Chebyshev Approximation of Discrete Polynomials and Splines

Jae H. Park

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Bradley Department of Electrical Engineering

Leonard A. Ferrari, Chair
A. Lynn Abbott
Peter M. Athanas
Ron Kriz
Hugh F. VanLandingham

Nov. 19, 1999
Blacksburg, Virginia

Keywords: Chebyshev Approximation, Discrete Polynomial, Discrete Spline, Computer Graphics, FIR Filter

Chebyshev Approximation of Discrete Polynomials and Splines

by

Jae H. Park

Leonard A. Ferrari, Chair
Bradley Department of Electrical and Computer Engineering

(Abstract)

The recent development of the impulse/summation approach for efficient B-spline computation in the discrete domain should increase the use of B-splines in many applications. Because we show here how the impulse/summation approach can also be used for constructing polynomials, the approach with a search table approach for the inverse square root operation allows an efficient shading algorithm for rendering an image in a computer graphics system. The approach reduces the number of multiplies and makes it possible for the entire rendering process to be implemented using an integer processor.

In many applications, Chebyshev approximation with polynomials and splines is useful in representing a stream of data or a function. Because the impulse/summation approach is developed for discrete systems, some aspects of traditional continuous approximation are not applicable. For example, the lack of the continuity concept in the discrete domain affects the definition of the local extrema of a function. Thus, the method of finding the extrema must be changed. Both forward differences and backward differences must be checked to find extrema instead of using the first derivative in the continuous domain approximation. Polynomial Chebyshev approximation in the discrete domain, just as in the continuous domain, forms a Chebyshev system. Therefore, the Chebyshev approximation process always produces a unique best approximation. Because of the non-linearity of free knot polynomial spline systems, there may be more than one best solution and the convexity of the solution space cannot be guaranteed. Thus, a Reméz Exchange Algorithm may not produce an optimal approximation. However, we show that the discrete polynomial splines approximate a function using a smaller number of parameters (for a similar minimax error) than the discrete polynomials do. Also, the discrete polynomial spline requires much less computation and hardware than the discrete polynomial for curve generation when we use the impulse/summation approach. This is demonstrated using two approximated FIR filter implementations.

Dedication

This disseration is dedicated to

my late mother,

Young Ryun Choi.

Acknowledgments

I would like to thank my advisor, Dr. Leonard A. Ferrari, for his guidance at each stage of this work, financial support through the years and thorough editing of the manuscript. In many ways, I am indebted to him.

I would also like to thank the other members of my dissertation committee, Professor A. Lynn Abbott, Professor Peter M. Athanas, Professor Ron Kriz and Professor Hugh F. VanLandingham, for their valuable comments and suggestions.

I am indebted to Dr. Dal Y. Ohm, who gave me job to pass through most difficult time during my Ph.d work.

I could not thank enough my parents and my brothers' family for their love and faith in me.

Most of all, my dearest thanks must go to my wife, Wonjung and my princess, Olivia Hanna.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

This dissertation deals with the approximation of arbitrary discrete functions using polynomials and polynomial splines. A k^{th} order polynomial on a continuous domain can be written

$$P(t) = \sum_{i=0}^{k-1} a_i t^i \tag{1.1}$$

where, $a_0, a_1, \cdots, a_{k-1}$ are real coefficients. Polynomials have been studied by mathematicians for a long time. Polynomial splines were introduced by Schoenberg[1] in the 1940s. Splines are used for approximation and interpolation because of their computational properties (e.g. compact representation, computational stability, etc.). Some functions, that are hard to integrate or differentiate, are sometimes first approximated with polynomials. Then, the integral or derivative is evaluated using the approximation polynomials[2,3]. Splines are sometimes used as an approximation in an engineering model when it is hard to describe the system with exact mathematical equations. With the development of the computer, polynomials are used for representing curves and surfaces for computer graphics because of their compact representation of the data and their smoothness properties[4]. Recently, Ferrari, *et al.*[5] developed the impulse/summation approach for fast B-spline calculation on a discrete domain. This approach is extended to more general cases of polynomials and polynomial splines in this

dissertation. It is demonstrated that the computational complexity can be decreased by an order of magnitude per dimension.

To take advantage of the impulse/summation approach, discrete data or discrete functions must be represented using discrete polynomials. For computer graphics, because most of the graphical objects are already described using polynomials, there is no need for approximation or interpolation. On the other hand, data collected with sensor readings or non-polynomial discrete functions need to be approximated or interpolated with polynomials or splines to take advantage of the new computational approach. Interpolation is very easy to compute. Its only constraint is that the interpolant should pass through the exact values of the defined discrete function. There is no definition of error to judge how different the interpolant is from the original data and no reason for concern because no given values are provided between the original data points. While interpolation has an expanding nature, approximation has a compressing nature. Approximation yields a simple function to express many data with a small number of parameters. Thus, it always involves errors and the errors have to be measured for the determination of a suitable approximation. Error measuring usually comes from the distance between the true values and the approximate values in the system space[6]. There are many ways to measure distance. The various distances are based on norms. The nature (existence, uniqueness, and characteristics of the best approximation solution) of the approximation depends on the norm used. When a particular norm is selected, the approximation must minimize the normed error with respect to the approximating function parameters. In this dissertation, the approximating functions are usually polynomials. One of the most popular errors is derived from the least squared error norm because of its smoothness and convexity properties. It is also easy to use. Thus, there are many papers[7,8,9] on least squared error approximation with polynomials. Although this norm has a nice behavior for obtaining approximations and a good solution in the sense of the error, in some applications, it may not be suitable because the solution may have a large local error over a very narrow interval or area. In some applications, it does not produce desirable approximations. For instance, when an approximated image has several large local errors, the approximated image may appear too different from the

original so that the approximation is not acceptable. For this type of application, the Chebyshev norm may work better because the error is confined within a specified limit, which can be made arbitrarily small.

Because a computer works as a discrete system, filters and graphics systems are developed in the discrete domain without necessary reference to continuous systems. So far, most of approximation theory has been developed for the continuous domain. Some aspects of the theory cannot be used in the discrete domain. For example, the concepts of continuity, differentiation and integration do not exist in a discrete domain. The absence of these concepts eliminates some of the constraints for approximation and shrinks the solution space.

1.2 Definitions

In general, a polynomial is defined on $(-\infty, \infty)$; however, only a piece of the polynomial can be used in a real system. Thus, it is convenient to define a polynomial segment. We need two end points in addition to the polynomial coefficient to define a polynomial segment. A polynomial segment $S(t)$, which has its end points at t_j and t_{j+1}, is shown in Figure 1.1.

It is necessary to define one-sided functions in order to explain polynomial splines and B-splines. Let's define a one-sided function for an arbitrary function $f(t)$ as

$$f(t)_+ = \begin{cases} f(t) & for\ t \geq 0 \\ 0 & otherwise \end{cases} \tag{1.2}$$

A polynomial spline is composed of a set of polynomial segments such that the polynomial spline's defined smoothness, (c^n), is preserved at every junction (termed knots) between adjacent except the two last segments. A n^{th} order m knot polynomial spline can be written

$$s(t) = \sum_{i=0}^{k-1} a_i t^i + \sum_{i=2}^{m-1} c_i (t - t_i)_+^{k-1} \tag{1.3}$$

where, $t_1, t_2, \cdots, t_m$ are knots and $t_1 < t_2 < \cdots < t_m$. B-splines are special cases of polynomial splines. A B-spline's value outside of boundary $[t_1, t_m]$ is always zero and its smoothness (c^{k-m}) is preserved at every junction between adjacent segments even at its end points, t_1 and t_m. B-spline can be written

$$q_k(t) = \sum_{i=1}^{m-1} c_i^{[k]} (t - t_i)_+^{k-1} \tag{1.4}$$

where,

$$c_i^{[r]} = \frac{c_i^{[r-1]} - c_{i-1}^{[r-1]}}{t_{i+k-r} - t_i}, \quad c_{i-1}^{[r]} = c_{i+r+1}^{[r]} = 0 \qquad r = 2,3,\cdots,k-1$$

$$k = r,\ c_i^{[k]} = c_i^{[k-1]} - c_{i-1}^{[k-1]},$$

and

$$c_i^{[1]} = \frac{1}{t_{i+k-1} - t_i}, \quad c_{i+1}^{[1]} = \frac{1}{t_{i+k} - t_{i+1}}$$

The last entity is a set of polynomial segments. It is defined by specifiying the knots and the order of the polynomials, which are used for every segment in the same manner as for polynomial splines or B-splines. The difference is that there is no enforced smoothness at the knot locations. A set of polynomial segments can be written

$$P_s(t) = \sum_{i=1}^{m} \sum_{j=0}^{k-1} a'_{ij} (t - t_i)_+^j \tag{1.5}$$

where, a'_{ij} are the real value coefficients for $i = 1,2,\cdots,m$ and $j = 0,1,\cdots,k-1$.

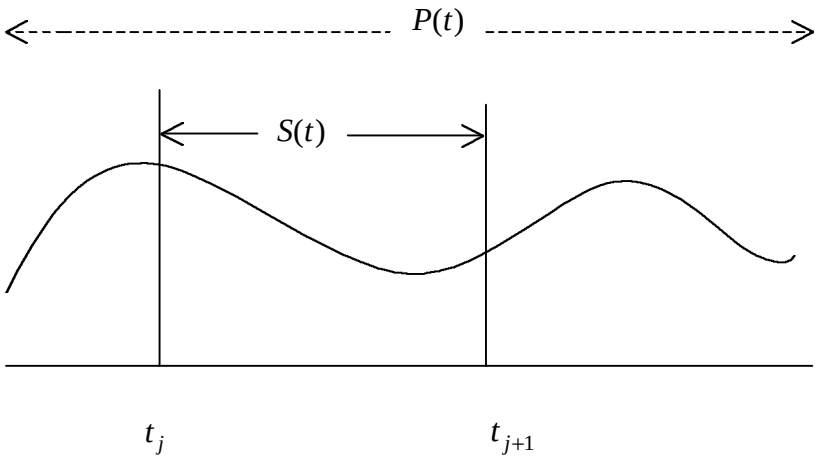


Figure 1.1 A polynomial and one of its segments.

There are many ways to sample a continuous function for producing a counterpart discrete function. However, it is logical to sample the continuous function with an even sampling period because most of the discrete devices are working on evenly distributed discrete spaces (e.g. a computer display system). Thus, discrete polynomials, discrete polynomial splines, discrete B-splines and a set of discrete polynomial segments are composed of samples of their continuous counterparts with an even sampling period.

As discussed before, knot locations are specified for polynomial splines, B-splines and a set of polynomial segments. Sometimes, the knot locations are given and cannot be changed by the interpolation or approximation process. When a polynomial spline or a B-spline has fixed knots, it can be called "a fixed knot polynomial spline" or "a fixed knot B-spline". When the knot locations are allowed to change for a better interpolation result or a better approximation result, the spline can be called "a free knot polynomial spline" or "a free knot B-spline". Even for free knot splines, the knot constraint $t_1 < t_2 < \cdots < t_m$ must be preserved.

For an approximation, an error function has to be defined. For a polynomial approximation, the error function can be written

$$e(t) = f(t) - P(t) \tag{1.6}$$

where $f(t)$ is an arbitrary function to be approximated and $P(t)$ is a polynomial. When only the interval $[a,b]$ is of interest for the approximation, $e(t)$ is defined on only $[a,b]$. The error function $e(t)$ may have local extremes on $[a,b]$. The points, a and b, and the points co-located with the local extremes, are called "alternation points" or "alternations".

1.3 Organization

Chapter 2 provides the necessary background theory for general approximation and Chebyshev approximation on a continuous domain. Most of the theories hold for discrete domain approximation. Also, the general description of the Reméz Exchange Algorithm is provided in this chapter. Because systems of free knot polynomial splines and B-splines form non-linear spaces, the latter part of Chapter 2 is dedicated to some aspects of non-linear approximations. The error alternation characteristics of free knot polynomial splines and B-splines are also discussed at the end of Chapter 2.

In Chapter 3, hardware implementations for some polynomial computations are discussed. The first part describes how the impulse/summation approach is derived from the integration and the delta function of the continuous domain. Next, two classes of discrete bases that can be used with impulse/summation are introduced. Finally, the proper basis usages are presented and implemented for the cases of polynomials, polynomial splines, the B-splines and a set of polynomial segments.

Chapter 4 shows the advantage of using the impulse/summation for shape generation and shading. The best partial differential estimate of each shading scheme is formed and described in the first section. The surface normal calculation complexity and a new method, to reduce computation, follows. In the last part of Chapter 4, we show that by using a smaller sample data patch size in covering a given surface, the shading quality

can be improved without requiring the large computational complexity of existing techniques.

In Chapter 5, we discuss Chebyshev approximation on a discrete domain. The optimality properties of discrete Chebyshev polynomial approximation are shown in Section 5.3. The necessary changes in an algorithm for finding a best approximation follows. Also, two new algorithms for polynomial spline approximation are derived in Section 5.3. Fast filter design examples are shown in Chapter 6 as examples of discrete Chebyshev approximation and the computation of approximated polynomials.

Chapter 2

Review of Chebyshev Approximation

2.1 Introduction

This chapter provides the theoretical background of Chebyshev approximation that is necessary for developing discrete Chebyshev approximation methods. All of the material, discussed in this chapter, comes from existing literature. In Section 2.2, a linear Chebyshev approximation is discussed from the viewpoint of general approximation: existence, uniqueness, characteristics and algorithm. The conditions (for a Chebyshev approximation to have a unique best solution or more than one best solution) appear in Section 2.2.1 and the characteristics of a best solution and the algorithm for the solution are discussed in Section 2.2.2. In the remaining sections, in addition to the four approximation elements of non-linear Chebyshev approximation, the characteristics of best solutions for free knot polynomial splines and free knot B-splines are also discussed in conjunction with alternations.

2.2 Linear Chebyshev Approximation

Let $f(t)$, $f_i(t)$, $i = 1,2,\cdots,k$ be continuous functions defined on $t \in [b_s, b_e]$. An approximation function can be expressed as a linear combination of a set of basis functions, $f_i(t)$:

$$g(A,t) = \sum_{i=1}^{k} a_i f_i(t) \tag{2.1}$$

where, $A = \{a_1, a_2, \cdots, a_k\}$ and $a_1, a_2, \cdots, a_k \in \boldsymbol{R}$.

