. Multiresolution surfaces of arbitrary topological type. Technical Report 93-10-05B, University of Washington, Department of Computer Science and Engineering, January 1994.

[20] T. Lyche and K. Mørken. Knot removal for parametric B-spline curves and surfaces. *Computer Aided Geometric Design*, 4(3):217–230, 1987.

[21] T. Lyche and K. Mørken. Spline-wavelets of minimal support. In D. Braess and L. L. Schumaker, editors, *Numerical Methods in Approximation Theory*, volume 9, pages 177–194. Birkhauser Verlag, Basel, 1992.

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

[23] M. Plass and M. Stone. Curve-fitting with piecewise parametric cubics. *Computer Graphics*, 17(3):229–239, July 1983.

[24] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Fetterling. *Numerical Recipes*. Cambridge University Press, Cambridge, second edition, 1992.

[25] E. Quak and N. Weyrich. Decomposition and reconstruction algorithms for spline wavelets on a bounded interval. CAT Report 294, Center for Approximation Theory, Texas A&M University, April 1993.

[26] M. P. Salisbury, S. E. Anderson, R. Barzel, and D. H. Salesin. Interactive pen-and-ink illustration. Proceedings of SIGGRAPH 94. In *Computer Graphics*, Annual Conference Series, 1994.

[27] P. J. Schneider. Phoenix: An interactive curve design system based on the automatic fitting of hand-sketched curves. Master's thesis, Department of Computer Science and Engineering, University of Washington, 1988.

[28] W. Welch and A. Witkin. Variational surface modeling. *Computer Graphics*, 26(2):157–166, 1992.

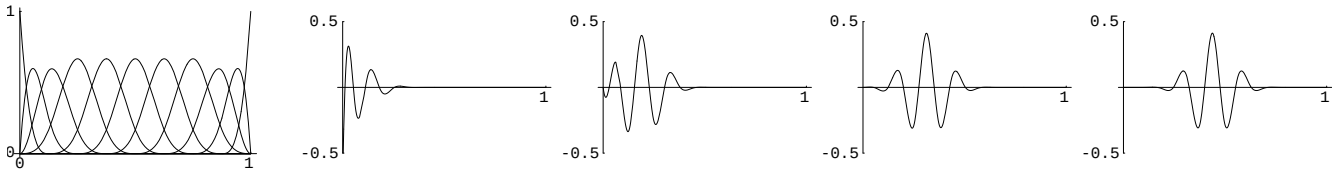


Figure 12: *The B-spline scaling functions and the first four wavelets at level 3.*

A Wavelets for endpoint-interpolating B-splines

As discussed in Section 2.1, a multiresolution analysis is completely determined by an initial set of scaling functions Φ^0 and a pair of synthesis filters P^j and Q^j for every level j in $[1, n]$. This appendix supplies these functions and matrices for endpoint-interpolating cubic B-splines, and outlines their derivation. Figure 12 shows some examples of these B-spline scaling functions and wavelets.

Initial scaling functions are given by the four cubic Bernstein polynomials:

$$\Phi^0(u) \quad = \quad \left[(1-u)^3,\ 3u(1-u)^2,\ 3u^2(1-u),\ u^3\right]$$

The matrices P^j and Q^j appear in Figure 13. Note that P^j is a matrix with dimensions $(2^j+3) \times (2^{j-1}+3)$ whose middle columns, for $j \geq 3$, are given by vertical translates of the fourth column, shifted down by 2 places for each column. Matrix Q^j has the same structure for $j \geq 4$, except with dimensions $(2^j+3) \times 2^{j-1}$.

