Variational Modeling With Wavelets

Steven J. Gortler and Michael F. Cohen

Department of Computer Science
Princeton University

Abstract

In many geometric modeling paradigms the user sculpts a curve or surface by dragging around some type of control points (eg. Bezier or B-spline). A more intuitive modeling interface allows the user to directly manipulate curves and surfaces. This manipulation defines some set of constraints that the curve or surface must satisfy (such as interpolation and tangent constraints). Direct manipulation, however, usually leads to an underconstrained problem since there are, in general, many possible surfaces meeting some set of constraints. Finding the "best" solution requires solving a variational problem. Unfortunately, this can be costly to compute. In particular, iterative descent methods converge slowly when a finite element basis such as B-splines is used.

This paper discusses how geometric variational modeling problems can be solved more efficiently by using a wavelet basis. Because the wavelet basis is hierarchical, the iterative methods converge rapidly. And because the wavelet coefficients indicate the degree of detail in the solution, they can be used to determine the number of basis functions needed to express the variational minimum, thus avoiding unnecessary computation. An implementation is discussed and experimental results are reported.

1 Introduction

In geometric modeling it is desirable to have the ability to directly manipulate curves and surfaces. This can provide a better interface than having a user interact with some set of control points and blending functions. Direct manipulation, however, leads to an underdetermined problem since there are, in general, many possible curves or surfaces meeting a user's constraints. Of all answers that meet the constraints, the user desires the "best" solution. The notion of best, which is formally defined as the solution that minimizes some *energy function*, is often taken to mean the smoothest solution. Finding the minimum-energy solution requires solving a variational modeling problem. There are many applications of variational modeling which include surface design [22, 2, 27, 18, 13], surface animation [28, 10] and surface reconstruction for vision problems [25, 26],

In theory, the desired solution is the curve or surface that has the minimum energy of *all* possible curves or surfaces that meet the constraints. Unfortunately there is little hope to find a closed form solution. Therefore, in practice, the "space" of parametric curves or surfaces is restricted to those represented by a linear combination of a fixed set of basis functions such as cubic B-splines. Given a set of n basis functions, the goal of finding the best curve or surface is then reduced to that of finding the best set of n coefficients. This reduction is referred to as the *finite element method* [25]. Casting the problem in terms of a finite number of unknown coefficients creates a well defined constrained optimization problem, however, it is one that is often costly to solve. The general case requires solving a non-linear optimization problem. In the best case, the energy function is quadratic and the constraints are linear. In this case only one linear system must be solved. But even this can be costly when n is large since direct methods for matrix inversion require $O(n^3)$ time. To accelerate this process it is tempting to use gradient-type iterative methods to solve the linear system. These methods only take $O(n)$ time per iteration, since finite element matrices for optimization problems are $O(n)$ sparse.

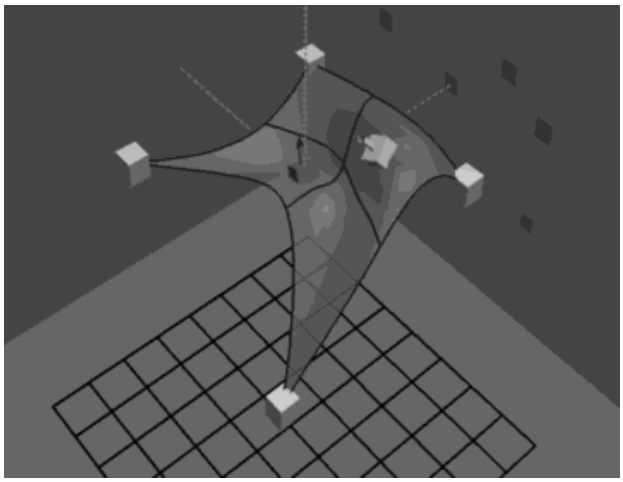

Figure 1: Sculpting a surface using a direct manipulation modeling tool.

Unfortunately, the linear systems arising from a finite element formulation are often expensive to solve using iterative methods. This is because the systems are ill-conditioned, and thus require many iterations to converge to a minimum [24, 23]. Intuitively speaking this occurs because each basis function represents a very narrow region of the answer; there is no basis function which can be moved to change the answer in some broad manner. For example, changing one coefficient in a cubic B-spline curve during an iteration alters the curvature in a local region only. In order to produce a broad smooth curve, the coefficients of the neighboring B-splines will move in next few iterations. Over the next many iterations, the solution process will affect wider and wider regions, and the effect will spread out slowly like a wave moving along a string. The result is very slow convergence (see Figure (7)). This problem also arises in other fields when finite elements are used for solving differential equations or doing surface reconstruction. One method used to combat this problem is multigridding [24, 10], where a sequence of problems at different resolution levels are posed and solved.

An alternative approach, which has been shown to be successful in those domains, is to use a *wavelet* basis instead of a standard finite element basis [23, 21, 14, 19]. In a wavelet basis, the answer is represented hierarchically. This allows the solution method to alter the answer at any desired resolution by altering the proper basis function, and thus the ill-conditioning is avoided.

In this paper we show how to use a wavelet construction, which is based on cubic B-splines, to quickly solve variational modeling problems in an elegant fashion. The wavelets described here are

derived in [5]. They are based on cubic B-splines, and hence are a natural choice for geometric modeling.

Another problem with the finite element approach is choosing the density of the basis functions. If too few basis functions (too few b-spline segment or tensor product b-spline patches) are used then the solution obtained will be far from the actual minimum. If too many basis functions are used then unnecessary computation will be performed during each iteration (n is too big). In order to successfully choose a proper density, one must know how much detail exists in the variational minimum answer. Since, a priori, this decision is difficult, an efficient solver must be able to adaptively change the basis during the solution process [27]. To do this effectively one needs an easy way to detect that too many or too few basis functions are being used. In addition, one needs a basis for which adding more detail, (i.e., refinement), is easy. Wavelets offer a basis where this task can be accomplished quickly and elegantly. During the solution process, the wavelet coefficients themselves signify what level of detail is being used in each region, making the adaptive decision easy. And in a wavelet basis, refinement is also achieved directly; all that must be done is to add a new wavelet basis function to the set.

The work presented in this paper combines the wavelet-hierarchical approaches of [23] and [12]. Like [23], this paper uses hierarchical basis functions as a pre-conditioner, so that fewer iterations are needed for convergence. Similar to [12], wavelets are also used as a method for limiting the solution method to the proper level of detail. By doing this a smaller (sparser) linear system is obtained. And this smaller system requires less computation per iteration. As shown by our experimental results, this wavelet method offers the ability to quickly obtain variational modeling solutions.

2 Variational Modeling

This section reviews the description of the variational modeling problem, and how it is solved using the finite element method. For more details see [27, 18, 2, 25]. In the standard variational modeling problem, a user wants to compute the smoothest curve

$$\gamma(t) = \langle x(t), y(t), z(t) \rangle \tag{1}$$

or surface

$$\gamma(s,t) = \langle x(s,t), y(s,t), z(s,t) \rangle \tag{2}$$

that meets some constraints. There are a number of measures of smoothness associated with a curve or surface. One convenient measure is the *thin plate* functional related to the square of the second derivative. Thus, for curves, this is [1]

$$E(\gamma) = \int_\Gamma dt \parallel \ddot{\gamma}(t)) \parallel_2^2 \tag{3}$$

and for surfaces is

$$E(\gamma) = \int_\Gamma dt \ ds \ \ \parallel \gamma_{tt}(t,s) \parallel_2^2 + 2 \parallel \gamma_{ts}(t,s) \parallel_2^2 + \parallel \gamma_{ss}(t,s) \parallel_2^2 \tag{4}$$

Other functionals include the *minimum variation surface* measure of [18] and the measures given in [22]. These functionals can also have variable weights over s and t, allowing the modeling material to be "stiffer" in some regions, and more flexible (and even creasable) in others [25]. The thin plate functional does not always produce results that are as desirable as some of the other measures [16], but it is often used because it has only quadratic terms, and thus leads to a quadratic programming problem. The wavelet formulation discussed in this paper is completely general, and is applicable for all of the smoothness functionals.

[1] the norm brackets "$\|_2$" are used since γ is a 3-vector.

Given some set of constraints and an energy functional E, the desired solution is the curve or surface that minimizes E while satisfying the constraints. Although this is a well defined problem of variational calculus, there is little hope to find a closed form solution [2]. In order to find a good approximate minimum solution, the curve or surface is limited to those answers that can be represented by a linear combination of a fixed set of basis functions, B_j, such as cubic B-splines. For curves this means [3]

$$x(t) = \sum_j c_j B_j(t) \tag{5}$$

and for surfaces

$$x(s,t) = \sum_{j,k} c_{j,k} B_{j,k}(s,t) \tag{6}$$

where c are scalar coefficients. With this simplification, instead of searching for the minimum of the space of all possible curves or surfaces, we only consider answers that are linear combinations of the fixed set of basis functions. The variational problem then becomes an optimization problem over a discrete set of variables c, to which a variety of numerical optimization techniques can be applied.