The Chebyshev approximation problem requires finding A such that

$$\underset{t}{MAX}\left|f(t) - g(A,t)\right| \leq \underset{t}{MAX}\left|f(t) - g(A_j^*,t)\right| \tag{2.2}$$

for any $A_j^* \in R^k$, where j is an index. In other words, the approximation determines $g(A,t)$ for the given $f(t)$. To use Chebyshev approximations in an approximation problem, four elements must be considered. The elements[10] are:

- the existence of a best solution
- the uniqueness of a best solution
- the characteristics of a best solution
- the algorithm to determine a best solution

The characteristics of the space, formed by its basis, $\{f_1(t), f_2(t), \cdots, f_k(t)\}$, determine the existence, the uniqueness, and the characteristics of a best solution for the approximation norm criterion. The solution characteristics give clues for developing an algorithm for determining a best solution. Therefore, the basis space characteristics are very important in any approximation. For a Chebyshev approximation to guarantee the existence of a solution, the basis functions should form a Chebyshev set, or a weak Chebyshev set (both sets will be explained in the next section), over a closed finite interval on which the basis functions are defined. To guarantee the uniqueness of a best solution, the basis functions should form a Chebyshev set. The characteristic of a best solution in Chebyshev approximation is that the error function $f(t) - g(A,t)$ alternates at least $k+1$ times (the detail description will be given in Section 2.2.2). The proof of the alternation theorem can be found in Rivlin[11]. The alternation characteristic only provides the properties of a best solution not a solution method. In the 1930's, E. Y. Reméz developed two algorithms, the First Algorithm and the Reméz Exchange Algorithm for solving Chebyshev approximation problems.

2.2.1 Chebyshev Set and Weak Chebyshev Set

The existence of a best solution means there is always a sequence of $g(A_j^*, t)$ (generated with the j index) that satisfies the equality of Equation 2.2 when $j = \infty$. It is clear that the sequence is uniformly bounded and the set of every $g(A_j^*, t)$ contains a set of pointwise convergent points[12]. Therefore, if the set is compact under pointwise convergence, any continuous function can be approximated with the basis functions and a best approximation exists[14]. Therefore, to have a best solution for a Chebyshev approximation, the basis functions of the approximating function should be continuous. The uniqueness of a best solution for an approximation problem depends on the convexity of the normed solution space. The compactness and the convexity of the space in Chebyshev approximation can be tested by examining whether the basis functions form a Chebyshev set, weak Chebyshev set, or non-Chebyshev set.

A set of basis functions $\{f_1(t), f_2(t), \cdots, f_k(t)\}$ can be called a Chebyshev set, if the determinant of the matrix below is always greater than zero whenever $b_s = t_1 < t_2 < \cdots < t_k = b_e$. The t_i's are strictly ordered by their index (by definition).

$$D(t_1, t_2, \cdots, t_k) = \begin{bmatrix} f_1(t_1) & f_2(t_1) & \cdots & f_k(t_1) \\ f_1(t_2) & f_2(t_2) & \cdots & f_k(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(t_k) & f_2(t_k) & \cdots & f_k(t_k) \end{bmatrix} \tag{2.3}$$

The above matrix can be obtained from the basis functions of $g(A,t)$ at the given $t_1, t_2, \cdots, t_k$. The uniqueness of A for any solution, including a best solution, is guaranteed by the linear independence of the linear equations, $f(t_i) - g(A, t_i)$, for $i = 1,2,\cdots,k+1$. Therefore, it is necessary that the determinant of $D(t_1, t_2, \cdots, t_k)$ is always non-zero for the given set of $t_1, t_2, \cdots, t_k$. This independence is called the Haar condition. For the basis functions to form a Chebyshev set, the determinant of $D(t_1, t_2, \cdots, t_k)$ is always greater than zero. The Chebyshev set characterizes the strict

monotonic increasing nature of the basis functions[13]. A Chebyshev set condition is stronger than the Haar condition. Thus, if the basis functions for an approximation form a Chebyshev set, there exists a best approximation solution for a smooth function. Chebyshev polynomial approximation provides a good example of this.

We may relax the condition of Chebyshev set to apply Chebyshev approximation by allowing the determinant of $D(t_1,t_2,\cdots,t_k)$ to be zero for the given set of $t_1,t_2,\cdots,t_k$. This eliminates the uniqueness of a best solution and the strictness of the monotonic nature of the basis functions. However, with this condition, a Chebyshev approximation still has best approximations or a best approximation. A set of basis functions with this condition is called a weak Chebyshev set. Unfortunately, both polynomial spline basis functions and B-spline basis functions form weak Chebyshev sets. It causes difficulty in using the Reméz exchange algorithm for Chebyshev norm approximation with splines.

2.2.2 The Reméz Exchange Algorithm

In general, the values of a smooth function $f(t)$ are not equal to the values of the corresponding $g(A,t)$ for every $t \in [b_s,b_e]$, unless $g(A,t)$ is the identical function $f(t)$. Therefore, an approximation problem can be handled as an over-determined problem[14]. Most of the early work on approximation started with the over-determined problem approach including Chebyshev approximation. Reméz's first algorithm adapted this approach for a Chebyshev approximation for a continuous function. The algorithm starts with $k+1$ initial discrete points for a k dimensional Chebyshev approximation, then trys to find a best approximation for the initial points. From the best approximation, an error function between the original function and the approximation can be obtained. A point is inserted that does not belong to the initial points and is located at the local error maximum. With this new set of points, the algorithm iterates until all parameters of the approximation function converge. Because only one data point is added at each iteration, the algorithm is called the single exchange algorithm. As the number of iterations increases, the number of points to be considered in the approximation will also increase.

The convergence is slower than Reméz's second algorithm (it is described in a later part of this section), although an advantage is that this algorithm can still be applied when a best solution contains fewer than $k+1$ alternations on its error function.

The mathematical description[15] of a best solution in Chebyshev approximation is;

1) There are $k+1$ alternation points: $b_s = t_1 < t_2 < \cdots < t_{k+1} = b_e$.
2) For any $t \in [b_s, b_e]$, $|f(t) - g(A,t)| \le d$.
3) $|f(t_i) - g(A,t_i)| = d$ for $i = 1,2,\cdots,k+1$.
4) $\frac{d}{dt} f(t) = \frac{d}{dt} g(A,t)$ only for $t = t_2, t_3, \cdots, t_k$

where the t_i's are the alternation points and δ is the estimated optimal error corresponding to the set of t_i's.

By checking 2) in the above description, one may notice that the number of conditions is more than the number of parameters in determining a best solution (as a matter of fact the given conditions are uncountably many for a continuous approximation problem). The second algorithm can be described as iterations of 3) and 4) of the above mathematical description of a best solution. From 3) of the above description, a $(k+1)$ by $(k+1)$ matrix equation can be derived as below.

$$\begin{bmatrix} f_1(t_1) & f_2(t_1) & \cdots & f_k(t_1) & -1 \\ f_1(t_2) & f_2(t_2) & \cdots & f_k(t_2) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ f_1(t_{k+1}) & f_2(t_{k+1}) & \cdots & f_k(t_{k+1}) & (-1)^{k+1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \\ d \end{bmatrix} = \begin{bmatrix} f(t_1) \\ f(t_2) \\ \vdots \\ \vdots \\ f(t_{n+1}) \end{bmatrix} \qquad (2.4)$$

It begins with an arbitrary set of t_i's. For the set of t_i's, the parameters $(a_1, a_2, \cdots, a_k)$ and δ will be estimated using Equation 2.4. Then, a new set of t_i's will be obtained using $a_1, a_2, \cdots, a_k$ by inspecting the first derivatives of the errors between $f(t)$ and $g(A,t)$ as shown in 4) of the description. The new alternation points t_i's locate the local

extreme points of the error function. The iterations of the two steps will be executed until the set of t_i's is converged. The second algorithm still tries to solve the approximation with a set of discrete data points as does the first algorithm. However, it utilizes the fact that there are only $k+1$ alternations on the error function between the best approximation and the original function. Since it finds new $k+1$ alternation points in every iteration and exchanges them with old set of $k+1$ alternation points, the second algorithm is called the multiple exchange algorithm. It converges faster than the first algorithm.

2.3 Non-Linear Chebyshev Approximation

Chebyshev approximation methods can be also used in conjunction with non-linear approximation systems. A non-linear system, like its linear counterpart, also requires parameters to determine shape. Thus, the Chebyshev approximation problem statement in Equation 2.2 is also valid for a non-linear system. There are many classes of non-linear systems. A free-knot polynomial spline and a free-knot B-spline are considered here. Although the free-knot splines give better results than splines with fixed knots, the freedom of the knot locations makes the approximation (or interpolation) problem much more complicated than the equivalent fixed knot problem. The free-knot selection condition makes the approximation system non-linear. The approximation function is a linear combination of linear basis functions and a set of one-parameter non-linear basis functions for the case of a free knot polynomial spline. It is only a linear combination of one-parameter non-linear basis functions in the case of a free knot B-spline. A non-linear approximation function can be written as

$$g(A,\tilde{T},t)=\sum_{i=1}^{m} a_i \tilde{g}(\tilde{t}_i,t) \qquad (2.5)$$

where A, a_i's and t are the same as in a linear system, $\tilde{g}(\cdot)$ is a non-linear function, and $\tilde{T}$ is a vector, consisting of the knots $\tilde{t}_i$. The $\tilde{t}_i$'s act as parameters of $g(A,\tilde{T},t)$.

2.3.1 Existence and Uniqueness

In non-linear approximation, the existence and uniqueness of a best approximation still depends on the convexity of the non-linear space. The convexity of the class of non-linear system described in Equation 2.5 can be verified by checking the convexity of each basis function $\tilde{g}(\cdot)$ for the corresponding parameter $\tilde{t}_i$. The convexity of each non-linear basis can be checked by verifying the equation below[16]:

$$\tilde{g}(\tilde{t}_i, t_2) - \tilde{g}(\tilde{t}_i, t_1) \geq (t_2 - t_1) \frac{\partial \tilde{g}(\tilde{t}_i, t_1)}{\partial t} \tag{2.6}$$

While strict convexity will guarantee existence and uniqueness of a best non-linear approximation, the equality of Equation 2.6 may lead to more than one best approximation because the equality makes the system semi-convex.

Because of the non-linearity, the parameters t_i's are not separable from the basis functions as in a linear system. Thus, even if Equation 2.3 can be formed, the rank of the Equation 2.3 does not represent the order of the non-linear system's degree of freedom (Haar sub-space dimension). When the system consists of only m non-linear basis functions without any linear basis, the number of the parameters for the system is $2m$ and the possible full rank of Equation 2.3 for the system is only m. Therefore, the dimension of the Haar sub-space for the system is $2m$[14]. The independence of each parameter of the system will determine the system's Chebyshev system characteristics as in the case of a linear system. When all parameters are independent from each other, the non-linear system has a unique best approximation.

2.3.2 Characteristics and Algorithm

Meinardus and Schwedt[14] proved the alternating characteristic of the error function between function and its best approximation in a non-linear Chebyshev approximation. The number of the alternation should be $2m+1$ for the approximation described in

Equation 2.5. In a non-linear system, the Chebyshev approximation forms a space that consists of the parameters and the normed error, as in the case of a linear system. The theorem states that the number of alternations is the dimension of the Haar sub-space plus one[14]. Thus, the alternating characteristic is valid for the case where there is dependence between some parameters, and the Haar sub-space has a smaller dimension than the number of parameters and a smaller alternation number than $2m+1$.

When strict convexity is guaranteed, the Gauss-Newton method[14], convex programming[16], and piecewise linearization[14] are possible approaches to solving a non-linear problem. The Gauss-Newton method may converge in some cases, but may only work for fewer alternations than $2m+1$. Convex programming requires strict convexity, so that it will not work for fewer alternations than $2m+1$. The piecewise linearization[14] method is not applicable for some types of basis functions (ex. one-sided power function) because the linearization is usually achieved by Taylor expansion and power functions are the basis function for Taylor expansion.

The Reméz exchange algorithm is the most efficient method of obtaining a best approximation for a non-linear Chebyshev approximation because of the alternating error characteristic. This is especially true, when the number of error alternations is $2m+1$, then the algorithm will generate a best approximation. Even when the number of error alternations is less than $2m+1$, the algorithm converges, but the approximation may not be the best[17]. Lack of uniqueness leads to a possible existence of a local minimum. In many engineering applications this is sufficient.

2.3.3 Polynomial spline Approximation

By Braess[18], a necessary and sufficient condition is derived for a general free knot spline to have a unique best Chebyshev approximation. He proved the existence of best solutions for a polynomial spline Chebyshev approximation of any function that is continuous on the approximating interval. However, he failed to show its characteristics; error alternation and $k+2m$ zero crossing of the error function, where k is the order of

the polynomial spline, and m is the number of knots. For a polynomial spline, it is obviously true because the order of the freedom of a free knot polynomial spline is $k+2m$. (It has k linear basis functions and m non-linear basis functions. The non-linear basis functions' shapes are controlled by the m knot locations.) The basis functions and the mini-max error form a $k+2m+1$ dimensional space. The $k+2m$ zero crossing of the error can be interpreted as the error's $k+2m+1$ alternations. It is an obvious result for a free knot polynomial spline because the order of the freedom is $k+2m$ (it has $k+m$ bases functions and its basis shapes are controlled by m knot locations). The basis functions and the mini-max error form a $k+2m+1$ dimensional space as discussed in previous section. However, the $k+2m+1$ error alternation cannot be guaranteed for a best approximation of a polynomial spline approximation because of gaps between the number of the error alternations for the necessary condition and the number of the error alternations for the sufficient condition[19]. Thus, there is no clear guideline on the number of the error alternations for an algorithm to find a best solution. (The number of error alternation is the information required to estimate the optimal error at each iteration.) However, as Watson wrote, a Reméz exchange algorithm can produce a solution for less than the optimal number of error alternations[14].

2.3.4 B-spline

Since both ends of B-spline function are zero by definition, it is impossible to find an approximation with reasonable error size at both its ends for any function unless the function's values at the endpoints are zero. Thus, B-splines are often used for applications that allow errors at the edges. (ex. Images: Important information is usually not contained in image edges.) Moreover, Karon[19] demonstrates the difficulty of the free knot B-spline Chebyshev approximation due to the difference in the number of alternations in the necessary and sufficient condition for a best solution. This means that there may be a best solution but it is difficult to characterize a best solution in terms of the error alternations, as we were able to do in the case of a polynomial spline approximation. However, it is worth finding an algorithm that obtains a Chebyshev B-

spline approximation of a function in the discrete domain because there is an efficient way to generate the solution, even if it is not the best solution. There are several adaptive methods used to insert and subtract knots (described in Watson[14,20]) for the free knot spline problem.