The P^j matrix is straightforward to derive from the Cox-de Boor recursion formula [12]; it encodes how each endpoint-interpolating B-spline can be expressed as a linear combination of B-splines that are half as wide. To derive the Q^j matrix, we use some new notation. Given two row vectors of functions X and Y, let $[\langle X \mid Y \rangle]$ be the matrix of inner products $\langle X_k\ ,\ Y_l \rangle$. Since, by definition, scaling functions and wavelets at the same level j are orthogonal, we have

$$\left[\langle \Phi^j \mid \Psi^j \rangle\right] \quad = \quad \left[\langle \Phi^j \mid \Phi^{j+1} \rangle\right] Q^{j+1} \quad = \quad \mathbf{0},$$

so the columns of Q^{j+1} span the null space of $\left[\langle \Phi^j \mid \Phi^{j+1} \rangle\right]$. We choose a basis for this null space by finding the matrix Q^{j+1} that has columns with the shortest runs of non-zero coefficients; this matrix corresponds to the wavelets with minimal support. The entries of the inner product matrix can be computed exactly with symbolic integration; thus, the fractions reported in Figure 13 are exact (though ugly).

B Linear-time filter-bank algorithm

Section 2.2 notes that the obvious filter-bank decomposition algorithm for endpoint-interpolating B-spline curves takes $O(m^2)$-time because A^j and B^j are dense. However, Quak and Weyrich [25] describe an algorithm for performing the algorithm in linear time, using a transformation to the "dual space." The derivation of this idea is beyond the scope of this paper; however, for completeness, we summarize here how the linear-time algorithm can be implemented.

Let I^j and J^j be the inner product matrices $\left[\langle \Phi^j \mid \Phi^j \rangle\right]$ and $\left[\langle \Psi^j \mid \Psi^j \rangle\right]$, respectively. Equations (1) and (2) can then be rewritten:

$$\begin{aligned} I^{j-1} C^{j-1} \quad &= \quad (P^j)^T I^j C^j \\ J^{j-1} D^{j-1} \quad &= \quad (Q^j)^T I^j C^j \end{aligned}$$

Since P^j, Q^j, and I^j are banded matrices, the right-hand side of these equations can be computed in linear time. What remains are two band-diagonal systems of equations, which can also be solved in linear time using LU decomposition [24].

The matrices I^j for $j \geq 3$ are given in Figure 13. Note that I^j is a symmetric matrix with dimensions $(2^j+3) \times (2^j+3)$ whose middle columns, for $j \geq 3$, are given by vertical translates of the sixth column. The I^j matrices for $j < 3$ and the J^j matrices may be found by:

$$\begin{aligned} I^j \quad &= \quad (P^{j+1})^T I^{j+1} P^{j+1} \\ J^j \quad &= \quad (Q^{j+1})^T I^{j+1} Q^{j+1} \end{aligned}$$

$$P^1 = \frac{1}{16}\begin{bmatrix} 16 & 0 & 0 & 0 \\ 8 & 8 & 0 & 0 \\ 0 & 8 & 8 & 0 \\ 0 & 0 & 8 & 8 \\ 0 & 0 & 0 & 16 \end{bmatrix}$$

$$P^2 = \frac{1}{16}\begin{bmatrix} 16 & 0 & 0 & 0 & 0 \\ 8 & 8 & 0 & 0 & 0 \\ 0 & 12 & 4 & 0 & 0 \\ 0 & 3 & 10 & 3 & 0 \\ 0 & 0 & 4 & 12 & 0 \\ 0 & 0 & 0 & 8 & 8 \\ 0 & 0 & 0 & 0 & 16 \end{bmatrix}$$

$$P^{j\geq 3} = \frac{1}{16}\begin{bmatrix} 16 & 0 & 0 & 0 & 0 & 0 & \\ 8 & 8 & 0 & 0 & 0 & 0 & \\ 0 & 12 & 4 & 0 & 0 & 0 & \\ 0 & 3 & 11 & 2 & 0 & 0 & \\ 0 & 0 & 8 & 8 & 0 & 0 & \cdots \\ 0 & 0 & 2 & 12 & 2 & 0 & \\ 0 & 0 & 0 & 8 & 8 & 0 & \\ 0 & 0 & 0 & 2 & 12 & 2 & \\ 0 & 0 & 0 & 0 & 8 & 8 & \\ 0 & 0 & 0 & 0 & 2 & 12 & \\ & & & \vdots & & & \ddots \end{bmatrix}$$