For example, given linear constraints $Ac = b$ and a quadratic energy functional (like the thin plate term), the optimal answer in the given space can be computed as the solution to the linear system

$$\begin{vmatrix} H & A^T \\ A & 0 \end{vmatrix} \begin{vmatrix} c \\ \lambda \end{vmatrix} = \begin{vmatrix} 0 \\ b \end{vmatrix} \tag{7}$$

where λ are lagrange multipliers, and where H is the Hessian matrix (partial second derivatives) of the energy function with respect to the coefficient variables [27]. For thin plate curves the Hessian terms are

$$H_{i,j} = \frac{\partial^2 E}{\partial c_i \partial c_j} = \int dt \ \ \ddot{B}_i(t) \ddot{B}_j(t) \tag{8}$$

One example of a linear constraint is that the curve at some fixed parameter value t_i interpolates the point b_i in space.

$$\sum_j c_j B_j(t_i) = b_i \tag{9}$$

Other types of constraints include tangent conditions, and curve interpolation (in surface design) [27]. If the functional is not quadratic, or the constraints not linear, then a non-linear optimization problem must be solved. Solution techniques such as sequential quadratic programming [11], lead to a series of linear systems with a form like Equation (7).

A common choice of a basis for variational modeling is the uniform cubic B-spline basis [4] [1]

$$B_j(t) = \phi(t - j) \tag{10}$$

where all of the basis functions are translations of one B-spline shape $\phi(t)$, which is made up of 4 cubic segments joined by simple knots at integer values. An advantage to this basis is that the linear system obtained in Equation (7) is $O(n)$ sparse; for example the Hessian terms defined in Equation (8) are only non-zero when the support of B_i and B_j overlap. And so for cubic B-spline curves there are 7 non-zero entries per row of H, and for tensor product cubic B-spline surfaces there are 49 non-zero entries per row. This means that when using some iterative method such as the conjugate gradient method [20], each iteration, which is computed by doing a few matrix-vector multiplies, can be performed in linear time. Unfortunately as the number of basis functions in the set

[2] Although see [25] for a discussion of finding closed form solutions

[3] From now on, only the x components of γ will be discussed. It is understood that y and z are handled equivalently.

[4] hereafter referred to simply as the B-spline basis

grows, the system becomes poorly conditioned resulting in the need for too many iterations to find a solution [24, 23].

One of the positive qualities of the B-spline basis, local control, is a disadvantage in the optimization framework. Since a change in a single coefficient (control point) effects only a very limited section of the curve or surface, there is a conflict between moving a control point to meet a positional constraint and maintaining a smooth curve. This causes the iterative optimization procedures to perform a series of small changes to a sequence of control points. Fewer wider basis functions would provide a better means to control the curve or surface as a whole, but this would limit the ability to model fine detail in certain regions of the curve or surface. What is needed is a hierarchical description of the curve, where some of the basis functions represent a coarse/broad description of the answer, while other basis functions refine the curve and describe a higher degree of detail.

A hierarchical description is also useful for the iterative solution method to be able to dynamically adapt to the complexity of the answer and best approximate the true variational minimum. This adaptivity can be done easily when using a hierarchical description; a few wide (or coarse) basis functions can be used until the solution procedure decides to add detail in some region.

Wavelets offer such a hierarchical basis for representing a curve or surface. Unlike hierarchical B-splines [9], which over-represent a curve or surface by using B-spline functions of different widths, wavelets offer a unique representation of a curve or surface. The wavelet coefficients directly encode the necessary detail at various degrees of resolution, and thus allow solutions of variational problems to proceed adaptively and quickly.

3 Wavelet Curves

This section describes how curves can be represented using a wavelet basis, concentrating on results relevant for this paper. For more details see [4, 17, 8, 15]. To explain the complete wavelet construction, a first step will be to create a new basis for the same space of curves defined by the B-splines, but consisting of two distinct types of basis functions.

3.1 Two-Part Basis

To understand the two-part basis, begin with the familiar cubic B-spline basis made up of translated copies of the B-spline blending shape $\phi(t)$. Let us denote the basis functions as

$$\phi_{L,j}(t) = \phi(t - j) \tag{11}$$

(the index j represents the translation of a specific basis from the canonical B-spline centered at zero, and L is the *level* or resolution of the basis). The space (or family) of curves spanned by all linear combinations of these basis functions will be denoted V_L (e.g., V_L contains all functions that are piecewise cubic, with simple knots at the integers).

Wavelets offer an L level hierarchical basis for the space V_L, but let us begin by building a two-part basis at level $L-1$ of the hierarchy. The two-part basis begins with the basis functions

$$\phi_{L-1,j}(t) = \phi(2^{-1}t - j) \tag{12}$$

These basis functions are twice as wide as the original B-spline basis functions, and hence the space they span contains piecewise cubic functions with knots at all *even* integers. This space will be referred to as V_{L-1}. According to the well known B-spline knot insertion algorithm [6, 9, 4] there is the following relationship

$$\phi_{L-1,j} \;=\; \sum_k h_{k-2j}\ \phi_{L,k} \tag{13}$$

where the sequence h is given in the appendix. This process is illustrated in Figure (2).

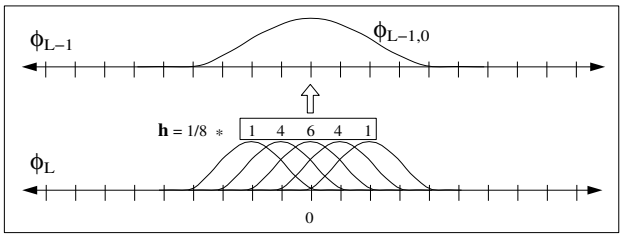

Figure 2: Five B-splines $\phi_{L,j}$ may be combined using the weights h to construct the double width B-spline $\phi_{L-1,0}$

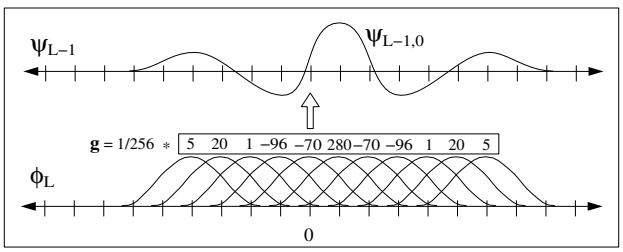

Figure 3: Eleven B-splines $\phi_{L,j}$ may be combined using the weights g to construct the wavelet function $\psi_{L-1,0}$

Clearly V_{L-1} is a proper subset of V_L and thus, it is not as rich as the space V_L. Therefore, more basis functions besides $\phi_{L-1,j}$ are needed in order to span the entire space V_L. In the wavelet methodology, this is accomplished by introducing into the basis, translated copies of a special wavelet shape ψ . Just as with the B-splines, the relationship between the wavelet basis functions and the model wavelet shape is notated $\psi_{L-1,j}(t) = \psi(2^{-1}t - j)$. These new basis functions are defined as:

$$\psi_{L-1,j} \;=\; \sum_k g_{k-2j}\ \phi_{L,k} \tag{14}$$

where the sequence g is given in the appendix. This process is illustrated in Figure (3). There is some degree of freedom in choosing the sequence g, as long the new basis functions "fill in" the missing space between V_L and V_{L-1}. Advantages to this particular choice of g include compactness and symmetry. With this construction, we now have two alternate bases for V_L, the B-spline basis

$$\{\phi_{L,j}\} \tag{15}$$

and the *two-part* basis

$$\{\phi_{L-1,j}, \psi_{L-1,j}\} \tag{16}$$

Just as the two-part basis functions can be expressed as a combination of the B-splines basis functions (Equations (13) and (14)), so too, the B-spline basis functions can be expressed as combinations of the two-part basis functions. This is given by

$$\begin{aligned} \phi_{L,j} \;&=\; \sum_k \tilde{h}_{j-2k}\ \phi_{L-1,k} \\ &+\; \sum_k \tilde{g}_{j-2k}\ \psi_{L-1,k} \end{aligned} \tag{17}$$

where the sequences $\tilde{h}$ and $\tilde{g}$ are given in the appendix. This process is illustrated in Figure (4). In the literature, there are many wavelet constructions, each with its own particular functions ϕ an ψ, and sequences h, g, $\tilde{h}$, and $\tilde{g}$ [5], The sequences and the particular wavelet construction described in this paper are derived

[5] In this particular construction h is related to $\tilde{g}$ and g is related to $\tilde{h}$ by reversing the sign of every other element. This is not in general true for all wavelet constructions.