Chapter 3

Discrete Polynomial Construction

3.1 Introduction

This chapter discusses how to construct a curve defined by a polynomial, polynomial splines, B-splines or a set of polynomial segments in a discrete domain. Section 3.2 explains the conceptual idea of the impulse/summation approach. Development and proof of Theorem 3.1 in Section 3.3 is one of the major contributions in this dissertation. The theorem is the basis used to apply the impulse/summation approach to polynomials. (Techniques for applying the impulse/summation approach to B-spline construction, shown in Section 3.3.4, were done by Wang[34].) The impulse/summation approach requires polynomials to be expressed using only one-sided power functions and Theorem 3.1 provides the transformation. By the same transformation, a polynomial spline can be written as a linear combination of one-sided power functions (discussed in Section 3.3.2). In Section 3.4, two kinds of basis functions, discrete one-sided power functions and discrete one-sided factorial functions are introduced. Both were developed sometime ago and used in numerous studies of spline curve construction. However, in this study, the definition of the discrete one-sided factorial function is modified to reduce the number of multiplications in its usage. (A new definition is given in Section 3.4.2.) Section 3.5 shows how to construct polynomials, polynomial splines, or B-splines efficiently with the impulse/summation approach.

3.2 Discrete Polynomials

The discrete polynomial segment is the essential block for building discrete splines and sets of piecewise discrete polynomial segments. Even for a polynomial approximation, only a polynomial segment within an approximate boundary is needed. Thus, defining a discrete polynomial is the first step in what is needed to do. A discrete polynomial can be obtained by sampling a continuous polynomial with a uniform sampling period as discussed in Section 1.1. Consider a k^{th} order polynomial on a continuous domain

$$P(t) = \sum_{i=0}^{k-1} a_i t^i \tag{3.1}$$

where, $a_0, a_1, \cdots, a_{k-1}$ are real coefficients. Let's assume, $P(t)$ is sampled by periodic sampling and $\mathrm{d}t$ is the sampling period, then the sampled sequence would be

$$P(n \cdot \mathrm{d}t) = \sum_{i=0}^{k-1} a_i (n \cdot \mathrm{d}t)^i \tag{3.2}$$

The corresponding discrete polynomial can be written as

$$P[n] = \sum_{i=0}^{k-1} a_i n^i \tag{3.3}$$

by setting $P[n] = P(n \cdot \mathrm{d}t)$. Equation 3.3 represents a discrete k^{th} order polynomial in terms of the basis n^i, $i = 0,1,\cdots,k-1$. The essential building block of a polynomial, whether in the continuous or the discrete domain, is the power basis. As Üstüner[21] pointed out, a basis t^i can be obtained by integrating a lower degree basis t^{i-1} or differentiating a higher degree one t^{i+1}, if t is a real number. However, for t^0 and t^{-1} this iterated relationship is not valid. Since t^0 is unity its derivative should be zero. If t is defined on $(-\infty, \infty)$, there is no way to recover t^0 from zero. Almost every case of approximation, especially in the discrete domain works on a well-bounded domain. The approximating function (interpolating function) values outside of the defined domain can

be considered zero or can be ignored, so that setting zero for the values outside of the boundaries does not change the shape of the curve. In some cases (e.g. FIR filters), the outside values should be zeros because an FIR filter has finite length. For a better explanation, let the domain be $[0,\infty)$. This assumption provides a way to construct t^0 by integrating a delta function. Also, t^0 on $[0,\infty)$ can be considered as a unit step function $u(t)$:

$$t^0 = u(t) = \int_{0-}^{t} d(y)dy \tag{3.4}$$

Thus, a k^{th} order continuous polynomial defined on $[0,\infty)$ would be expressed as

$$\begin{aligned} P(t) &= a_0 t^0 + a_1 t^1 + \cdots + a_{(k-1)} t^{(k-1)} \\ &= a_0 \int_{0-}^{t} \delta(y_1)dy_1 + a_1 \int_{0-0-}^{t\ y_1} \delta(y_2)dy_2 dy_1 + \\ &\quad \cdots + a_{(k-1)} k! \int_{0-0-}^{t\ y_1} \cdots \int_{0-}^{y_{k-1}} \delta(y_k)dy_k dy_{k-1} \cdots dy_1 \\ &= \int_{0-}^{t} (a_0 \delta(y_1) + \int_{0-}^{y_1} (a_1 \delta(y_2) + \\ &\quad \cdots + \int_{0-}^{y_{k-1}} a_{(k-1)} k! \delta(y_k)dy_k)dy_{k-1})\cdots)dy_1 \end{aligned} \tag{3.5}$$

We notice that the last line of the above equation indicates that the multiplication is required only at each delta functions. The construction of $P(t)$ can be done with the k-tuple integration of the delta functions, scaled with polynomial coefficients. If integration can be executed faster or simpler than multiplication in a system, the above equation would give a way to construct a polynomial faster or simpler than the brute force method suggested by Equation 3.1. In fact, the number of multiplications required to construct a polynomial can be reduced substantially. Consider a k^{th} order polynomial that needs ten points to be constructed with a reasonable appearance in a system. With the brute force method, it takes at least $20k$ multiplies to construct the polynomial, even

using Horner's rule[22], which is considered very efficient. It requires only k multiplies with the integration/delta function approach. Of course, there is no concept of integration or differentiation in a discrete system because we cannot define a concept of continuity. The successive summation and differences of the discrete domain are analogous to integration and differentiation in the continuous domain. Therefore, the relationship among the power bases of the continuous domain cannot be applied to the power bases of the discrete domain in exactly the same way, especially between n^0 (for convenience, let's call n^0 the constant basis) and its lower degree basis. Two different approaches have been suggested for a discrete version of the integration/delta function approach. Üstüner[21], Mangasarian, and Schumaker[23,24] introduced a new set of basis functions that has one impulse on its i^{th} derivative for the i^{th} basis function. It is called the factorial basis. Ferrari, *et. al.*[25] used the discrete power function n^i as a basis. It has multiple impulses on the i^{th} derivatives of each basis function n^i. These two bases are discussed in later sections of this chapter.

3.3 Transformation for Construction

3.3.1 Polynomial

A polynomial is defined on $(-\infty, \infty)$, but usually an approximation process is only used on a finite domain. As discussed in Section 3.2, the polynomial should be segmented in order to utilize the integration/delta function approach for construction. Usually, a polynomial is expressed with Equation 3.1. It does not carry any information about where the segment starts and where the segment ends. Thus, it is necessary to choose a method to express a segmented polynomial. The expression should include a start point, an end point and the polynomial parameters. Consider a polynomial $P(t)$ and one of its segments $S(t)$ in Figure 3.1. Before discussing a polynomial segment, it is convenient to first define a one-sided polynomial:

$$O_i(t) = \begin{cases} P(t) & t_i \le t \\ 0 & t_i > t \end{cases} \tag{3.6}$$

$O_{i+1}(t)$ is defined in the same manner but with a different starting point t_{i+1}. $S(t)$ can be expressed as $S(t) = O_i(t) - O_{i+1}(t)$.

Theorem 3.1 For any order $k > 0$ of the polynomial $P(t)$, there always exists a unique set of a_i' for the polynomial such that

$$P(t) = \sum_{i=0}^{k-1} a_i t^i = \sum_{i=0}^{k-1} a_i'(t-h)^i \tag{3.7}$$

for any real number h. See Appendix A for a proof.

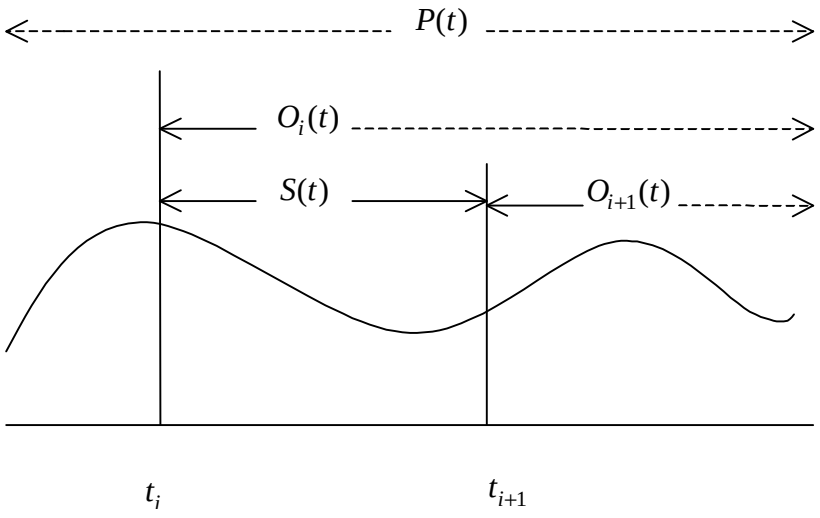


Figure 3.1 A polynomial segment $S(t)$

Theorem 3.1 can be extended to the case that $(t-h)^i$ substitutes t^i in Equation 3.7 and $(t-h')^i$ substitutes $(t-h)^i$ in Equation 3.7 where $h \neq h'$. As shown in Equation 3.6, the one-sided polynomial $O_i(t)$ has the value of $P(t)$ for $t \ge t_i$. Therefore, to represent a

one-sided polynomial, it is useful to define a one-sided basis function (power function) such as;

$$(t-h)_+^n = \begin{cases} (t-h)^n & h \le t \\ 0 & h > t \end{cases} \tag{3.8}$$

for $n \ge 0$. With Theorem 3.1 and the one-sided power basis functions $O_i(t)$, the one-sided polynomial can be written as

$$O_i(t) = \sum_{j=0}^{k-1} a_j'(t-t_i)_+^i \tag{3.9}$$

Thus,

$$S(t) = O_i(t) - O_{i+1}(t) = \sum_{j=0}^{k-1} a_j'(t-t_i)_+^i - \sum_{j=0}^{k-1} a_j''(t-t_{i+1})_+^i \tag{3.10}$$

where $O_{i+1}(t) = \sum_{i=0}^{k-1} a_i''(t-t_{i+1})_+^i$.

3.3.2 Polynomial Spline

As shown in Figure 3.2, a simple polynomial spline on $[t_1, t_m]$ can be defined with a chain of polynomial segments, $s_1(t), s_2(t), \cdots, s_{m-1}(t)$. To maintain the continuity between the adjacent segments, the segments, must share not only the same function value, but also the same derivative value at each junction point between the segments[26]. A k^{th} order, m simple knot spline curve can be represented by polynomial segment coefficients. $k(m-1)$ parameters are needed. This implies that a spline forms a $k(m-1)$ dimensional space. As matter of fact, in general, splines are not required to keep the specified continuity. The continuity is a property that can be controlled for some purpose, ex. 2-5-

2 spline[27]. Therefore, to express an uneven spaced, multiple knot spline, it is necessary to keep all of the coefficients of all of the segments belonging to the spline. However, we are not only concerned with the simple knot case in this section. A k^{th} order spline has C^{k-2} continuity for $k > 3$. The spline can be expressed with the set of basis functions[13,28]:

$$\left\{1, t, \cdots, t^{k-1}, (t-t_2)_+^{k-1}, \cdots, (t-t_{m-1})_+^{k-1}\right\} \tag{3.11}$$

Because the one-sided functions are used at each knot location as a basis (except at both endpoints), expressing the spline on this set of basis functions will guarantee C^{k-2} continuity at every junction between adjacent segments including both end points. With this basis, the spline can be written

$$s(t) = \sum_{i=0}^{k-1} a_i t^i + \sum_{i=2}^{m-1} c_i (t-t_i)_+^{k-1} \qquad \text{for } [t_1, t_m]. \tag{3.12}$$

The spline requires a smaller number of parameters than the general polynomial segment. Equation 3.12 consists of a k^{th} order polynomial (the first sums on the right side of Equation 3.12) and a sum of $k-1$ degree one-sided power functions at the knot locations, $t_2, t_3, \cdots, t_{m-1}$ (the second sums of the right side in Equation 3.12). The k^{th} order polynomial portion (the first sums of the right side in Equation 3.12) of the spline can be transformed into another set of basis functions, $\left\{(t-t_1)^0, (t-t_1)^1, \cdots, (t-t_1)^{k-1}\right\}$ by Theorem 3.1. Since $s(t)$ is defined only on $[t_1, t_m]$, the values of each basis function, which lies outside of $[t_1, t_m]$, can be considered as zero because we do not care about the out of interval values. Thus, the basis set $\left\{(t-t_1)^0, (t-t_1)^1, \cdots, (t-t_1)^{k-1}\right\}$ can be substituted for by the set of the one-sided power functions $\left\{(t-t_1)_+^0, (t-t_1)_+^1, \cdots, (t-t_1)_+^{k-1}\right\}$ without changing the shape of $s(t)$. New basis functions for $s(t)$ can be transformed from the set of the basis functions in Equation 3.11. A new basis set follows

$$\left\{(t-t_1)_+^0,(t-t_1)_+^1,\cdots,(t-t_1)_+^{k-1},\cdots,(t-t_{m-1})_+^{k-1}\right\} \tag{3.13}$$

and $s(t)$ can be written as

$$s(t)=\sum_{i=0}^{k-2}a_i'(t-t_1)_+^i+\sum_{i=1}^{m-1}c_i'(t-t_i)_+^{k-1} \tag{3.14}$$

where

$$\begin{bmatrix}a_0\\a_1\\a_2\\\vdots\\a_{k-1}\end{bmatrix}=\begin{bmatrix}1&0&0&\cdots&0\\-t_1&1&0&&0\\(-t_1)^2&-2t_1&1&&0\\\vdots&&&\ddots&\vdots\\(-t_1)^{k-1}&&\cdots&&1\end{bmatrix}^{-\mathrm{T}}\begin{bmatrix}a_0'\\a_1'\\\vdots\\a_{k-2}'\\c_1'\end{bmatrix}$$

and $c_i'=c_i$ for $i=2,3,\cdots,m-1$ (See Appendix B for the coefficient relationship). $s(t)$ can be constructed with Equation 3.14 using the integration and delta function approach.