$$Q^1 = \frac{1}{3}\begin{bmatrix} 1 \\ -2 \\ 3 \\ -2 \\ 1 \end{bmatrix}$$

$$Q^2 = \frac{1}{2064}\begin{bmatrix} -1368 & 0 \\ 2064 & 240 \\ -1793 & -691 \\ 1053 & 1053 \\ -691 & -1793 \\ 240 & 2064 \\ 0 & -1368 \end{bmatrix}$$

$$Q^3 = \begin{bmatrix} \frac{-394762}{574765} & 0 & 0 & 0 \\ 1 & \frac{-7166160}{28124263} & 0 & 0 \\ \frac{-33030599}{41383080} & \frac{333497715}{478112471} & \frac{6908335}{478112471} & 0 \\ \frac{633094403}{1655323200} & \frac{-881412943}{956224942} & \frac{-74736797}{956224942} & \frac{27877}{1655323200} \\ \frac{-19083341}{137943600} & 1 & \frac{8833647}{28124263} & \frac{-864187}{413830800} \\ \frac{4681957}{165532320} & \frac{-689203555}{956224942} & \frac{-689203555}{956224942} & \frac{4681957}{165532320} \\ \frac{-864187}{413830800} & \frac{8833647}{28124263} & 1 & \frac{-19083341}{137943600} \\ \frac{27877}{1655323200} & \frac{-74736797}{956224942} & \frac{-881412943}{956224942} & \frac{633094403}{1655323200} \\ 0 & \frac{6908335}{478112471} & \frac{333497715}{478112471} & \frac{-33030599}{41383080} \\ 0 & 0 & \frac{-7166160}{28124263} & 1 \\ 0 & 0 & 0 & \frac{-394762}{574765} \end{bmatrix}$$

$$Q^{j\geq 4} = \begin{bmatrix} \frac{-394762}{574765} & 0 & 0 & 0 \\ 1 & \frac{-1050072320}{4096633377} & 0 & 0 \\ \frac{-33030599}{41383080} & \frac{2096854390}{2989435167} & \frac{307090}{19335989} & 0 \\ \frac{633094403}{1655323200} & \frac{-11070246427}{11957740668} & \frac{-6643465}{77343956} & \frac{-1}{24264} \\ \frac{-19083341}{137943600} & 1 & \frac{6646005}{19335989} & \frac{31}{6066} \\ \frac{4681957}{165532320} & \frac{-157389496903}{221218202358} & \frac{-29839177}{38671978} & \frac{-559}{8088} \\ \frac{-864187}{413830800} & \frac{1732435193}{5821531641} & 1 & \frac{988}{3033} \\ \frac{27877}{1655323200} & \frac{-27809640281}{44243640471 6} & \frac{-58651607}{77343956} & \frac{-9241}{12132} \\ 0 & \frac{171326708}{36869700393} & \frac{6261828}{19335989} & 1 \\ 0 & \frac{-1381667}{36869700393} & \frac{-1328199}{19335989} & \frac{-9241}{12132} \\ 0 & 0 & \frac{98208}{19335989} & \frac{988}{3033} \\ 0 & 0 & \frac{-792}{19335989} & \frac{-559}{8088} \\ 0 & 0 & 0 & \frac{31}{6066} \\ 0 & 0 & 0 & \frac{-1}{24264} \\ & & \vdots & & \ddots \end{bmatrix}$$

$$I^{j\geq 3} = \frac{1}{10080 \cdot 2^j}\begin{bmatrix} 1440 & 882 & 186 & 12 & 0 & 0 & \\ 882 & 2232 & 1575 & 348 & 3 & 0 & \\ 186 & 1575 & 3294 & 2264 & 239 & 2 & \\ 12 & 348 & 2264 & 4832 & 2382 & 240 & \cdots \\ 0 & 3 & 239 & 2382 & 4832 & 2382 & \\ 0 & 0 & 2 & 240 & 2382 & 4832 & \\ 0 & 0 & 0 & 2 & 240 & 2382 & \\ 0 & 0 & 0 & 0 & 2 & 240 & \\ 0 & 0 & 0 & 0 & 0 & 2 & \\ & & \vdots & & & & \ddots \end{bmatrix}$$

Figure 13: *The synthesis filters P^j and Q^j and the inner product matrices I^j.*