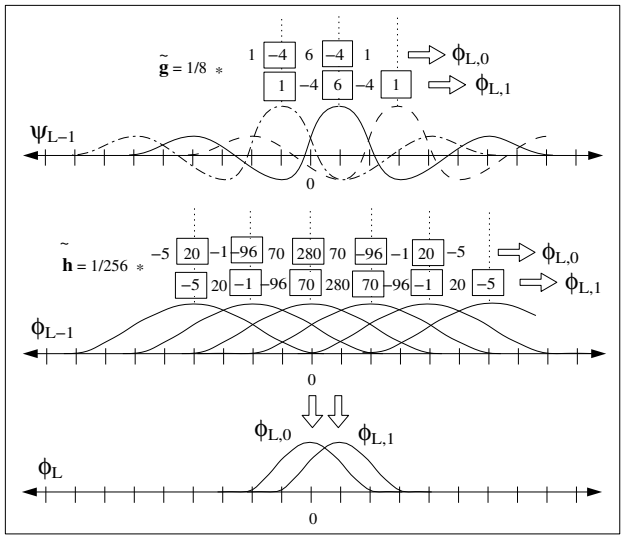

Figure 4: Two wavelets $\psi_{L-1,j}$ together with five double width B-splines $\phi_{L-1,j}$ may be combined using the weights $\tilde{g}$ and $\tilde{h}$ to construct the B-spline $\phi_{L,0}$. Three wavelets $\psi_{L-1,j}$ together with six double width B-splines $\phi_{L-1,j}$ may be combined using the weights $\tilde{g}$ and $\tilde{h}$ to construct the B-spline $\phi_{L,1}$.

in [5]. We have chosen this wavelet construction because the associated ϕ is a cubic B-spline, the function ψ is symmetric with compact support, and most importantly, all of the four sequences h, g, $\tilde{h}$, and $\tilde{g}$ are finite.

Suppose some function $x(t)$ in V_L has been expressed as a linear combination of the B-spline basis function

$$x(t) = \sum_j c_{\phi_{L,j}} \ \phi_{L,j} \tag{18}$$

where the c are scalar coefficients. The coefficients needed to express the function in the two-part basis can be found using the following formula

$$\begin{aligned} c_{\phi_{L-1,j}} &= \sum_k \tilde{h}_{k-2j} \ c_{\phi_{L,k}} \\ c_{\psi_{L-1,j}} &= \sum_k \tilde{g}_{k-2j} \ c_{\phi_{L,k}} \end{aligned} \tag{19}$$

and now

$$x(t) = \sum_j c_{\phi_{L-1,j}} \ \phi_{L-1,j} + c_{\psi_{L-1,j}} \ \psi_{L-1,j} \tag{20}$$

(This process is similar to the one illustrated in Figures (2-3) except that h and g and are interchanged with $\tilde{h}$ and $\tilde{g}$). Intuitively speaking, the $c_{\phi_{L-1,j}}$ encode the smooth (low frequency) information about the curve x , and the $c_{\psi_{L-1,j}}$ encode the fine detail (high frequency) information.

Alternatively if $x(t)$ has been represented with respect to the two-part basis, the representation with respect to the B-spline basis can be found with

$$\begin{aligned} c_{\phi_{L,j}} &= \sum_k h_{j-2k} \ c_{\phi_{L-1,k}} \\ &+ \sum_k g_{j-2k} \ c_{\psi_{L-1,k}} \end{aligned} \tag{21}$$

3.2 Bases on the Interval

In a classical wavelet construction, the index j goes from $-\infty \ldots \infty$, and V_L includes functions of unbounded support. In a geometric modeling context, only functions over some fixed finite interval need to be expressed, and it is important to only deal with a finite number of basis functions. One way to limit wavelets to an interval is to design special basis functions near the boundaries of the interval, much like B-splines with repeated knots at the ends [3]. The wavelet constructions in this case can become complicated and inelegant. The other possibility is to extend the function x to all of the real line and use a regular wavelet construction. There are a number of ways to extend x, for example, one can define the function to be zero outside of the interval, or one can extend the function by making periodic copies of x. Both of these methods are unacceptable since they will require the function to either be zero at the ends, periodic (a closed B-spline curve), or have discontinuities.

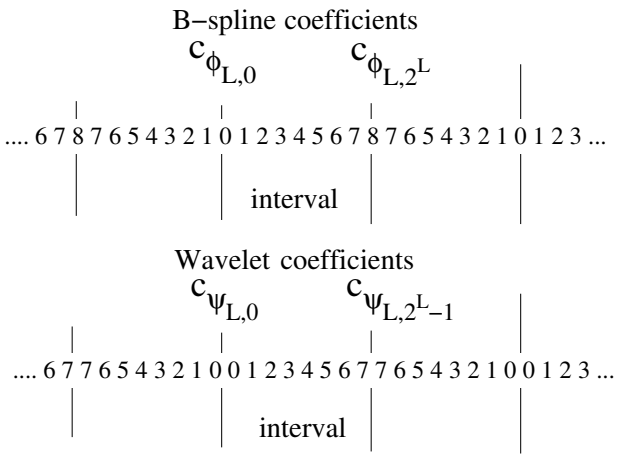

Figure 5: When a function on an interval is extended to the entire real line using mirror reflection, the corresponding B-spline and wavelet coefficients have mirrored patterns as well

Another solution, and the one selected here, is to use a mirror-reflection of the interval [8]. This technique, borrowed from signal processing, is simple to understand and requires the least special cases in the code. Because the technical details are non-trivial, and because the authors have not seen the details in any reference, this paper includes specific pseudo-code in the appendix.

Suppose some function $x(t)$ is only defined for t in $[0 \ldots 2^L]$. This function can be extended to the entire real line by placing a mirror at 0 and a mirror at 2^L, and reflecting the function back and forth. (This is the same as reflecting the functions around 0, and then making periodic copies of the singly reflected function). If this extended function is in V_L (is piecewise cubic with simple knots at the integers) [6] then there will be $2^L + 1$ unique B-spline coefficients $c_{\phi_{L,j}}$ with j in $\{0 \ldots 2^L\}$. The rest of the infinite number of coefficients will be defined by the mirrors. This pattern is shown in Figure (5). If the unique B-spline coefficients are stored in an array, **bs[**$0 \ldots 2^L$**]** any of the extended B-spline coefficients can be obtained with the indexing procedure **bs_index** given in the appendix.

After obtaining coefficients of the mirror reflected functions in the two-part basis with Equation (19), the result will have $2^{L-1}+1$ unique coefficients $c_{\phi_{L-1,j}}$ and 2^{L-1} unique coefficients $c_{\psi_{L-1,j}}$. The reflection pattern of the ψ coefficients is slightly different than that of the ϕ coefficients since each wavelet basis function is centered half way between the centers of two B-splines. The pattern of wavelet coefficients is shown in Figure (5).

[6] The only limitation of this method is that in order for the extended function to be in V_L, $\frac{dx}{dt}$ must be zero at the boundaries. This does not pose any serious limitations.

The unique ψ coefficients may be stored in an array **wave[**$0 \ldots 2^L - 1$**]**, and then any of the extended wavelet coefficients can be obtained with the indexing procedure **wave_index** given in the appendix.

Given the above indexing functions, the interval version of Equation (19) can be implemented with the procedure **coef_xform_up**. The inverse transformation described by Equation (21) may be implemented by the procedure **coef_xform_down** (see appendix).

The interval wavelet transformation must be applied to the basis functions as well as the coefficients. Equations (13) and (14) are implemented in the procedure **basis_xform_up**. And finally Equation (17) may be implemented with the procedure **basis_xform_down**.

3.3 Wavelet Basis

The reasoning that was used to construct the two-part basis can now be applied L times to construct a multilevel *wavelet basis*. Thus far, a two-part basis $\{\phi_{L-1,k1}, \psi_{L-1,j1}\}$ with $k1$ in $\{0 \ldots 2^{L-1}\}$,and $j1$ in $\{0 \ldots 2^{L-1} - 1\}$ has been discussed as an alternative for the B-spline basis $\{\phi_{L,k0}\}$ with $k0$ in $\{0 \ldots 2^L\}$.

Note that half of the basis functions in the two-part basis are themselves B-spline basis functions (only twice as wide). To continue the wavelet construction, keep the basis functions $\psi_{L-1,j1}$ and re-apply the reasoning of section 3.1 to replace the $\phi_{L-1,k1}$ with $\{\phi_{L-2,k2}, \psi_{L-2,j2}\}$ ($k2$ in $\{0 \ldots 2^{L-2}\}$ and $j2$ in $\{0 \ldots 2^{L-2} - 1\}$). This results in the new basis $\{\phi_{L-2,k2}, \psi_{L-2,j2}, \psi_{L-1,j1}\}$ where $\phi_{L-2,j}(t) = \phi(2^{-2}t - j)$, and $\psi_{L-2,j}(t) = \psi(2^{-2}t - j)$.