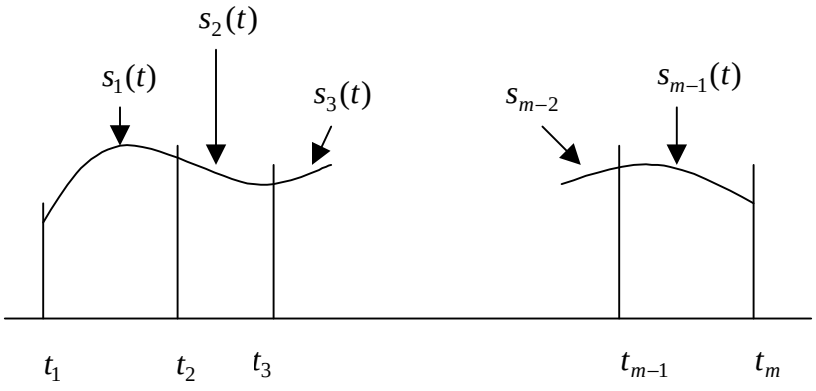


Figure 3.2 A Polynomial Spline $s(t)$

3.3.3 B-Spline

A k^{th} order B-spline with m knots is defined

$$q(t) = \sum_{i=1}^{m} V_i B_{i,k}(t) \tag{3.15}$$

where $V_1, V_2, \cdots, V_m$ are real coefficients (usually, these are called 'vertices') and $B_{1,k}(t), B_{2,k}(t), \cdots, B_{m,k}(t)$ are the basis functions for the B-spline $q(t)$. The knots are $t_1, t_2, \cdots, t_{m+k}$. Thus, $q(t)$ is defined on $\left[t_1, t_{m+k}\right]$ as shown in Figure 3.3. Because we are concerned with simple knot B-splines, $t_1 < t_2 < \cdots < t_{m+k}$. Equation 3.15 is also valid for multiple knot B-splines.

There are several computational procedures used to produce a B-spline curve $q(t)$. Each scheme is based on the technique used to construct the B-spline basis function. Cox[29] and de Boor[30] found the recursive relation for the B-spline basis function as follows:

$$B_{i,j}(t) = \frac{t - t_i}{t_{i+j-1} - t_i} B_{i,j-1}(t) + \frac{t_{i+j} - t}{t_{i+j} - t_{i+1}} B_{i+1,j-1}(t) \qquad j = 2,3,\cdots,k \tag{3.16}$$

where

$$B_{i,1}(t) = \begin{cases} 1 & t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

Bartels, et. al.[31] show an equation using a divided difference[32] operator for a B-spline basis:

$$B_{i,k}(t) = (-1)^k (t_{i+k} - t_i)[t_i, t_{i+1}, \cdots t_{i+k} : x](t - x)_+^{k-1} \tag{3.17}$$

where the divided difference operator $[t_i, t_{i+1}, \cdots t_{i+k} : x]$ is defined

$$[t_i, t_{i+1}, \cdots t_{i+k} : x] f(x) = \frac{[t_{i+1}, \cdots t_{i+k} : x] f(x) - [t_i, \cdots t_{i+k-1} : x] f(x)}{t_{i+k} - t_i}$$

It is difficult to apply the integration/delta function method with these approaches. However, for a k^{th} order simple knot B-spline, a basis set can be written using just the set of k^{th} order one-sided power functions as

$$B_{i,k}(t) = \sum_{j=i}^{i+k} c_j^{[k]} (t - t_j)_+^{k-1} \tag{3.18}$$

where $c_i^{[k]}, c_{i+1}^{[k]}, \cdots, c_{i+k}^{[k]}$ are real coefficients. Therefore, the entire $q(t)$ can be written as a linear combination of a set of the one-sided power functions and the integration/delta function approach can be applied. The number of polynomial segments from t_i to t_{i+k} is $k-1$, but there are k one-sided power functions in the basis of Equation 3.18. One may notice from Equation 3.16, that $B_{i,k}(t)$ and its derivatives up to the $(k-2)^{\text{th}}$ derivative at t_{i+k} should be zero (One may see why both ends of $q(t)$ are zero in Figure 3.3). Thus, $(t - t_{i+k})_+^{k-1}$ is necessary to satisfy the end condition of the B-spline base and the coefficients $c_i^{[k]}, c_{i+1}^{[k]}, \cdots, c_{i+k}^{[k]}$ cannot be arbitrary. They are dependent on the knots $t_i, t_{i+1}, \cdots, t_{i+k}$, which define the B-spline base. These coefficients can be obtained from the recursive equations[25, 33, 34]:

$$c_i^{[r]} = \frac{c_i^{[r-1]} - c_{i-1}^{[r-1]}}{t_{i+k-r} - t_i}, \quad c_{i-1}^{[r]} = c_{i+r+1}^{[r]} = 0 \qquad r = 2,3,\cdots,k-1 \tag{3.19}$$

where, $k = r$, $c_i^{[k]} = c_i^{[k-1]} - c_{i-1}^{[k-1]}$, and $c_i^{[1]} = \dfrac{1}{t_{i+k-1} - t_i}$, $c_{i+1}^{[1]} = \dfrac{1}{t_{i+k} - t_{i+1}}$.

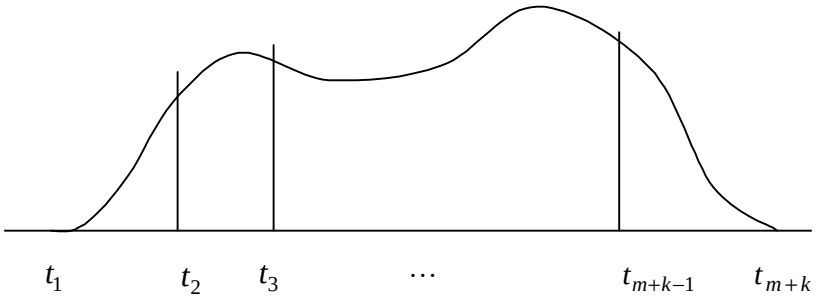


Figure 3.3 A B-spline $q(t)$

3.3.4 Set of Polynomial Segments

Let the set of polynomial segments be $S(t)$ and each segment $s_i(t)$ have non-zero values only on $[t_i, t_{i+1}]$ as shown in Figure 3.4. Then,

$$S(t) = \sum_{i=1}^{m-1} s_i(t) \tag{3.20}$$

and

$$s_i(t) = \sum_{j=1}^{k-1} a_{ij} t^j \tag{3.21}$$

where a_{ij}'s are the real coefficients, $i = 1,2,\cdots,m-1$ and $j = 0,1,\cdots,k-1$. By Theorem 3.1, Equation 3.19 can be rewritten as the difference of two one-sided polynomials that obtains their non-zero values over the knot interval t_i to t_{i+1}. Because all $s_i(t)$ are

followed by the next indexed segment $s_{i+1}(t)$ except the last segment (ref. Figure 3.4), $S(t)$ can be written using the summation of the set of the one-sided polynomials $O_i'(t)$'s as

$$S(t) = \sum_{i=1}^{m} O_i'(t) = \sum_{i=1}^{m} \sum_{j=0}^{k-1} a_{ij}' (t - t_i)_+^j \tag{3.22}$$

where, the a_{ij}' are the coefficients of the one-sided polynomials (see Appendix C on how to obtain a_{ij}' from a_{ij}). One may note that the set of polynomial segments can be considered as multiple knot splines by examining Equation 3.22. For a spline, the value of the multiplicity at a particular knot location appears to have more basis functions (of lower degree, one-sided power functions) at the knot location. For example, if a double multiple knot is formed at t_i, the spline has not only $(t - t_i)_+^{k-1}$ as a basis function, but also $(t - t_i)_+^{k-2}$. Therefore, the set of polynomial segments can be treated as a spline that has k multiple knots at every knot location. However, in this case the integration/delta function approach to computation can not provide any advantage.

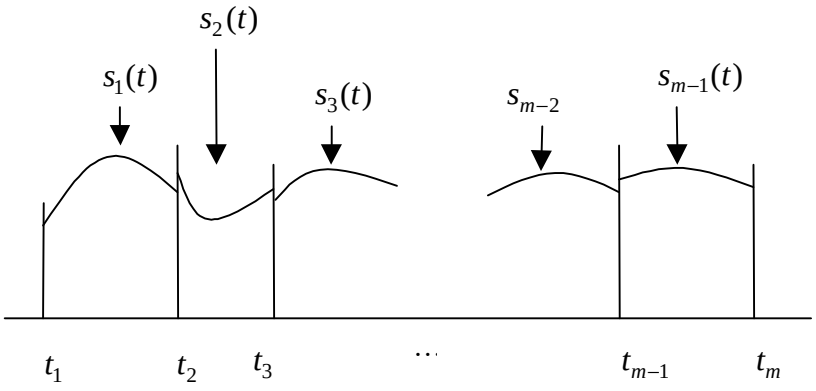


Figure 3.4 A Set of Polynomial Segments

3.4 Discrete Bases

Implementations of an integrator and a multiplier in an analog system require almost the same number of components. For an integrator, one OP amp, two (or three resistors for a differential amplifier) and a capacitor are needed and, for a multiplier, one OP amp and two resisters may be needed. There is no significant performance difference. As discussed in Section 3.2, successive summation in the discrete domain is analogous to the analog integrator in the continuous domain. Successive summation can be implemented using a single adder and a delay as shown in Figure 3.5. Also, in a discrete system, the implementation of an adder is much simpler than the implementation of a multiplier, and needs a smaller number of clock cycles than a multiplier when a position weighted number system[35] (a conventional binary number system) is used.

Because the integration of a delta function is replaced with the recursive summation of a discrete sample point in the discrete system, the discrete version of the integration/delta function approach will be called the impulse/summation approach from this point. Let's define a k cascaded successive summation operator as in Equation 3.23 and refer $(\cdot)_{[k]}$ as the repeated summation operator[27].

$$(f[n])_{[k]} = \sum_{r_k=0}^{n} \cdots \sum_{r_2=0}^{r_3} \sum_{r_1=0}^{r_2} f[r_1] \tag{3.23}$$

It can be easily implemented by cascading the hardware shown in Figure 3.5 k times. However, in general, operating k^{th} repeated summations on a single impulse in a discrete system does not exactly produce the desired k^{th} order, one-sided, discrete power function as was discussed in Section 3.2. In the following two sub-sections, two solutions to this problem are presented. The first one needs multiple impulses on the k difference of the one-sided discrete power function and produces the function simply using k^{th} repeated summation. The second approach uses a single impulse on the k difference and produces a new set of functions (one-sided factorial functions) using the k^{th} repeated summation.

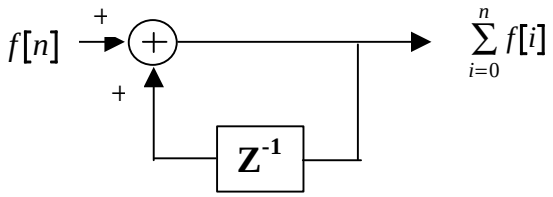


Figure 3.5 A successive summation implementation

3.4.1 One-sided discrete power function base

The discrete version of the one-sided power function is identical with its counterpart in the continuous domain. The only difference is that the function will only take an evenly distributed discrete number or integers as a variable. Thus, it is valid to use the continuous version of the one-sided power function for the discrete version. Let the k^{th} order discrete one-sided power function be written

$$\tilde{b}_k[n] = (n)_+^{k-1} \tag{3.24}$$

where n is an integer.

Let

$$F[n] = \sum_{i=0}^{n} f[n] \tag{3.25}$$

then,

$$F[n] = \sum_{i=0}^{n-1} f[i] + f[n]$$
$$= F[n-1] + f[n] \qquad (3.26)$$

$$\Rightarrow \quad f[n] = F[n] - F[n-1]$$

From Equation 3.26, $f[n]$ can be considered as a difference of $F[n]$ and the difference operation $(F[n] - F[n-1])$ is the inverse operator of the successive summation. Let's denote ∇ for the difference, and ∇^k for the k times application of the operator ∇. Since the operator is defined as $F[n] - F[n-1]$, let's call it the backward difference.

To obtain a corresponding impulse pattern for each one-sided discrete power function $\tilde{b}_k[n]$, the operator ∇^k should be applied to $\tilde{b}_k[n]$. For the example of the 4th order, one-sided discrete power function, the table can be formed as

n	0	1	2	3	4	5	6	…
$\tilde{b}_4[n]$	0	1	8	27	64	125	216	…
$\nabla\tilde{b}_4[n]$	0	1	7	19	37	61	…	…
$\nabla^2\tilde{b}_4[n]$	0	1	6	12	18	…	…	…
$\nabla^3\tilde{b}_4[n]$	0	1	5	6	6	6	6	…
$\nabla^4\tilde{b}_4[n]$	0	1	4	1	0	0	0	…

Table 3.1 Backward differences of $\tilde{b}_4[n]$ for obtaining the impulse pattern of the 4 differences of $\tilde{b}_4[n]$

One-sided discrete power functions of other orders can be obtained by forming similar tables as Table 3.1. The impulse patterns of the k^{th} order one-sided discrete power functions on k differences are shown in Table 3.2. The other method for obtaining these impulse patterns is presented by Refling[36]. It uses an algebraic recurrence for the Z-transform of the one-sided power function to obtain Table 3.2 directly. At this point, it is clear that it takes only a few multiplications for the evaluation of $c\tilde{b}_k[n]$, $c \in$ real, with the

impulse and the repeated summation approach. Multiplication is only needed at the impulse locations. There is another impulse pattern suggested by Wang[34]. However, her scheme requires a few more impulses for each basis than this scheme. One draw back of this scheme is, as the order k of the base increases, the required number of impulses on its k differences increases. Consequently, the number of the required multiplications and the order of the repeated summation increase. Therefore, it may not be suitable for very high order polynomials.

There is another difference between the k^{th} derivatives of the continuous one-sided power function and the k differences of the discrete one-sided power function. The k cascaded integration of a delta function is not simply $(t)_+^{k-1}$. It is $\frac{1}{(k-1)!}(t)_+^{k-1}$. Thus, every integration generates a scale factor in the continuous domain. However, the successive summation of $\nabla^k \tilde{b}_k[n]$ does not generate a scale factor and, therefore, eliminates several additional multiplications.

n	0	1	2	3	4	5	6	…
$\nabla \tilde{b}_1[n]$	1	0	0	0	…	…	…	…
$\nabla^2 \tilde{b}_2[n]$	0	1	0	0	0	…	…	…
$\nabla^3 \tilde{b}_3[n]$	0	1	1	0	0	…	…	…
$\nabla^4 \tilde{b}_4[n]$	0	1	4	1	0	0	…	…
$\nabla^5 \tilde{b}_5[n]$	0	1	11	11	1	0	0	…
$\nabla^6 \tilde{b}_6[n]$	0	1	26	66	26	1	0	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 3.2 The impulse patterns of the k^{th} order one-sided discrete power functions on k differences

One may raise a question: What will happen to the differences of $\tilde{b}_k[n]$ beyond the k differences? When the backward difference is applied to $\nabla^k \tilde{b}_k[n]$, there will be non-zero

elements in the higher order differences of $\nabla^k \tilde{b}_k[n]$ and the number of non-zero elements will be greater than those of $\nabla^k \tilde{b}_k[n]$ (ex. $\nabla^3 \tilde{b}_3[n] = [\cdots, 0, 1, 4, 1, 0, \cdots]$ and $\nabla(\nabla^3 \tilde{b}_3[n]) = [\cdots, 0, 1, 3, -3, -1, 0, \cdots]$). For the sake of implementation, when more than two one-sided power functions are included in a system, it makes it possible for the system to be designed with only one input stream, but the required number of multiplications may increase significantly. Thus, the assumption, $\nabla^j \tilde{b}_k[n] = 0$ for $j > k$, is convenient. The assumption is also proved by Brualdi[36, 37].

3.4.2 One-sided factorial function

The definition of one-sided factorial function, suggested by Mangasarian and Shumaker[23,24], is

$$b_k[n] = \left(\prod_{i=1}^{k} (n - (k-1)) \right) \tag{3.27}$$

where $b_0[n] = (n)_+^0$ and $n \geq 0$. The reason, it is called a one-sided factorial function, is that Equation 3.27 can be rewritten with factorials as

$$b_k[n] = \frac{n!}{(n-k)!} \tag{3.28}$$

for $n \geq 0$. Let's define the forward difference operation Δ as below

$$\Delta f(n) = f(n+1) - f(n) \tag{3.29}$$

and apply it to Equation 2.28. Then,

$$
\begin{aligned}
\Delta b_k[n] &= b_k[n+1] - b_k[n] \\
&= \left(\frac{(n+1)!}{(n+1-k)!} - \frac{n!}{(n-k)!} \right) && \text{for } n \geq 0 \\
&= \frac{n!\big(n+1-(n+1-k)\big)}{(n+1-k)!} && \text{for } n \geq 0 \\
&= k\left(\frac{n!}{(n-(k-1))!} \right) && \text{for } n \geq 0 \\
&= k b_{k-1}[n]
\end{aligned}
\tag{3.30}
$$

Conversely,

$$
b_k[n] = b_k[n-1] + k b_{k-1}[n-1] \qquad \text{for } n \geq 0 \tag{3.31}
$$

The recursive nature of Equation 3.31 leads to an expression of $b_k[n]$ in terms of the repeated summations of $b_0[n]$ as

$$
b_k[n] = k! \sum_{r_k=-\infty}^{n-1} \sum_{r_{k-1}=-\infty}^{r_k-1} \cdots \sum_{r_1=-\infty}^{r_2-1} b_0[r_1] \tag{3.32}
$$

Note that $b_0[n]$ has more than one non-zero value as a function of n. It means, more than one multiplication is needed when $cb_k[n]$ is calculated with the impulse/summation scheme. Fortunately, $\Delta b_0[n]$ has only one non-zero value at n = -1. It is convenient to have $b_{-1}[n] = \Delta b_0[n]$[21]. Therefore, Equation 3.32 can be rewritten in terms of $b_{-1}[n]$ as

$$
b_k[n] = k! \sum_{r_k=-\infty}^{n-1} \sum_{r_{k-1}=-\infty}^{r_k-1} \cdots \sum_{r_0=-\infty}^{r_1-1} b_{-1}[r_0] \tag{3.33}
$$

The relationship between $b_k[n]$ and $b_{k-1}[n]$, shown in Equation 3.29, is similar to the integration and derivative relationship between the continuous one-sided power functions $(t)_+^k$ and $(t)_+^{k-1}$. However, the term $k!$ is not helpful in reducing the implementation

complexity as was discussed in the previous section. Thus, it is better to re-define the one-sided factorial function as

$$b_k[n] = \frac{1}{k!}\left(\prod_{i=1}^{k}(n-(k-1))\right) \tag{3.34}$$

with the same $b_{-1}[n]$ and $b_0[n]$ in order to eliminate the need for multiplication at each successive summation. Consequently, Equation 3.33 can be rewritten as

$$b_k[n] = \sum_{r_k=-\infty}^{n-1}\sum_{r_{k-1}=-\infty}^{r_k-1}\cdots\sum_{r_0=-\infty}^{r_1-1} b_{-1}[r_0] \tag{3.35}$$

Therefore, any order k of the one-sided factorial function can be constructed from a single impulse of its k differences using only integer summations. Table 3.3 shows the newly defined one-sided factorial functions.

n	-1	0	1	2	3	4	5	…
$b_{-1}[n]$	1	0	0	0	0	0	0	…
$b_0[n]$	0	1	1	1	1	1	1	…
$b_1[n]$	0	0	1	2	3	4	5	…
$b_2[n]$	0	0	0	1	3	6	10	…
$b_3[n]$	0	0	0	0	1	4	10	…
$b_4[n]$	0	0	0	0	0	1	5	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 3.3 One-sided factorial functions

According to Üstüner, there is always a unique set of a_i's and a_i''s for any order $k > 0$ of the discrete one sided polynomial $P[n]$ such that

$$P[n] = \sum_{i=0}^{k-1} a_i \tilde{b}_i[n] = \sum_{i=0}^{k-1} a_i' b_i[n] \tag{3.36}$$

Thus, any one-sided polynomial can be constructed with either one-sided power functions or one-sided factorial functions.

3.4.3 Shift Invariance

The successful construction of a segmented polynomial, a set of polynomial segments, a polynomial spline, or a B-spline with the impulse/summation scheme depends on the exact construction of the one-sided power functions that start with non-zero values at their knot locations. This is also true for the one-sided factorial functions. The repeated summation's shift invariant property is guaranteed because applying the repeated summations on the shifted impulse (or the shifted impulse pattern for the one-sided power function) generates the one-sided factorial function. Although it is easy to see that it is shift invariant system, a proof is necessary.

Theorem 3.2 Whenever $y[n] = (f[n])_{[k]}$ for any real number function $f[n]$, it is always true that $y[n-n_0] = (f[n-n_0])_{[k]}$ for any real number n_0. See Appendix D for a proof.

3.5 Construction

The repeated summation operator is the basic component of the impulse/summation scheme of both the one-sided power functions and the one-sided factorial functions. The impulse pattern in the basis function's k differences is determined by the type of basis functions that the system consists of (one-sided power basis functions or the one-sided factorial basis functions). As mentioned in Section 3.4, the repeated summation operator can be implemented by cascading simple successive summations which are composed of one summation and one delay. The hardware for constructing $b_k[n]$ and the required

input for the hardware are shown in Figure 3.6. Simply exchanging the $\nabla^k \tilde{b}_k[n]$ impulse pattern for the single $d[n+1]$ is all that is required in order to generate $\tilde{b}_k[n]$ with Implementation-1 shown in Figure 3.6. In general, the larger the number of impulses on the k differences means the larger the required number of multiplications. Therefore, there are situations, in which using the one-sided factorial functions as basis functions instead of the one-sided power functions provides advantages. However, the spline and B-spline are expressed using piecewise polynomials and the coefficient transformation from power functions to one-sided factorial functions may increase the required number of basis functions. Thus, there are some other cases, for which using the one-sided power functions as a basis or using both types of functions provides simpler hardware implementations. (See implementations in Example 3.1 and Example 3.2.)

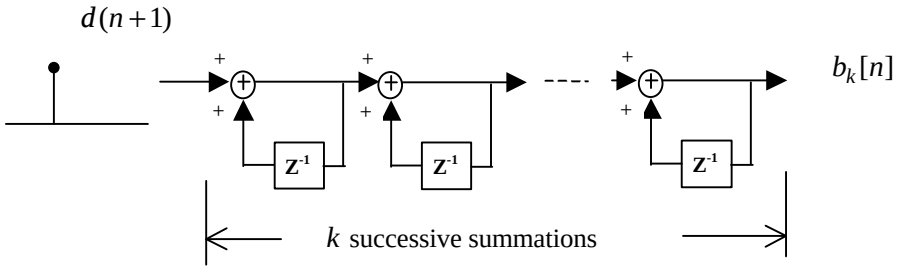


Figure 3.6 Implementation-1: The hardware implementation of the k th order repeated summation

3.5.1 Polynomials

Even though Equation 3.10 can be used to describe a polynomial segment, in some applications construction of the polynomial requires only the one sided polynomial coefficients and its end point. The polynomial values following the end point are not important or are not needed. Thus, the one-sided polynomial implementation is better for this type of application. A discrete version of a one-sided polynomial can be obtained by even sampling of the continuous one-sided polynomial shown in Equation 3.9 and can be written as

$$O_i[n] = \sum_{j=0}^{k-1} a_j (n - n_i)_+^i = \sum_{j=0}^{k-1} a_j \tilde{b}_j (n - n_i) \tag{3.37}$$

where the a_i's are real numbers and n_i is $O_i[n]$'s starting point. Then, using the repeated summation operator,

$$O_i[n] = \sum_{j=0}^{k-1} a_j \left(\nabla^{(j)} \tilde{b}_j (n - n_i) \right)_{[j]} \tag{3.38}$$

Equation 3.38 can be implemented with Implementation-2 of Figure 3.7. The repeated summation can be broken down into single successive summations and re-arranged as

$$\begin{aligned} O_i[n] = & \sum_{j=-\infty}^{n_{ie}} (a_1 \nabla \tilde{b}_1[j - n_i] + \sum_{r_{k-1}}^{j} (a_2 \nabla^2 \tilde{b}_2[r_{k-1} - n_i] + \\ & \cdots + \sum_{r_1}^{r_2} a_{k-1} \nabla^{k-1} \tilde{b}_{k-1}[r_1 - n_i])) \cdots) \end{aligned} \tag{3.39}$$

Equation 3.39 is similar to Equation 3.5 and provides a way to use the successive summations by cascading them. There are $(1 + 2 + \cdots + (k-1))$ 2-input summations and one $(k-1)$ input summation for Implementation-2 of Figure 3.7. Thus, Implementation-2 has $(k^2 + k - 4)/2$ adders because one $(k-1)$ input summation needs $(k-2)$ adders. In another embodiment we can have only $(k-2)$ 3-input summations and one 2-input summation. Thus, Implementation-3 of Figure 3.8 has $(2k-3)$ adders. In Figure 3.8, there are $(k-1)$ input sequences and the impulse on each input sequence requires a multiplication for each coefficient of the one-sided polynomial and the corresponding $\nabla^j \tilde{b}_j[n - n_i]$. For $j = 1, 2,$ and 3, the multiplications can be done by simply assigning the coefficients at the appropriate positions on the difference sequences except for the $\nabla^3 \tilde{b}_3[2]$'s location (this position needs a shift of two bits of the corresponding coefficient a_3 because $\nabla^3 \tilde{b}_3[2] = 4$). However, for higher order polynomials than 4th order, the required number of multiplications increases by $\boldsymbol{O}(k^2)$[38].

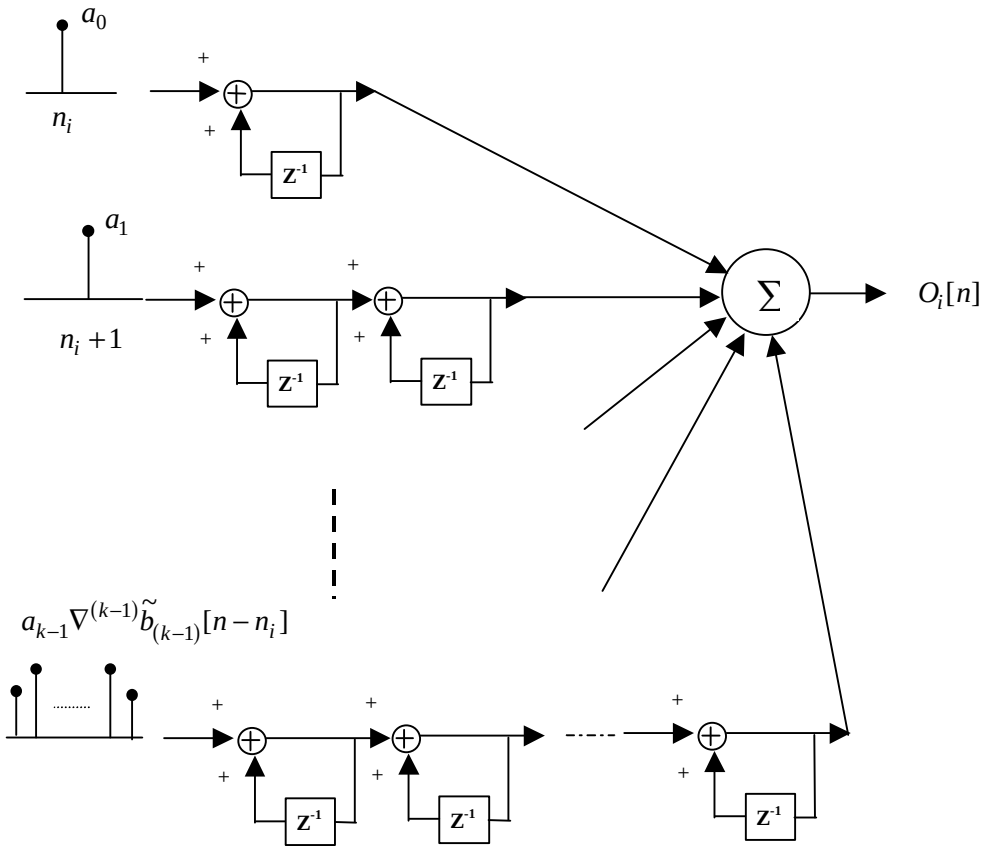


Figure 3.7 Implementation-2: The hardware for constructing a one-sided polynomial

Example 3.1 Let's construct $p[n] = n^4 - 31n^3 + 305n^2 - 1025n + 750$ using Implementation-3 from $n = 0$ to $n = 15$. Each summing junction of Implementation-3 can be considered as the appropriate order difference of $p[n]$. Thus, the first junction from the left end of Implementation-3 should be $\nabla^5 p[n]$, and the remaining junctions should be labeled with $\nabla^4 p[n], \nabla^3 p[n]$, $\nabla^2 p[n]$ and $\nabla p[n]$ from the left, and their values are shown in Figure 3.9. As shown in f) of Figure 3.9, there is no error between the direct evaluation values of $p[n]$ and the hardware constructed values of $p[n]$. As matter of fact, the roots of $p[n]$ are [1,5,10,15] and (f) of Figure 3.9 shows, $p[n]$'s zero crossing exactly at the roots.