Each time this reasoning is applied, the number of B-spline functions in the hierarchical basis is cut in half, and the new basis functions become twice as wide. After L applications, the wavelet basis

$$\{\phi_{0,0}, \phi_{0,1}, \psi_{i,j}\} \tag{22}$$

is obtained, with i in $\{0 \ldots L-1\}$ and j in $\{0 \ldots 2^i - 1\}$, where

$$\begin{aligned} \phi_{i,j}(t) &= \phi(2^{(i-L)}t - j) \\ \psi_{i,j}(t) &= \psi(2^{(i-L)}t - j) \end{aligned} \tag{23}$$

This basis is made up of two wide B-splines, and translates (index j) and scales (index i) of the wavelet shape (see Figure (6)).

The coefficients representing some function in the B-spline basis can be transformed to the wavelet basis using the procedure **coef_pyrm_up**, that makes L calls to **coef_xform_up** each time with an input vector of $1/2$ the length. (Note since this transforms an n-vector to an n-vector, it can be implemented with proper indexing using linear storage).

```
coef_pyrm_up( bs_in[][], bs_out[][], w_out[][], L )
    bs_temp[L][] = bs_in[L][] ;
    for( i = L; i ≥ 1; i-- )
        coef_xform_up( bs_temp[i][], bs_temp[i-1][],
                       w_out[i-1][], i ) ;
    bs_out[0][] = bs_temp[0][] ;
```

The inverse transformation is

```
coef_pyrm_down( bs_in[][], w_in[][] , bs_out[][], L )
    bs_temp[0][] = bs_in[0][] ;
    for( i = 1; i ≤ L; i++ )
        coef_xform_down( bs_temp[i-1][], w_in[i-1][],
                         bs_temp[i][], i ) ;
    bs_out[L][] = bs_temp[L][] ;
```

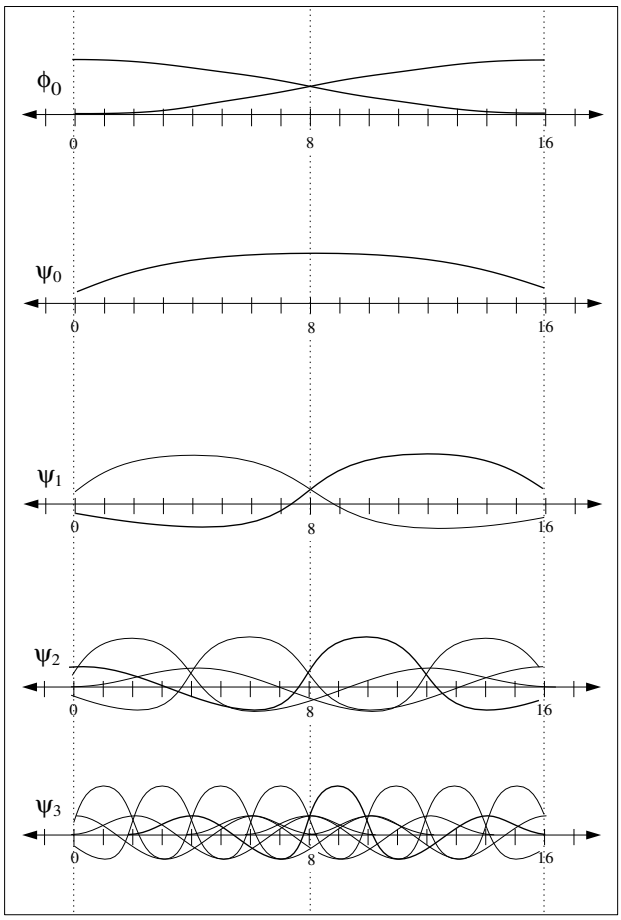

Figure 6: The wavelet basis is a hierarchical basis with L levels (in this figure $L = 4$). On the coarsest level 0, there are two wide B-splines $\phi_{0,0}$ and $\phi_{0,1}$, and one wide wavelet function $\psi_{0,0}$. On each subsequent level there are twice as many wavelet functions, that are twice as thin.

Finally, the basis transformations are

```
basis_pyrm_up( bs_in[][], bs_out[][], w_out[][], L )
    bs_temp[L][] = bs_in[L][] ;
    for( i = L; i ≥ 1; i-- )
        basis_xform_up( bs_temp[i][], bs_temp[i-1][],
                        w_out[i-1][], i ) ;
    bs_out[0][] = bs_temp[0][] ;
```

and

```
basis_pyrm_down( bs_in[][], w_in[][] , bs_out[][], L )
    bs_temp[0][] = bs_in[0][] ;
    for( i = 1; i ≤ L; i++ )
        basis_xform_down( bs_temp[i-1][], w_in[i-1][],
                          bs_temp[i][], i ) ;
    bs_out[L][] = bs_temp[L][] ;
```

The running time of these pyramid procedures is governed by the geometric series $(n + \frac{n}{2} + \frac{n}{4} + \ldots + 1) = O(n)$, and hence they run in linear time. Each one of these four procedures transforms one n-vector, to another, and thus can be represented as a

matrix. If W is the matrix of the linear transformation performed by the procedure **coef_pyrm_up**, then W^{-1} is the matrix of **coef_pyrm_down**, W^{-T} is the matrix of **basis_pyrm_up** and W^T is the matrix of **basis_pyrm_down**.

The wavelet basis is an alternative basis for V_L, but unlike the B-spline basis, it is an L level hierarchical basis. At level 0 there are two broad B-splines, and one broad wavelet. These three basis functions give the coarse description of the function. At each subsequent level going from level 0 to $L-1$, the basis includes twice as many wavelets, and these wavelets are twice as narrow as the ones on the previous level. Each level successively adds more degrees of detail to the function. If the function is sufficiently smooth in some region, then very few non-zero wavelet coefficients will be required in that region[7].

3.4 Scaling

One final issue is the scaling ratio between the basis functions. Traditionally [17, 19] the wavelet functions are defined with the following scaling:

$$\begin{aligned} \phi_{i,j}(t) &= 2^{(i-L)/2}\ \phi(2^{(i-L)}t - j) \\ \psi_{i,j}(t) &= 2^{(i-L)/2}\ \psi(2^{(i-L)}t - j) \end{aligned} \tag{24}$$

This means that at each level up, the basis functions become twice as wide, and are scaled $\frac{1}{\sqrt{2}}$ times as tall. While in many contexts this normalizing may be desirable, for optimization purposes it is counter productive. For the optimization procedure to be well conditioned [14, 7] it is advantageous to emphasize the coarser levels and hence use the scaling defined by

$$\begin{aligned} \phi_{i,j}(t) &= 2^{L-i}\ \phi(2^{(i-L)}t - j) \\ \psi_{i,j}(t) &= 2^{L-i}\ \psi(2^{(i-L)}t - j) \end{aligned} \tag{25}$$

where the wider functions are also taller. In the pyramid code, this is achieved by multiplying all of the h and g entries by 2, and all of the $\tilde{h}$ and $\tilde{g}$ by $1/2$.

4 Wavelet Surfaces

Basis functions of two variables, $B_{j,k}(s,t)$, are needed to represent a surface. One choice for basis functions is the tensor product B-spline basis $\{\phi_{L,j}(s)\phi_{L,k}(t)\}$. Another choice is the L level tensor wavelet basis.

For curves, the wavelet construction began with the two-part basis. For surfaces, the tensor wavelet construction begins with the *four-part basis*. To construct the four-part basis, one starts with $\phi_{L-1,j}(s)\phi_{L-1,k}(t)$ (tensor product B-spline functions that are twice as wide) and completes the basis by adding in the basis functions $\psi_{L-1,j}(s)\psi_{L-1,k}(t)$ and $\psi_{L-1,j}(s)\phi_{L-1,k}(t)$ and $\phi_{L-1,j}(s)\psi_{L-1,k}(t)$. Note that with curves, only one type special function $\psi(t)$ was needed to complete the basis, while with surfaces three types of functions need to be added. With this choice, transformations between the tensor B-spline and the tensor four-part bases can be accomplished with the procedures
coef_tensor_xform_up,
coef_tensor_xform_down,
basis_tensor_xform_up, and
basis_tensor_xform_down.

The procedure **coef_tensor_xform_up** works as follows: The coefficients $c_{\phi_{L,j},\phi_{L,k}}$ are placed in a "matrix" according to their indices j and k. Each row of this matrix is transformed using **coef_xform_up** resulting in a new matrix of coefficients. Now each column of this resulting matrix is submitted to **coef_xform_up**, resulting in the four-part coefficients $c_{\phi_{L-1,j},\phi_{L-1,k}}$ and $c_{\phi_{L-1,j},\psi_{L-1,k}}$ and $c_{\psi_{L-1,j},\phi_{L-1,k}}$ and $c_{\psi_{L-1,j},\psi_{L-1,k}}$.