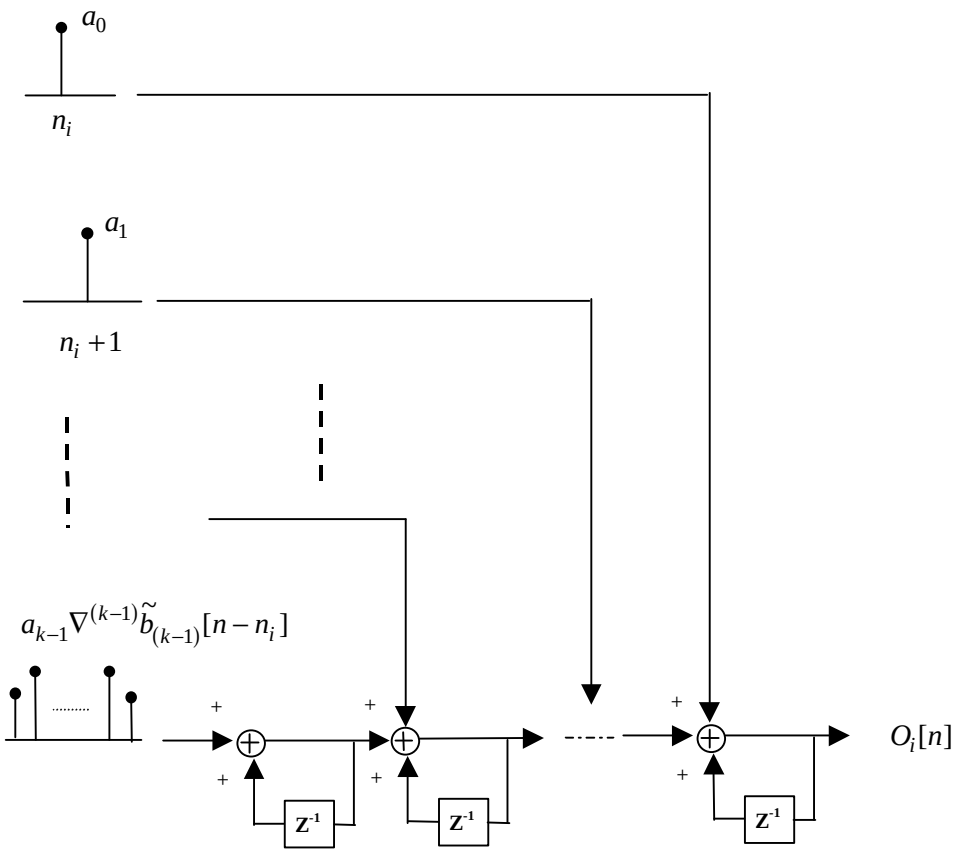


Figure 3.8 Implementation-3: the improved hardware for constructing a one-sided polynomial by cascading the successive summations

The direct evaluation needs at least 10 multiplications (using Horner's rule) and 4 additions for each point. Therefore, a total of 160 multiplications and 64 additions are required for 16 points. The overall complexity (in the case, Horner's rule is used) is approximately $160w^2+64w$*. With Implementation-3, only 11 multiplications and 112 additions are required for 16 points. For this case, the overall complexity is approximately $11w^2+112w$*. Thus, Implementation-3 is a far more efficient implementation than Horner's rule because the difference between the conventional method and the Implementation-3 is about $149w^2-48w$ for this specific case and w is

* w represents the hardware complexity of an adder. Roughly, the hardware complexity of a multiplier is about w^2 and the one of a delay is about w. Thus, the overall complexity measures in the table are representing the complexity very roughly.

always a positive number. Also, Implementation-3 for a SISD machine requires 101 fewer instructions. As the number (ξ) of date points increases, the required number of instructions for the direct evaluation increases by approximately 14ξ (the number of the multiplications and the number of the additions increase proportionally to ξ) and the factor for the hardware increase is about 7ξ (only, the number of the additions is increased as ξ increases). Implementation-3 has a great advantage over the direct evaluation in a SISD machine as well as in custom made hardware.

The $\boldsymbol{O}(k^2)$[38] increase in the number of the multiplication can be reduced by changing the basis functions to one-sided factorial functions. Of course, a new set of coefficients is needed for the new set of basis functions. The transformation operator is discussed in Appendix E. The operator is different from Üstüner's because the definition of the one-sided factorial function in here is slightly different from his. Although the difference between the two is only in scale factors, it eliminates one multiplication at each successive summation and makes it possible to use Implementation-3 using new input sequences. Because each order (k) of the one-sided factorial requires a single impulse on its k difference sequences, the required number of the multiplications increases proportional to order. Also, notice that the starting index is different (ref. Table 3.2).

Example 3.2 Let's construct $p[n] = n^4 - 31n^3 + 305n^2 - 1025n + 750$, again, as in Example 3.1. This time, we will use the one-sided factorial functions. First, the new set of the coefficients a_i''s will be obtain to satisfy $p[n] = a_4' b_4[n] + a_3' b_3[n] + a_2' b_2[n] + a_1' b_1[n] + a_0' b_0[n]$ with the transformation operator shown in Appendix E. Thus, $a_0' = 750$, $a_1' = -750$, $a_2' = 438$, $a_3' = -150$ and $a_4' = 24$. The coefficients are the scale factors for the corresponding input impulses of Implementation-3. The differences of $p[n]$ and $p[n]$ itself are shown in Figure 3.10. The plots d) and e) of Figure 3.9, and the plots d) and e) of Figure 3.10 are identical to each other because $b_0[n] = \tilde{b}_0[n]$ and $b_1[n] = \tilde{b}_1[n]$. The required number of additions for the one-sided factorial bases is the same as for the one-sided power bases, but now only 5 multiplications are required. It requires 6 less multiplications than the computation using the one-sided power bases. In

addition to a smaller number of multiplications, the multiplications can be required by assignment in this case because the impulse magnitudes of $\Delta^k b_k[n]$ are all unity for $k = 0,1,\cdots,\infty$.

In the case that the outside values of the polynomial segment are required to be zero, the strategy shown in Figure 3.1 should be adopted. The implementation is quite simple and there is no difference between the one-sided power function based system and the one-sided factorial function based system. $O_i(t)$ and $O_{i+1}(t)$ of Equation 3.10 are on the same polynomial as shown in Figure 3.1. Using Theorem 3.1 and the transformation given in Appendix B, the coefficients of $O_{i+1}(t)$ can be obtained from $O_i(t)$. Then, super-posed the negative of $O_{i+1}(t)$'s coefficient weighted impulses with $O_i(t)$'s coefficient weighted impulses, and feed in the impulses to the corresponding inputs of Implementation-3. By nature of the segmentation, its implementation is identical with the set of polynomial segments'.

There is another efficient method, which is used to construct a polynomial. It is called the forward difference method because it uses the relationship between the differences of a polynomial and their estimates[39]. For clarification, a 4th order polynomial will be used in the explanation. From Equation 3.1 and 3.2,

$$\begin{aligned} & \Delta P(t) = P(t+\mathrm{d}t) - P(t) \\ \Rightarrow \quad & P(t+\mathrm{d}t) = P(t) + \Delta P(t) \\ \Rightarrow \quad & P[n+1] = P[n] + \Delta P[n] \end{aligned} \tag{3.40}$$

Since $P(t) = \sum_{i=0}^{3} a_i t^i$, $\Delta P[n]$ can be estimated as

$$\begin{aligned} \Delta P[n] &= \sum_{i=1}^{3} a_i (t+\mathrm{d}t)^i - \sum_{i=1}^{3} a_i t^i \\ &= 3a_3 t^2 \mathrm{d}t + t(a_3 \mathrm{d}t^2 + 2a_2 \mathrm{d}t) + a_3 \mathrm{d}t^3 + a_2 \mathrm{d}t^2 + a_1 \mathrm{d}t \end{aligned} \tag{3.41}$$

Similarly,

$$\Delta^2 P[n] = 6a_3 t\mathrm{d}t^2 + 6a_3\mathrm{d}t^3 + 2a_2\mathrm{d}t^2$$
$$\Delta^3 P[n] = 6a_3\mathrm{d}t^3 \tag{3.42}$$

From Equation 3.41 and 3.41, an initial set of $P[n_0]$, $\Delta P[n_0]$, $\Delta^2 P[n_0]$ and $\Delta^3 P[n_0]$ can be obtained by setting t at a set of initial points. For convenience, let the first point be zero with index 0 ($n = 0$). Then, the initial values are obtained using

$$\begin{bmatrix} P[0] \\ \Delta P[0] \\ \Delta^2 P[0] \\ \Delta^3 P[0] \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ \mathrm{d}t^3 & \mathrm{d}t^2 & \mathrm{d}t & 0 \\ 6\mathrm{d}t^3 & 2\mathrm{d}t^2 & 0 & 0 \\ 6\mathrm{d}t^3 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \tag{3.43}$$

A recursive matrix equation can be formed from the definition of the forward difference as

$$\begin{bmatrix} P[n+1] \\ \Delta P[n+1] \\ \Delta^2 P[n+1] \\ \Delta^3 P[n+1] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P[n] \\ \Delta P[n] \\ \Delta^2 P[n] \\ \Delta^3 P[n] \end{bmatrix} \tag{3.44}$$

and used to construct $P[n]$. This method requires one less addition than the method shown in Example 3.2, but requires approximately $k(k-1)/2$ initial multiplications. Thus, its performance is almost same as the method used in Example 3.1.

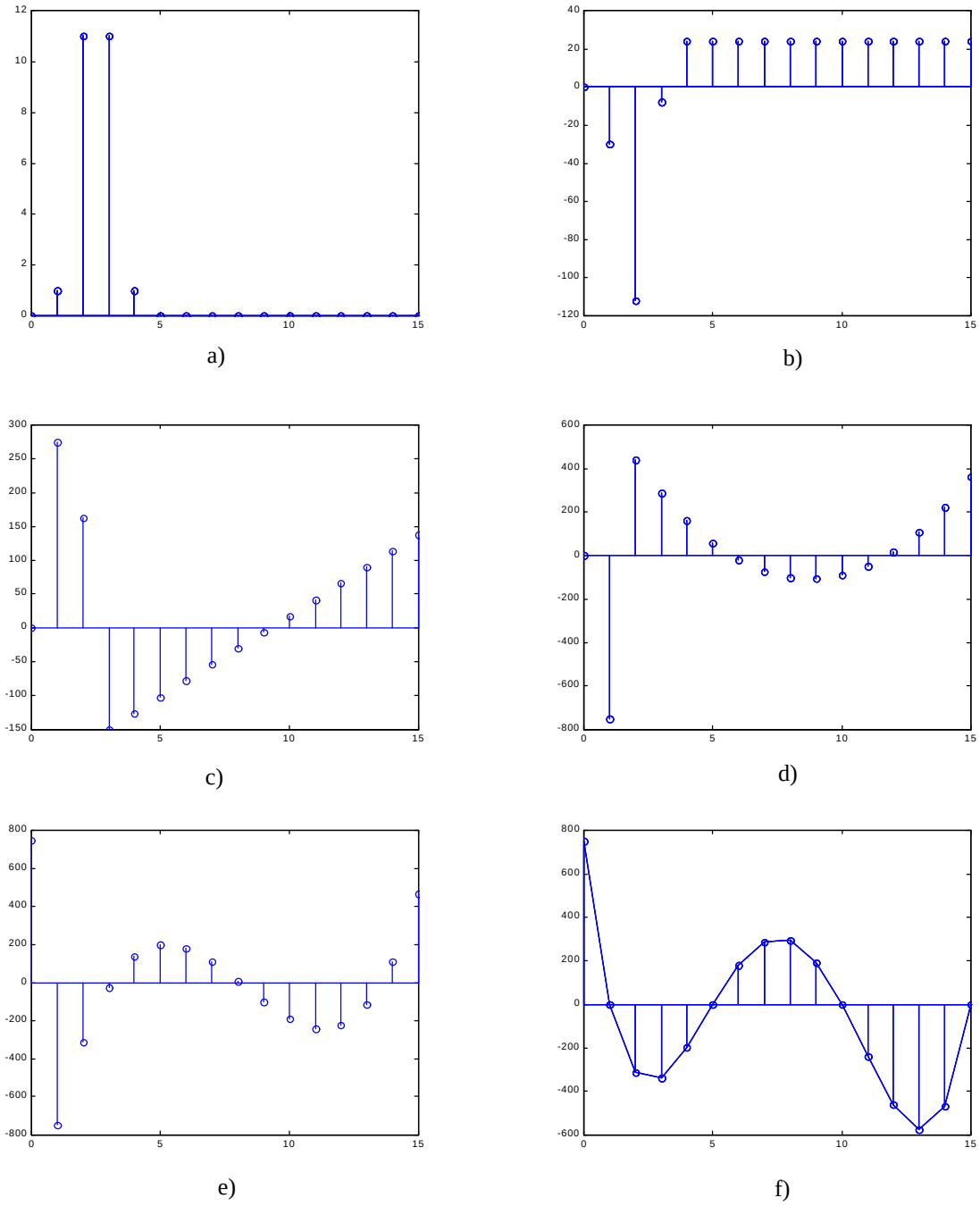


Figure 3.9 Backward differences of $p[n]$ and $p[n]$: a) $\nabla^5 p[n]$, b) $\nabla^4 p[n]$, c) $\nabla^3 p[n]$, d) $\nabla^2 p[n]$, e) $\nabla p[n]$ and f) $p[n]$ from the hardware configuration using one-sided power bases and solid connecting line is from the direct calculation of $p[n]$.