Just as with curves, reapplication of this reasoning to the $\phi_{L-1,j}(s)\phi_{L-1,k}(t)$ functions results in another alternative basis, and after L applications, the tensor wavelet basis is obtained

$$\begin{matrix} \phi_{0,j}(s)\phi_{0,k}(t) & \phi_{i,j}(s)\psi_{i,k}(t) \\ \psi_{i,j}(s)\phi_{i,k}(t) & \psi_{i,j}(s)\psi_{i,k}(t) \end{matrix} \tag{26}$$

(with i in $\{0 \ldots L-1\}$).

Again, as with curves, one can transform between the tensor B-spline and the tensor wavelet bases bases using the procedures
coef_tensor_pyrm_up,
coef_tensor_pyrm_down,
basis_tensor_pyrm_up, and
basis_tensor_pyrm_down.
where the procedure **coef_tensor_pyrm_up** makes L calls to **coef_tensor_xform_up**, each time with an input "matrix" of $1/4$ the size.

5 Hierarchical Conditioning

In the B-spline basis, the optimization procedure resulted in the linear system given by Equation (7). In the wavelet basis, a different linear system results which is given by

$$\begin{vmatrix} \tilde{H} & \tilde{A}^T \\ \tilde{A} & 0 \end{vmatrix} \begin{vmatrix} \tilde{c} \\ \lambda \end{vmatrix} = \begin{vmatrix} 0 \\ b \end{vmatrix} \tag{27}$$

where the tildes signify that the variables are wavelet coefficients, $\tilde{c} = Wc$, and the Hessian and constraint matrix are expressed with respect to the wavelet basis. To see the relationship with the B-spline system, the new system can also be written down as

$$\begin{vmatrix} W^{-T}HW^{-1} & W^{-T}A^T \\ AW^{-1} & 0 \end{vmatrix} \begin{vmatrix} \tilde{c} \\ \lambda \end{vmatrix} = \begin{vmatrix} 0 \\ b \end{vmatrix} \tag{28}$$

Although Equation (7) and Equation (27/28) imply each other, they are two distinct linear systems of equations. Because the wavelet system (27/28) is hierarchical it will not suffer from the poor conditioning of the B-spline system of Equation (7) (see Figure (7)). For a rigorous discussion of the relevant theory see [7].

There is now a choice to make. In an iterative conjugate gradient solver, the common operation is multiplication of a vector times the wavelet matrix given in Equations (27/28). There are two ways to implement this.

One approach, the *explicit* approach, is to compute and store the wavelet Hessian matrix $\tilde{H}$ and the wavelet constraint matrix $\tilde{A}$ (Equation (27)). These can be computed directly from a closed form (piecewise polynomial) representation of the wavelet functions $\psi_{i,j}$. For example, for wavelet curves, the Hessian terms of the thin plate energy are $\int dt\ \ddot{\psi}_{i,j}(t)\ddot{\psi}_{k,l}(t)$. Unfortunately, these matrices are not as sparse as the B-spline Hessian and constraint matrices for two reasons. Firstly, the wavelets are wider than B-spline functions. On a single level, i, there are 13 wavelet curve functions that overlap with a chosen wavelet, (instead of the 7 B-splines that overlap a single B-spline). For tensor product surfaces, there are over 300 wavelet functions that overlap with a chosen wavelet (instead of the 49 B-splines). Secondly, since wavelets are hierarchical, a single wavelet will overlap with wavelets on all L levels. As a result there are $O(n \lg n)$ overlapping wavelet basis functions, and thus the wavelet Hessian matrix is only $O(n \lg n)$ sparse.

Alternatively, there is the *implicit* approach [23] which only computes and stores the entries of the B-spline matrices H and A (Equation (28)). In this approach, the multiplication of a vector

[7] In this case, non-zero can be defined to be having an absolute value greater than some epsilon without incurring significant error in the representation.

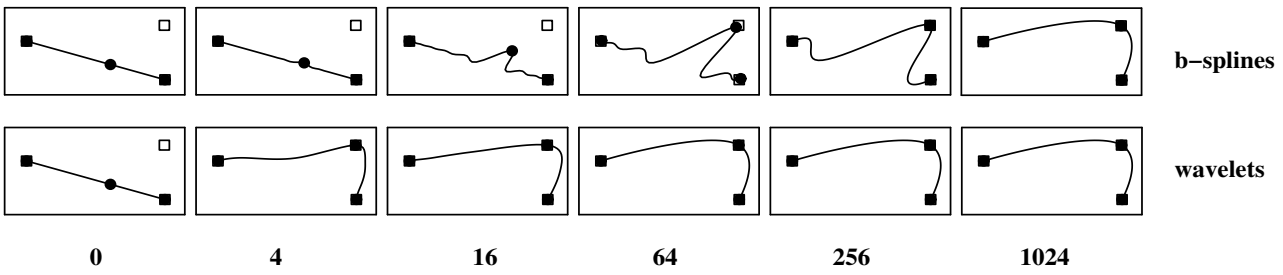

Figure 7: Solutions found by B-spline and wavelet methods after various numbers (0-1024) of iterations. (65 variables, 3 constraints). This illustrates the ill conditioning of the B-spline optimization problem.

v by a wavelet matrix $\tilde{H}$ or $\tilde{A}$ is computed as $W^{-T}HW^{-1}v$ or $AW^{-1}v$, where the matrix-vector multiplies involving W^{-1} are implemented using the **coef_pyrm_down** (tensor version for surfaces), and the multiplies involving W^{-T} are implemented using the **basis_pyrm_up** (tensor). The advantage of this approach is that the whole multiply remains $O(n)$ in both time and space, since the **pyrm** procedures run in linear time, and the matrices H and A are $O(n)$ sparse. There is, of course, some constant factor of overhead introduced by executing the **pyrm** procedures.

Even though one of the methods explicitly uses wavelet terms while the other uses B-spline terms, these two methods are mathematically equivalent, and so both will have the same condition properties. On the other hand, they may have different space requirements, and require different amounts of time to compute a matrix-vector multiply. We have implemented both approaches and discuss the results in Section 8.

6 Adaptive Oracle

By limiting the possible surfaces to only those that can be expressed as a linear combination of a fixed set of basis functions (perhaps uniform B-splines of a given size), one obtains an approximation of the true optimal surface. As more basis functions are added, perhaps by using smaller B-splines, the space of possible solutions becomes richer and a closer approximation to the true optimal surface can be made. Unfortunately, as the space becomes richer, the number of unknown coefficients increases, and thus the amount of computation required per iteration grows. A priori it may be difficult to know how many basis functions are needed to allow an approximation with a sufficiently small error. It is therefore desirable to have a solution method that adaptively chooses the appropriate basis functions given some user defined error bound. This approach was applied using hierarchical B-splines in [27]. When refinement was necessary, "thinner" B-splines basis functions were added, and the redundant original "wider" B-splines were removed. With wavelets, all that must be done is to add in new "thinner" wavelets wherever refinement is deemed necessary. Since the wavelets coefficients correspond directly to local detail, all previously computed coefficients are still valid.

The decision process of what particular wavelets to add and remove is governed by an **oracle** procedure which is called after every fixed number of iterations. The oracle must decide what level of detail is required in each region of the curve or surface. In [27] two criteria for refinement were suggested. Refinement to better satisfy a constraint, and refinement to obtain an answer with lower energy. Using B-splines, the first criterion is easy to implement because it is simple to detect that a constraint is not being satisfied, but there is no easy way to implement the second criterion because there is no direct way of knowing how much local detail is being used in the solution. In a wavelet basis the second criterion is also easy because the magnitude of a wavelet coefficient itself specifies the degree of detail in the answer thus far.

When some region of the solution does not need fine detail, the corresponding wavelet coefficients are near zero, and so the first thing the **oracle** does is to deactivate the wavelet basis functions whose corresponding coefficients are below some small threshold. The **oracle** then activates new wavelet basis functions where it feels more detail may be needed. There are two criteria used. If a constraint is not being met, then the oracle adds in finer wavelet functions in the region that is closest in parameter space to the unmet constraint. Even if all the constraints are being met, it is possible that more basis functions would allow the freedom to find a solution with lower energy. This is accomplished by looking for the current basis functions with coefficients above some maximum threshold, and activating finer wavelet functions in that parameter region. This is based on the reasoning that if the current solution requires detail in some region, it may be improved by adding even finer detail in that region.

There can be some tendency for the **oracle** to get into cycles; at one iteration it may activate a new basis function, and at some later iteration it may notice that the basis function is not being used and deactivate it. To avoid the immediate reactivation of basis functions, a basis function is marked as being **dormant** when it is removed from consideration. Of course, it is possible that later on the solution may really need this basis function, and so periodically there is a **revival** phase, where the **dormant** marks are removed.