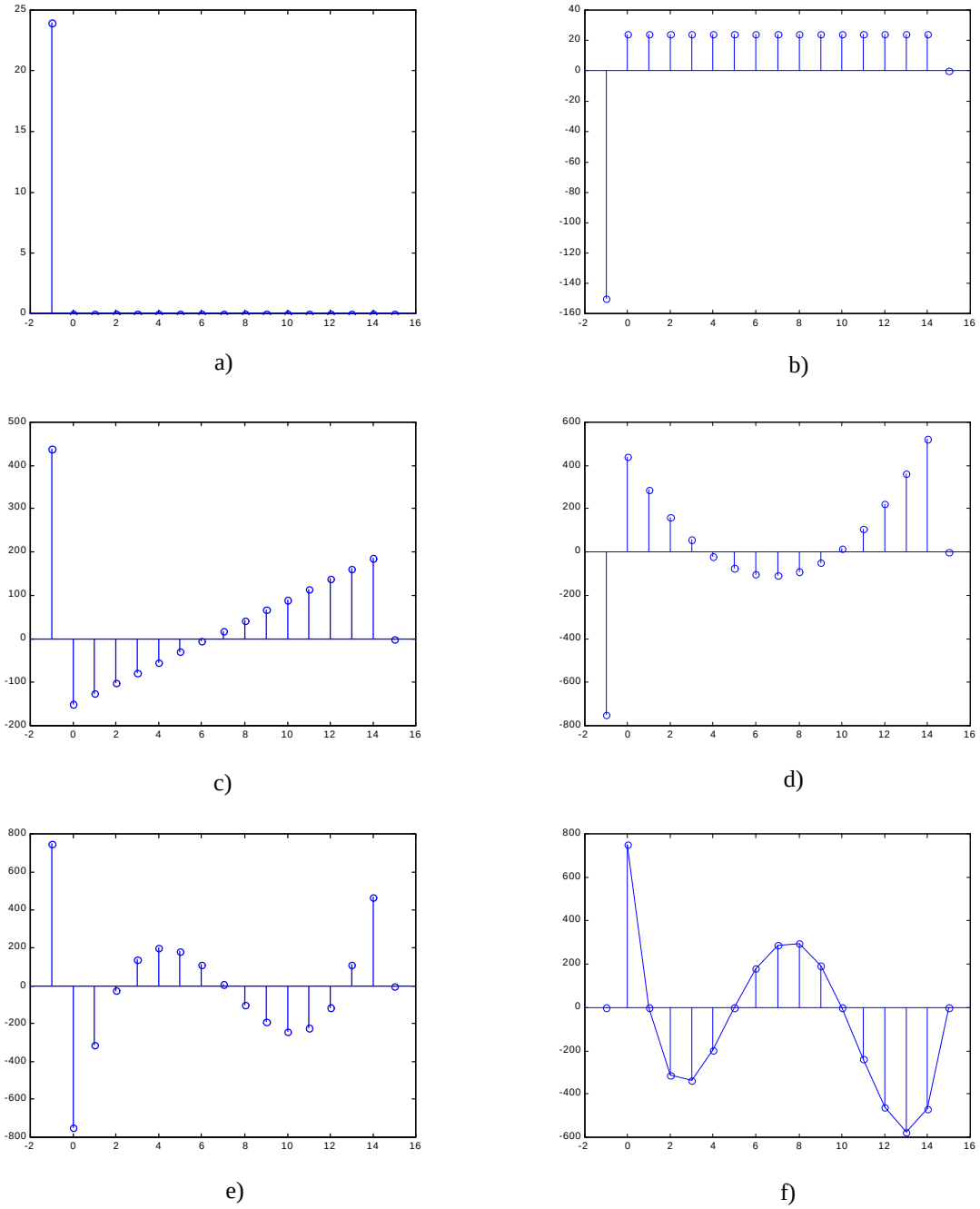


Figure 3.10 Forward differences of $p[n]$ and $p[n]$: a) $\nabla^5 p[n]$, b) $\nabla^4 p[n]$, c) $\nabla^3 p[n]$, d) $\nabla^2 p[n]$, e) $\nabla p[n]$ and f) $p[n]$ from the hardware configuration using one-sided factorial bases and solid connecting line is from the direct calculation of $p[n]$.

3.5.2 B-spline

Since Schoenberg developed the B-spline[1,40] several numerical methods to evaluate B-splines have been developed. Among them, only three methods are currently considered to be efficient and numerically accurate. The first method, based on the recursive relationship described in Equation 3.16, is developed by Cox[29] and de Boor[30]. The second one is the Forward Difference method, which utilizes the relationship between the forward difference of a polynomial and their estimations from the continuous version of the polynomial as described in the previous section[41]. The last one is the impulse/summation approach. The performance comparison study result[33] among these three methods is shown in Table 3.4. It shows the advantage of the impulse/summation approach over others. Even though some number of multiplications and divisions appears for the impulse/summation entries in the table, with proper scaling factors, they can all be eliminated. The implementation for the forward difference requires a fresh start at each knot location because a new set of coefficients is needed at these locations to restart the recursive equation. A major advantage of the impulse/summation approach is that it does not require refreshing.

	Any n, m			$n = 100, m = 10$	
	k	+ and -	× and /	+ and -	× and /
de Boor's Method	3	$9n + 2m - 3$	$9n$	917	900
	4	$18n + 3m - 6$	$18n$	1824	1800
Forward Difference	3	$2n + 8m - 6$	$5m - 3$	274	47
	4	$3n + 15m - 12$	$10m - 6$	438	94
Impulse/Summation	3	$2n + 7m - 6$	$2m - 3^*$	264	17
	4	$3n + 10m - 12$	$3m - 6^*$	388	24

* They can be zeros with proper scaling factors.

Table 3.4 Calculation requirements for the direct spline evaluations[33]: k is the order, m is the number of the knots and n is the number of sampled points

To use the impulse/summation technique, the coefficients of the one-sided power functions for the B-spline must be obtained. A good algorithm for converting B-spline vertices into the one-sided power function based system is given in Sankar, *at al*[33]. It utilizes the relationship between the k^{th} derivative of the B-spline $q(t)$ and the set of its one-sided power functions. It can be shown that

$$q^{[k]}(t) = \sum_{i=1}^{m} a_i d(t - t_i) \tag{3.45}$$

where $q^{[k]}(t)$ is the k^{th} derivative of $q(t)$. See Sankar, *at al*[33] for detail.

After obtaining the coefficients for the B-spline from the knots and vertices, the B-spline can be handled as a summation of weighted one-sided power functions, which have non-zero values after their knots and the same orders. Thus, a discrete version of a B-spline ($q(t)$ in Equation 3.15) can be

$$\begin{aligned} q[n] &= \sum_{i=1}^{m-1} a_i (n - n_i)_+^{k-1} \\ &= \sum_{i=1}^{m-1} a_i \tilde{b}_k [n - n_i] \end{aligned} \tag{3.46}$$

Thus,

$$\nabla^k q[n] = \sum_{i=1}^{m-1} a_i \nabla^k \tilde{b}_k [n - n_i] \tag{3.47}$$

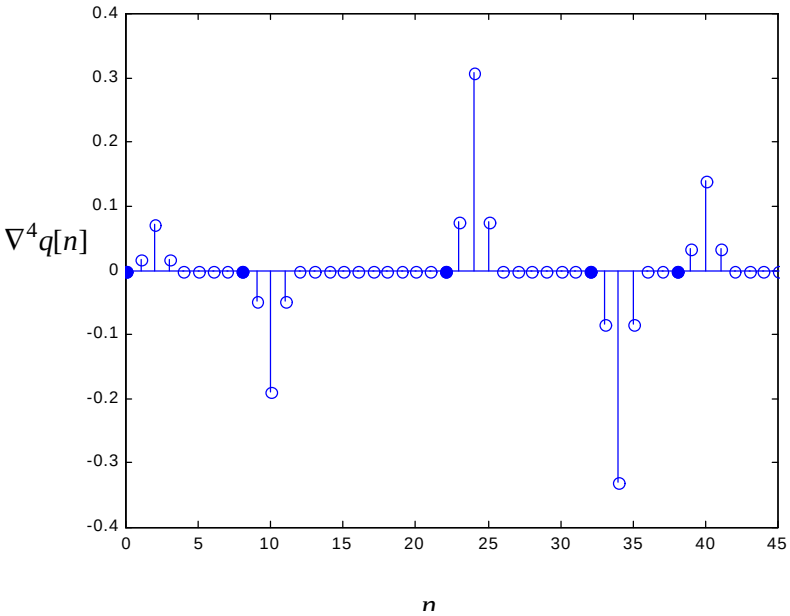
The homogeneity of order for every segment make it possible that the generation of a B-spline needs only a single input. The implementation will be much simpler than for a polynomial and there will be no 3-input adders. The implementation will be the same as Implementation-1. The input should be $\nabla^k \tilde{b}_k[n]$ instead of a single impulse and the 4^{th}

order example is shown in Figure 3.11. The constructed $q[n]$ with this implementation is also shown in Figure 3.11. The required number of the multiplications will be approximately $(m-1)(k-1)$ (when zero-values are required at the outside boundary, $m(k-1)$). The number of additions is kx where x is the number of sample points.

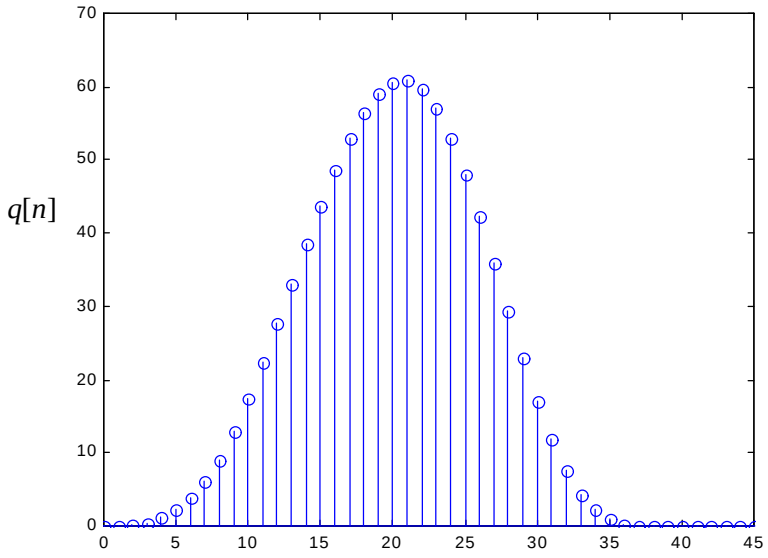
It is possible that the generation of a B-spline can be achieved using the one-sided factorial functions. The implementation will be identical to Implementation-3. However, it will require more additions for each sample point because it needs 3-input additions at every successive summation junction and one additional multiplication at the knot locations (where the multiplication by unity is considered as a multiplication). As k increases, the advantage in the number of multiplications is reduced, but the number of additions is increased. Also, due to the higher order splines' oscillation tendencies, the cubic B-spline is usually used in most practical applications. Thus, the implementation with the one-sided power function bases is better in most applications.

3.5.3 Polynomial Splines and Set of Polynomial Segments

The B-splines can be considered as a subset of the polynomial splines because the B-splines can be written with one-sided power functions while a polynomial spline can be written with both one-sided power functions and power functions. However, a polynomial spline can be considered as a combination of a polynomial and the same ordered B-spline by the same reasoning. Therefore, again, Implementation-3 can generate a polynomial spline if we allow a change in inputs. The first input, at the left end of the successive summations in Figure 3.8, is the B-spline component. The remaining inputs take care of the polynomial component. Because it is the same as that given in Section 3.5.1, the performance analysis is not repeated in here. Also, because of its oscillatory tendency, the 4^{th} order polynomial spline is usually implemented as a B-spline. The polynomial segments implementation is the same as the one used for polynomial splines.



a)



b)

Figure 3.11 a) An example of $\nabla^4 q[n]$ and b) Constructed $q[n]$, which has 5 knots at $n = 0$, $n = 8$, $n = 22$, $n = 32$ and $n = 38$ (marked by filled dots) and $a_1 = 0.017756$, $a_2 = -0.04712$, $a_3 = 0.07711$, $a_4 = -0.08247$ and $a_5 = 0.034722$

Chapter 4

Shape and Shading

4.1 Introduction

This chapter demonstrates the advantages in using the impulse/summation approach for smooth surface generation and a new approach to shading in computer graphics. Thus, the chapter discusses 2-D B-spline construction and partial difference construction. Thus far, there have been several studies in smooth surface generation using the impulse/summation approach (especially with 2-D or 3-D B-splines). Most of the discussions and the methods of generating the surface examples in Section 4.3 come from Wang[34] and Üstüner[21]. A new finding, in which the impulse/summation approach produces partial differences which are required for shading (during the surface generation without additional calculation) is a new contribution of this dissertation (see Section 4.4). The size of the shading task is extensively reduced. Another new contribution in this chapter is the finding that a small increase in the number of patches using Lambertian shading can achieve a comparable shading result to the more complex Gouraud shading scheme without a significant increment in computational cost (see Section 4.4.3).

4.2 3-D Computer Graphics

Computer graphics is one of the most popular applications in which polynomials and splines are used since they can model a curved line or a smoothly curved surface with a small number of parameters. Computer graphics involves two interrelated processes, shaping and shading. Shaping determines the parts of the 3-D model (or 3-D object) that

appeared in the 2D-image and, if they appear, the locations to be rendered in the image. The model's 3-D geometric description, view point (viewing direction), and projection scheme[42] are required. Shading determines the brightness of the patches at the given locations (if it is for color graphics, the color and the intensity of the color on the patches). The process requires the direction and the intensity of the illumination, the surface properties of the patches[43] and the surface normal directions of the patches. See Figure 4.1 for an example.

The surface normal directions of a 3-D object are very important in both shaping and shading. For shaping, the surface direction and the viewpoint can determine whether its patch faces away from the viewpoint, or not. For shading, the surface normal and the illumination direction determine the brightness of the patch in the 2-D image. Of course, the geometric information is as important as the surface normal because, not only can the appearance of the object in the 2-D image be determined, but, in addition, the surface normal can also be obtained. Let's first discuss the geometric description of the surface.

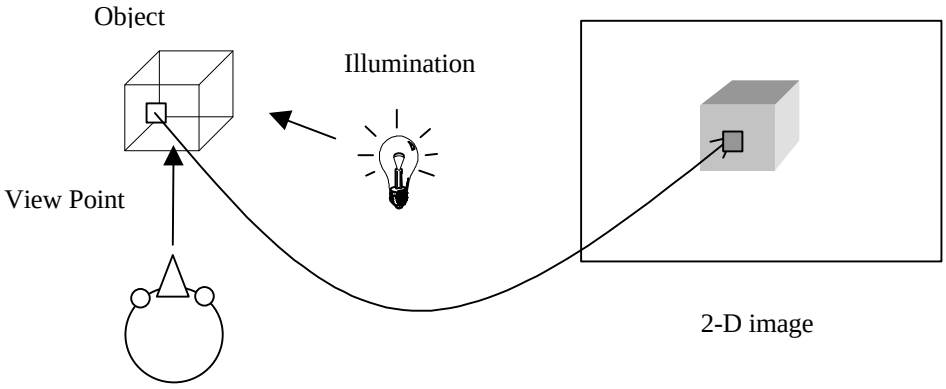


Figure 4.1 A computer graphics process diagram

4.3 Surface

Since all the points on a surface must be defined in a 3-dimensional space, the simplest way is to let the surface take on a set of "heights" for every point on a plane. The mathematical expression is

$$Surface = \{S | S(x, y) \in z\} \tag{4.1}$$

on a 3-D Cartesian co-ordinates. See Figure 4.2 for an example. With the above equation, it is impossible to represent a concave surface along the z-axis. One more parameter is required for the concave surface. Thus, the model may not be used for general computer graphics. However, the model in Equation 4.1 is good enough to use in the demonstration of the theory. Because the polynomial, B-spline, polynomial spline and polynomial segment can be used for surface generation and the differences among them are small (other than the dimensionality and the issues discussed in Chapter 3) only the B-spline is used in this chapter.