7 Implementation

The ideas outlined in the previous sections have been used to build an interactive variational modeling system (see Figure (1)). Some implementation details are described here.

A user of the system is first presented with a default curve or surface. Constraints can then be introduced by clicking on the curve or surface with the mouse. The location of the mouse click defines a parametric position t (and s) on the curve (or surface). The user can then drag this point to a new location to define an interpolation constraint. Tangent constraints at a point can also be defined by orienting "arrow" icons at the point. Once the constraint is set, the solver is called to compute the minimum energy solution that satisfies the constraints placed so far. The system uses thin plate energy to measure the smoothness of the answer. Resulting curves and surfaces are displayed using SGI GL **nurbscurve** and **nurbssurface** calls [8].

7.1 Oracle

The solver sets up the appropriate linear system (Equation (27/28)). Initially the solver assumes that very few degrees of

[8] One GL call to **nurbssurface** can be more expensive than a complete iteration.

freedom are needed, thus only a few variables $\tilde{c}$ are active, corresponding to the basis functions on wavelet level 0 (the coarsest level). The matrices are represented sparsely, so the solver only needs storage for the non-zero entries, and the cost of a matrix-vector multiply scales with the number of non-zero entries. This linear system is then solved by using conjugate gradient iterations [20]. After every k iterations (25 is the default), the **`oracle`** is called. As explained in Section 6, the oracle decides which new wavelet basis functions and their corresponding variables $\tilde{c}$ should be activated, and which ones can be deactivated.

At this point the matrices must be updated. If the explicit approach is being used (Equation (27)), then the bookkeeping is very simple. When activating a new wavelet basis function all that must be done is to add in a row and column to $\tilde{H}$, and a column to $\tilde{A}$. These matrix terms are computed on the fly symbolically. When deactivating a wavelet basis function, all that is done is to delete a column (and row) from the matrices. If there is sufficient memory on the computer, then the discarded values are cached; if the **`oracle`** later reactivates those basis function, then the solver does not need to symbolically recompute those matrix entries.

When the implicit approach is being used (Equation (28)), the bookkeeping becomes a bit more complicated since, when very few wavelet functions are active, it is not necessary or efficient to compute the entire B-spline matrices H and A, nor is it necessary to pay the cost of the complete **`pyrm`** procedures. Instead the following is done. Given the active set of wavelet functions, the solver finds the minimal set of B-splines, $\phi_{i,j}$, that spans the space of the active wavelet set. A *sufficient* B-spline set is found by replacing each wavelet function, $\psi_{i,j}$, with enough B-spline functions from one level lower $\phi_{i+1,j}$, that can construct it (Equation (14)). This sufficient set is then made smaller by removing the B-splines that can be constructed using other B-splines from the sufficient set that are on lower levels. (Note that in this minimal set, B-spline functions from any level i are allowed, thus in regions where detail is not needed, this set will contain wider B-splines.) The Hessian and constraint terms are then computed with respect to these B-spline functions. Finally, the **`pyrm`** procedures are altered so that they perform transformations between the active wavelet set, and the minimal B-spline set. (The **`for`** loops in the **`pyrm`** procedures of the appendix only iterate over the necessary **`j`** and **`k`**). The running time of these altered **`pyrm`** procedures is linear in the number of active wavelets [9].

This adaptive approach is possible with the wavelets described in this paper, but was not possible with the hierarchical construction in [23]. This is because an adaptive method must be able to compute new matrix terms on the fly, and therefore one needs an explicit expression for the hierarchical basis functions. Fortunately, the $\phi_{i,j}$ and $\psi_{i,j}$ functions described in this paper have a piecewise polynomial form.

The conjugate gradient iterations are continued until the residual is smaller than some tolerance.

7.2 Delta Constraints

When the solution is completed, the result provides information for not only the curve or surface satisfying the specific value of the new constraint, but for curves or surfaces with respect to any value of this constraint. Once the linear system (Equation (27/28)) with the newest constraint has been solved, the solver stores the delta vector

$$\frac{\Delta \tilde{c}}{\Delta b_m} \tag{29}$$

where m is the index of the newest constraint, and b_m is the constraint value (i.e., the position or tangent specified by the user).

[9] Code to do this is available from the authors upon request.

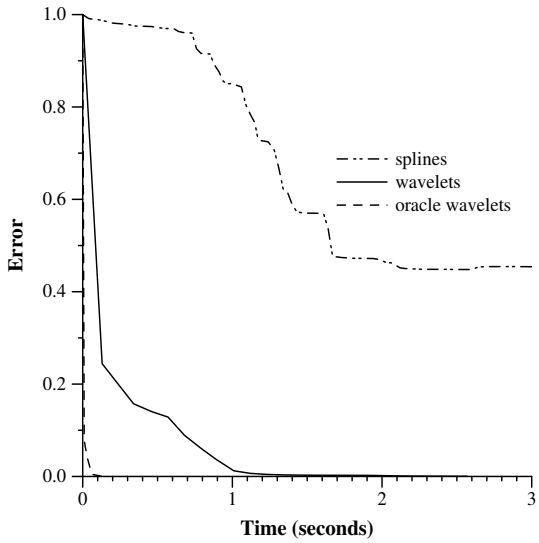

Figure 8: Error per time. Curve with 65 control points, 3 constraints.

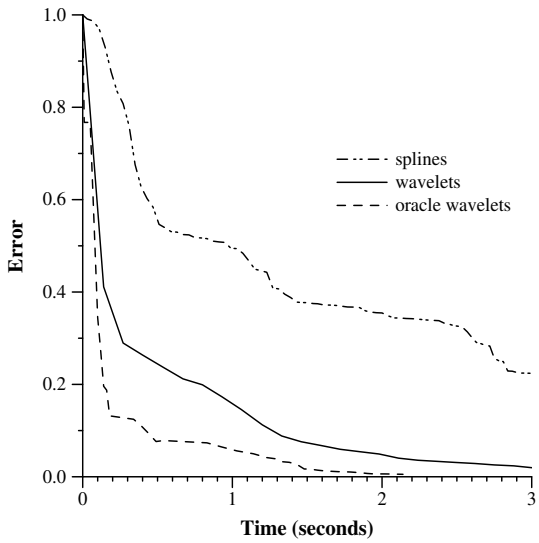

Figure 9: Error per time. Curve with 65 control points, 7 constraints.

This vector stores the change of the coefficient vector due to a unit change in the new constraint Δb_m, essentially a column of the inverse matrix. The user is now free to interactively move the target location of the constraint without having to resolve the system since, as long as the parameters s, and t of the constraints do not change, the matrix of the system, and thus its inverse, do not change. However, as soon as a new constraint is added (or a change to the parameters s and t is made) there is fresh linear system that must be solved, and all of the delta vectors are invalidated. The ability to interactively change the value of a constraint is indicated to the user by coloring the constraint icon.

8 Results

A series of experiments were conducted to examine the performance of the wavelet based system compared to a B-spline basis. In the curve experiments, the number of levels of the hierarchy, L, was fixed to 6, and thus there were 65 B-spline, or wavelet

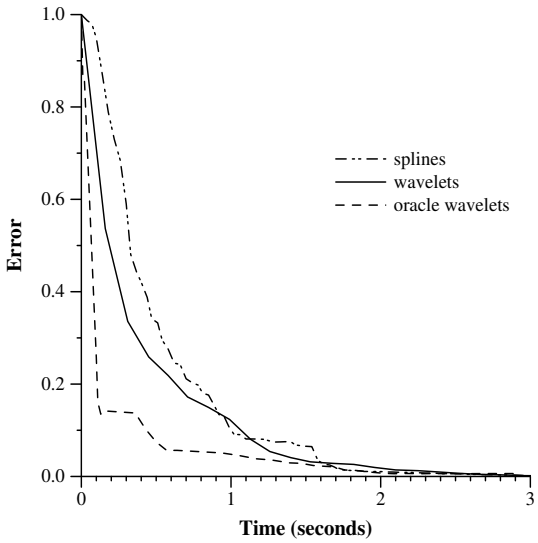

Figure 10: Error per time. Curve with 65 control points, 13 constraints.

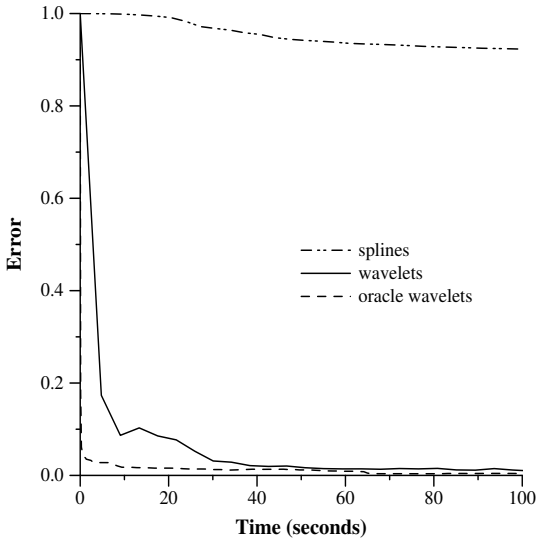

Figure 11: Error per time. Surface with 1089 control points, 5 constraints.

functions in the complete basis for $x(t)$ (and another 65 basis functions for y as well as z). The optimization process was then run on problems with various amounts of constraints. The results of these tests are shown in Figures (8-10). These graphs show the convergence behavior of three different methods, solving with the complete B-spline basis, solving with the complete wavelet basis, and solving with an adaptive wavelet basis that uses an oracle. (The wavelet results shown here are using the *implicit* implementation). If $c^{(m)}$ is the computed solution expressed as B-spline coefficients at time m, and c^* is the correct solution of the complete linear system [10] (ie. the complete system with 2^L+1 variables, and no adaptive oracle being used) then the error at time m is defined as

$$\frac{\sum_j \mid c_j^* - c_j^{(m)} \mid}{\sum_j \mid c_j^* - c_j^{(0)} \mid} \tag{30}$$

[10] computed numerically to high accuracy

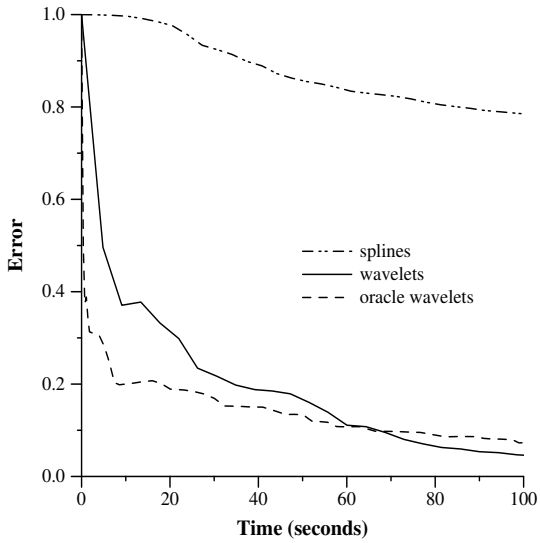

Figure 12: Error per time. Surface with 1089 control points, 11 constraints.

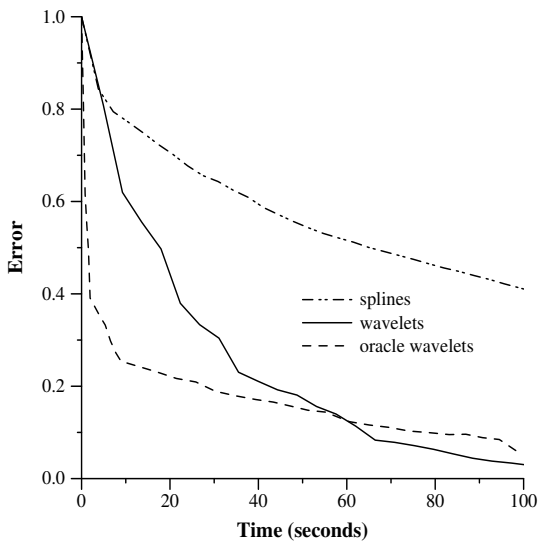

Figure 13: Error per time. Surface with 1089 control points, 23 constraints.

To obtain the starting condition $c^{(0)}$, two constraints were initialized at the ends of the curve, and the minimal thin plate solution (which in this case is a straight line) was computed. (For surfaces, the four corners were constrained.) All times were taken from runs on an SGI R4000 reality engine.

When there are a small number of constraints, and the solution is very under constrained, the B-spline method is very poorly conditioned, and converges quite slowly while the wavelet method converges dramatically faster (this is illustrated in Figure (7)). Since the optimization problem is very underconstrained, the oracle decides that it needs only a very small active set of wavelets and so the adaptive method converges even faster. As the number of constraints is increased, the solution becomes more tightly constrained, and the condition of the B-spline system improves. (Just by satisfying the constraints, the B-spline solution is very close to minimal energy). Meanwhile the oracle requires a larger active set of wavelets. Eventually, when enough constraints are present, the wavelet methods no longer offer an advantage over

B-splines. While it still requires fewer iterations to converge, it must pay more per iteration.

How much is this extra cost? Experiments with both the explicit and implicit wavelet methods showed that for curves with a complete basis (no oracle), and $L = 6$, one iteration of the explicit or the implicit wavelet method was slower than one B-spline iteration by about a factor of 5. Thus, for problems of this size, the explicit and the implicit methods cost about the same. For the implicit method, this factor of 5 is the extra constant factor incurred by using the **pyrm** procedures and is independent of L. For the explicit method, this factor of 5 was a result of the denser wavelet matrices $\tilde{H}$ and $\tilde{A}$, which are only $O(n \lg n)$ sparse, and so this factor will increase as L increases.

As expected, for surfaces everything becomes more expensive (because of the denser surface matrices), except for the **pyrm** procedures. And so for surfaces, a single iteration of the implicit wavelet method is only slower than a single iteration of the B-spline method by about a factor of 1.5, regardless of L. Meanwhile one iteration of the explicit wavelet method for $L = 5$, where the complete basis has 1089 functions (33 in s, times 33 in t), is slower than one B-spline iteration by about a factor of 10.

Since the constant factor of overhead for the implicit method is so small, it outperforms the B-spline method, even when many constraints are added to the problem. In the surface experiments, L was set to 5. (There are 1089 basis functions for the x coordinates, and another 1089 for y as well as z). The experimental results are shown in Figures (11-13). As the solution progresses, the complete wavelet basis outperforms the adaptive method, because the oracle is using a smaller set of basis functions, and its solution has slightly more energy.

9 Conclusion

The ability to directly manipulate curves and surfaces provides an intuitive interface to the user for design purposes. Direct manipulation of a surface, however, leads to an underconstrained problem since there are, in general, many possible surfaces through a given point or set of points. Finding the "best" solution requires solving a variational problem.

This paper has presented a methodology to efficiently solve such problems based on a hierarchical representation of the curves and surfaces. In particular, a wavelet basis is used to accelerate the computation of optimal curves and surfaces based on a thin plate functional. Implementation and a system developed with these ideas are presented and experimental results are discussed that demonstrate the advantages over a pure B-spline based approach.

Another application for the adaptive oracle is data reduction. The size of the active set of wavelet coefficients is often much smaller than the number of B-spline control points in the computed solution. For example, in the curve experiments, the size of the final active sets for 3, 7, and 13 constraints respectively were 8, 21, and 29, as compared to the 65 B-spline control points. Similarly, in the surface experiments, the size of the final active sets for 5, 11, and 23 constraints respectively were 82, 150, and 350, as compared to the 1089 B-spline control points.

Future work will be required to explore the use of higher order functionals like those given in [18, 22]. Because the optimization problems resulting from those functionals are non-linear, they are much more computationally expensive, and it is even more important to find efficient methods. It is also important to study optimization modeling methods where constraint changes only have local effects. And it would be desirable to extend the multiresolution framework beyond tensor product bases in order to model surfaces with arbitrary topology [13]. Presently, the only way to model an arbitrary structure is to piece together patches, and constrain the boundaries to have the necessary continuity. In addition, improvements to the *oracle* function that determines the appropriate local resolution may lead to more efficient results.

Acknowledgements

We are grateful to James H. Shaw for developing the graphical interface to the optimization program.

References

[1] BARTELS, R., BEATTY, J., AND BARSKY, B. *An Introduction to Splines for Use in Computer Graphics and Modeling*. Morgan Kaufmann, 1987.

[2] CELNIKER, G., AND GOSSARD, D. Deformable curve and surface finite-elements for free-from shape design. *Computer Graphics 25*, 4 (July 1991), 257–266.

[3] CHUI, C., AND QUAK, E. Wavelets on a bounded interval. *Numerical Methods of Approximation Theory 9* (1992), 53–75.

[4] CHUI, C. K. *An Introduction to Wavelets*, vol. 1 of *Wavelet Analysis and its Applications*. Academic Press Inc., 1992.

[5] COHEN, A., DAUBECHIES, I., AND FEAUVEAU, J. C. Biorthogonal bases of compactly supported wavelets. *Communication on Pure and Applied Mathematics 45* (1992), 485–560.

[6] COHEN, E., LYCHE, T., AND RIESENFELD, R. Discrete b-splines and subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing 14*, 2 (October 1980), 87–111.

[7] DAHMEN, W., AND KUNOTH, A. Multilevel preconditioning. *Numerische Mathematik 63* (1992), 315–344.

[8] DAUBECHIES, I. *Ten Lectures on Wavelets*, vol. 61 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, 1992.

[9] FORSEY, D., AND BARTELS, R. Hierarchical b-spline refinement. *Computer Graphics 22*, 4 (August 1988), 205–212.

[10] FORSEY, D., AND WENG, L. Mulit-resolution surface approximation for animation. In *Graphics Interface* (1993).

[11] GILL, P., MURRAY, W., AND WRIGHT, M. *Practical Optimization*. Academic Press, 1981.

[12] GORTLER, S., SCHRÖDER, P., COHEN, M., AND HANRAHAN, P. Wavelet radiosity. In *Computer Graphics, Annual Conference Series, 1993* (1993), Siggraph, pp. 221–230.

[13] HALSTEAD, M., KASS, M., AND DEROSE, T. Efficient, fair interpolation using catmull-clark surfaces. In *Computer Graphics, Annual Conference Series, 1993* (1993), Siggraph, pp. 35–43.

[14] JAFFARD, S., AND LAURENÇOT, P. Orthonormal wavelets, analysis of operators, and applications to numerical analysis. In *Wavelets: A Tutorial in Theory and Applications*, C. K. Chui, Ed. Academic Press, 1992, pp. 543–602.

[15] JAWERTH, B., AND SWELDENS, W. An overview of wavelet based multiresolution analysis, (submitted to siam review).

[16] LOUNSBERY, M., MANN, S., AND DEROSE, T. Parametric surface interpolation. *IEEE CG&A 12*, 5 (September 1992), 45–52.

[17] MALLAT, S. G. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE PAMI 11* (July 1989), 674–693.

[18] MORETON, H., AND SEQUIN, C. Functional optimization for fair surface design. *Computer Graphics 26*, 4 (July 1992), 167–176.

[19] PENTLAND, A. Fast solutions to physical equilibrium and interpolation problems. *The Visual Computer 8*, 5 (1992), 303–314.

[20] PRESS, W., TEUKOLSKI, S., VETTERLING, W., AND FLANNERY, B. *Numerical Recipies in C, The Art of Scientific Computing*, 1 ed. Cambridge University Press, 1988.

[21] QIAN, S., AND WEISS, J. Wavelets and the numercal solution of partial differential equations. *Journal of Computational Physics 106*, 1 (May 1993), 155–175.

[22] RANDO, T., AND ROULIER, J. Designing faired parametric surfaces. *Computer Aided Design 23*, 7 (September 1991), 492–497.

[23] SZELISKI, R. Fast surface interpolation using hierarchical basis functions. *IEEE PAMI 12*, 6 (June 1990), 513–439.

[24] TERZOPOULOS, D. Image analysis using multigrid relaxation methods. *IEEE PAMI 8*, 2 (March 1986), 129–139.

[25] TERZOPOULOS, D. Regularization of inverse visual problems involving discontinuities. *IEEE PAMI 8*, 4 (July 1986), 413–424.

[26] TERZOPOULOS, D. The computation of visible-surface representations. *IEEE PAMI 10*, 4 (July 1988), 417–438.

[27] WELCH, W., AND WITKIN, A. Variational surface modling. *Computer Graphics 26*, 2 (July 1992), 157–166.

[28] WILLIAMS, L., AND LITWINOWICZ, P. Personal commuincation.

A Some Technical Details

This section provides pseudo code for the procedures discussed in Section 3.2, as well as other technical details.

The **bs_index** procedure returns an index into the array of B-spline coefficients **bs[**$0 \ldots 2^L$**]** using mirror reflection The procedure first reflects around 0, and then periodizes.

```
int bs_index(L, j)
    range = 2^L ;
    j = absolute_value(j) ;
    j = j % (2 * range) ;   /* "%" == "mod" */
    if (j ≤ range)
        return(j) ;
    else
        return(2 * range - j);
```

This **wave_index** procedure indexes into the array of mirrored wavelet coefficients **wave[**$0 \ldots 2^L - 1$**]**.

```
int wave_index(L, j)
    range = 2^L ;
    j = j + .5 ;
    j = absolute_value(j) ;
    j = j - .5 ;
    j = ( j % (2 * range)) ;
    if (j ≤ range - 1)
        return(j) ;
    else
        return(2 * range - 1 - j) ;
```

Coefficient transformations are implemented with the procedures **coef_xform_up** and **coef_xform_down**. The first procedure implements Equation (19). (The sequences h,g,$\tilde{h}$, and $\tilde{g}$ are zero outside of their defined ranges).

$h[-2..2] = \{\frac{1}{8}, \frac{4}{8}, \frac{6}{8}, \frac{4}{8}, \frac{1}{8}\}$;
$g[-4..6] = \{\frac{5}{256}, \frac{20}{256}, \frac{1}{256}, \frac{-96}{256}, \frac{-70}{256}, \frac{280}{256}, \frac{-70}{256}, \frac{-96}{256}, \frac{1}{256}, \frac{20}{256}, \frac{5}{256}\}$;
$\tilde{h}[-5..5] = \{\frac{-5}{256}, \frac{20}{256}, \frac{-1}{256}, \frac{-96}{256}, \frac{70}{256}, \frac{280}{256}, \frac{70}{256}, \frac{-96}{256}, \frac{-1}{256}, \frac{20}{256}, \frac{-5}{256}\}$;
$\tilde{g}[-1..3] = \{\frac{1}{8}, \frac{-4}{8}, \frac{6}{8}, \frac{-4}{8}, \frac{1}{8}\}$;

```
coef_xform_up( bs_in[], bs_out[], w_out[], L)
    range = 2^(L-1) ;
    bs_out = wave_out = 0 ; /* zero vectors */
    for (j = 0 ; j < range + 1 ; j++)
        wdx = wave_index(L - 1, j) ;
        bdx = bs_index(L - 1, j) ;
        for (k = (2j - 5); k < (2j + 6) ; k++)
            idx = bs_index(L, k) ;
            bs_out[bdx]   += h̃[k - 2j] * bs_in[idx] ;
            wave_out[wdx] += g̃[k - 2j] * bs_in[idx] ;
```

The inverse transformation which uses Equation (21) is implemented in the following procedure

```
coef_xform_down( bs_in[], wave_in[], bs_out[], L )
    range = 2^L ;
    bs_out = 0 ; /* zero vector */
    for(j = 0; j < range + 1; j++)
        odx = bs_index(L, j) ;
        for(k = (j-6)/2; k <= (j+4)/2; k++)
            bdx = bs_index(L - 1, k) ;
            wdx = wave_index(L - 1, k) ;
            bs_out[odx] += h[j - 2k] * bs_in[bdx]
                         + g[j - 2k] * wave_in[wdx] ;
```

The following procedure transforms the basis functions themselves (see Equations 13 and 14). Because the basis functions centered exactly on the boundaries do not have a nearby mirrored copy, they must be scaled slightly differently.

```
basis_xform_up( bs_in[], bs_out[], w_out[], L )
    range = 2^(L-1) ;
    downRange = 2^L ;
    bs_in[0] *= 2.0;    bs_in[downRange] *= 2.0 ;
    bs_out = wave_out = 0 ; /* zero vectors */
    for (j = 0; j < range + 1; j++)
        wdx = wave_index(L - 1, j) ;
        bdx = bs_index(L - 1, j) ;
        for (k = (2j - 4); k < (2j + 7); k++)
            idx = bs_index(L, k) ;
            bs_out[bdx]   += h[k - 2j] * bs_in[idx] ;
            wave_out[wdx] += g[k - 2j] * bs_in[idx] ;
    bs_out[0] *= 0.5 ;    bs_out[range] *= 0.5 ;
```

And the inverse transformation which implements Equation 17

```
basis_xform_down( bs_in[], wave_in[], bs_out[], L )
    range = 2^L ;
    upRange = 2^(L-1) ;
    bs_in[0] *= 2.0;    bs_in[upRange] *= 2.0 ;
    bs_out = 0 ; /* zero vector */
    for(j = 0; j < range + 1; j++)
        odx = bs_index(L, j) ;
        for(k = (j-5)/2; k <= (j+5)/2; k++)
            bdx = bs_index(L - 1, k) ;
            wdx = wave_index(L - 1, k) ;
            bs_out[odx] += h̃[j - 2k] * bs_in[bdx]
                         + g̃[j - 2k] * wave_in[wdx] ;
    bs_out[0] *= 0.5 ;    bs_out[range] *= 0.5 ;
```

Effectively there are n basis functions in the interval construction. The basis functions away from the boundaries are defined by ϕ and ψ. Because of the mirror reflection, a basis function that overlaps the boundaries must be mirrored back on top of itself, and added together with itself. This alteration is important when symbolically computing Hessian or constraint matrix terms.