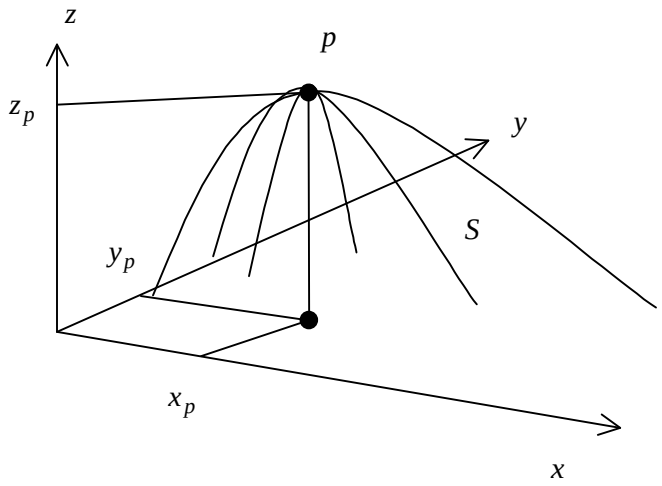


Figure 4.2 A surface example, where $p = [x_p, y_p, z_p]$ is on a surface S.

Let's define a k^{th} order 2-D B-spline as

$$q(x,y)=\sum_{i=1}^{m_x}\sum_{j=1}^{m_y}V_{i,j}B_{i,k}(x)B_{j,k}(y) \tag{4.2}$$

and consider it as a surface model. Equation 4.2 can be rewritten with one-sided power functions and the 2-dimensional partial derivatives can be written as

$$q^{[k,k]}(x,y)=\sum_{i=1}^{m_x}\sum_{j=1}^{m_y}a_{i,j}d(x-x_i)d(y-y_j) \tag{4.3}$$

where

$$f^{[n,m]}(x,y)=\frac{\partial^n}{\partial x^n}\frac{\partial^m}{\partial y^m}(f(x,y)).$$

We can then apply the impulse/summation technique. The 4th order discrete version of Equation 4.3 is

$$\nabla_{n_x}^4\nabla_{n_y}^4 q[n_x,n_y]=\sum_{i=1}^{m_x}\sum_{j=1}^{m_y}a_{i,j}\nabla_x^4\tilde{b}_4[n_x-n_{x_i}]\nabla_y^4\tilde{b}_4[n_y-n_{y_j}]^T \tag{4.4}$$

where

$$\begin{aligned}\nabla_n\nabla_m f[n,m]&=\nabla_m f[n,m]-\nabla_m f[n-1,m]\\&=f[n,m]-f[n,m-1]-f[n-1,m]+f[n-1,m-1]\end{aligned}$$

The above equation is nothing more than a 2-dimensional expansion of Equation 3.44. Thus, the construction of the surface, defined by Equation 4.4, can also be achieved by applying 2-dimensional repeated summations. Because Equation 4.4 forms a $\left[m_x\times m_y\right]$ matrix with 2-dimensional impulses, the 2-dimensional successive summation operation can be executed by first doing a successive summation along the columns and then repeating the processes with the rows. k executions of the 2-dimensional successive

summation operation will result in a k^{th}-order 2-dimensional repeated summation operation. Because a k^{th}-order 2-dimensional repeated summation operation can be separated into two 1-dimensional k^{th} repeated summations, Implementation-1 can be used for constructing the 2-D B-spline. As in the 1-dimensional case, the coefficient relationship between Equation 4.2 and 4.3 can be obtained from the k^{th} 2-dimensional derivative of the 2-D B-spline. An efficient algorithm to obtain relevant coefficients for Equation 4.3 from Equation 4.2 is developed by Sankar, *et al*[33].

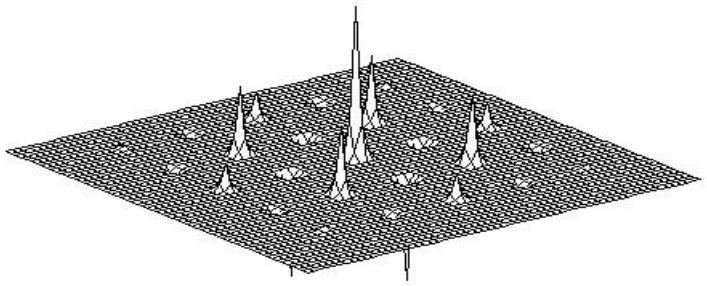
Example 4.1 Let's construct a surface expressed with a 2-D B-spline basis function where $B_{i,4}[n_x]$ and $B_{j,4}[n_y]$ are defined on knots [10 20 30 40 50]. Since each function is defined on even spacing and 4$^{\text{th}}$ ordered, the coefficients for each are $a_1 = 1$, $a_2 = -4$, $a_3 = 6$, $a_4 = -4$ and $a_5 = 1$. Thus, the $a_{i,j}$'s form the matrix

$$\begin{bmatrix} 1 & -4 & 6 & -4 & 1 \\ -4 & 16 & -24 & 16 & -4 \\ 6 & -24 & 36 & -24 & 6 \\ -4 & 16 & -24 & 16 & -4 \\ 1 & -4 & 6 & -4 & 1 \end{bmatrix} \tag{4.5}$$

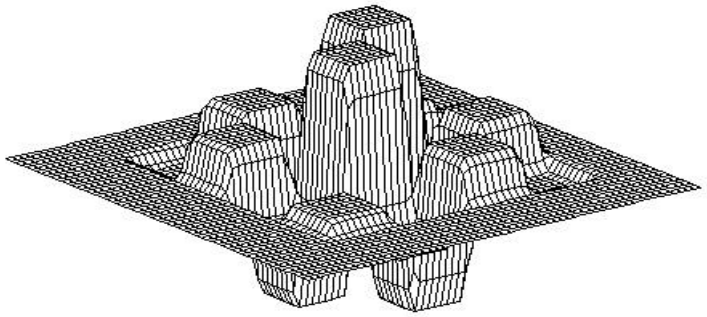
Therefore, $\nabla^4_{n_x}\nabla^4_{n_y} q[n_x, n_y]$ can be shown in a) of Figure 4.3. In the figure, it does not show the detail pattern of each peak. Since $\nabla^4_x \tilde{b}_4[n_x - n_{x_j}]\nabla^4_y \tilde{b}_4[n_y - n_{y_j}]^T$ forms the patch

$$\nabla^4_x \tilde{b}_4[n_x - n_{x_j}]\nabla^4_y \tilde{b}_4[n_y - n_{y_j}]^T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 4 & 16 & 4 \\ 0 & 1 & 4 & 1 \end{bmatrix} \tag{4.6}$$

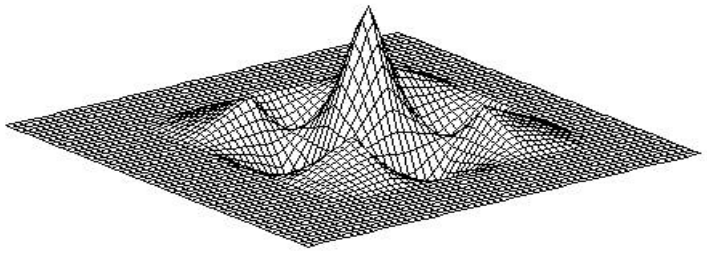
at the knot location (n_{x_i}, n_{y_j}), each peak contains the pattern of the above patch, and each value of the patch is weighted with the corresponding $a_{i,j}$. Figure 4.3 shows the construction result and its higher order differences.



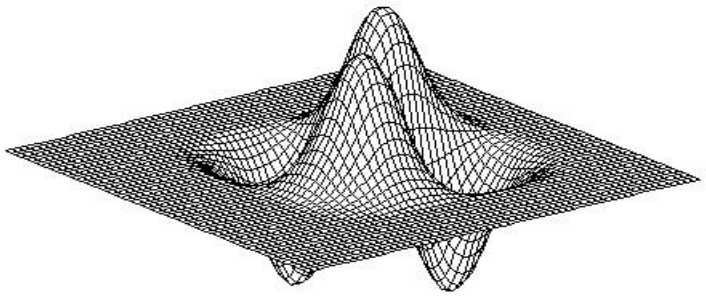
a)



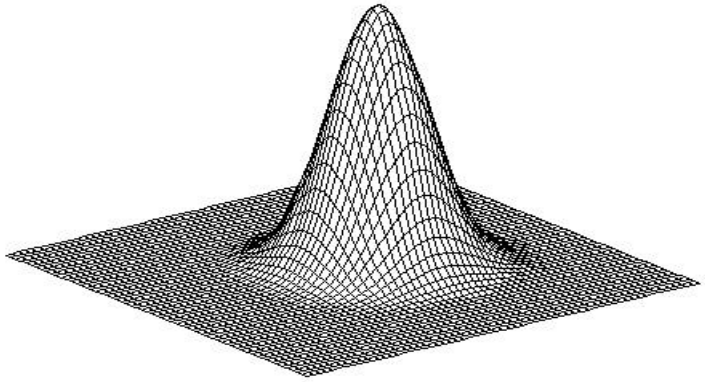
b)



c)



d)



e)

Figure 4.3 The construction of a 2-D B-spline in Example 4.1 and its 2-dimensional repeated summation result; a) the 2-dimensional 4 differences of the B-spline, b) the first 2-dimensional repeated summation result, c) the 2^{nd}, d) the 3^{rd} ,and e) the constructed B-spline

The computation analysis for a 2-dimensional B-spline with de Boor's Recurrence method, forward difference and the impulse/summation is similar to the one for a curve (1-dimensional B-spline). de Boor's algorithm requires $\mathbf{O}(n_x n_y)$ multiplications and additions. The forward difference algorithm requires $\mathbf{O}(m_x m_y)$ multiplications and

$\mathbf{O}(n_x n_y)$ additions. The impulse/summation approach requires the same big-O[38] of multiplications and additions as the forward difference algorithm but a significantly lower number of actual multiplications and additions[33]. The higher the dimensionality the more significant the improvement in complexity of the impulse/summation approach.

4.4 Surface Normal

Obtaining the surface normal of a surface in a continuous domain is simple. At each point on the surface, a normalized vector, which is perpendicular to the tangent plane at the point, is the surface normal. The expression for the surface normal $\vec{N}(x,y)$ on a Cartesian co-ordinate system (see Figure 4.2 for notations and an example) can be given as

$$\vec{N}(x,y) = \frac{\left[-\frac{\partial S}{\partial x}\vec{x}, -\frac{\partial S}{\partial y}\vec{y}, \vec{z}\right]}{\sqrt{\left(\frac{\partial S}{\partial x}\right)^2 + \left(\frac{\partial S}{\partial y}\right)^2 + 1}} \tag{4.7}$$

where $\vec{x}$, $\vec{y}$ and $\vec{z}$ are the unit vectors along with each axis in Figure 4.2. See Appendix F for the detail derivation of Equation 4.7.

We see in Equation 4.7, that obtaining the partial derivatives along the x-axis and the y-axis at every point on S is required to calculate the surface normal. In a discrete domain, there are several ways to obtain an estimate of these partial derivatives. Backward difference and forward difference along an axis are both suitable candidates for the estimation, of the derivatives of a single variable continuous function. Let's call the differences along a single axis the 'partial difference' and use the mathematical notation shown in Equation 4.4. However, the partial differences do not cover every point on the surface as the partial derivatives can. In the case of the example in Section 4.2, the sampled points form quadrangles on the surface. It is necessary to find a way to assign

the surface normal (at least, the shading value) to the mid-point on the surface between the sampled points. Because a quadrangle is a polygon, the shading techniques for polygons may be used for this problem. There are three well known techniques, Lambertian shading[44], Gouraud shading[45] and Phong shading[46].

4.4.1 Lambertian Shading

Lambertian shading uses the same surface normal for all points on the polygon. Using the same normal for every point on the planar polygon surface forces every point on the polygon to have the same shaded value. Thus, the entire polygon looks flat. This technique is called flat shading in some articles because of its appearance. If the sampled points are close enough, it can be assumed that every point on the quadrangles, formed by the sampled points, is on the same plane. The assumption is valid because, unlike the polygon model of 3-D graphics, a polynomial surface can be implemented with a sufficient number of data points that the surface appears naturally curved. Also, the assumption is made that the partial difference estimation of partial derivatives are suitable for Lambertian shading because the partial differences of the surface $S[n_x, n_y]$ will be exactly matched to the slope of each quadrangle along each axis. In Figure 4.4, the straight lines between two adjacent sampled points presents the slopes of the quadrangles between two adjacent sampled points along n_x. When the forward partial difference $\Delta_{n_x} S[i, n_y]$ is used for estimating the partial derivatives at $n_x = i$, the difference will be the slope of the quadrangle, which is located following the point i, because $\Delta_{n_x} S[i, n_y] = S[i+1, n_y] - S[i, n_y]$. When backward differences are used, the slope will be for the one preceding i.

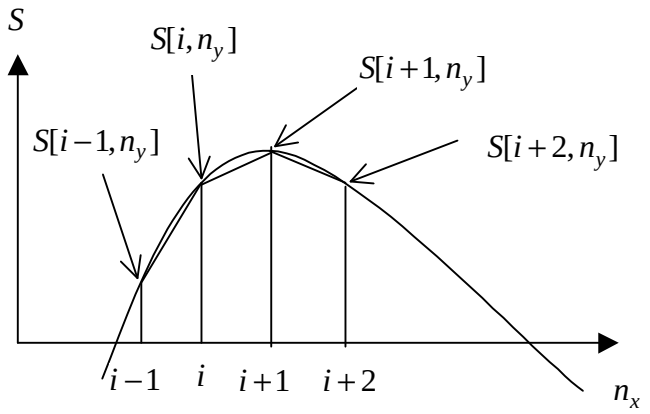


Figure 4.4 The cross-sectional view of the discrete surface from n_y-axis at fixed n_y

Here, the impulse/summation technique provides a great advantage. During surface construction with this technique the partial differences of the surface along an axis are automatically obtained. Only one direction's partial differences are need for the surface normal. For a cubic B-spline model of the surface, they can be obtained by either the successive summation of $\nabla_{n_x}\nabla_{n_y}S[n_x,n_y]$ or the backward differences of the constructed surface $S[n_x,n_y]$.

From Equation 4.7, the discrete version of the surface normal can be rewritten as

$$\vec{N}[n_x,n_y]=\frac{\left[-\nabla_{n_x}S\cdot\vec{n}_x,-\nabla_{n_y}S\cdot\vec{n}_y,\vec{z}\right]}{\sqrt{\left(\nabla_{n_x}S\right)^2+\left(\nabla_{n_y}S\right)^2+1}} \tag{4.8}$$

In Equation 4.8, square root operation ($\sqrt{\cdot}$) consumes the most time in obtaining the surface normal. However, because the sums of the squares of the partial differences are in a limited range of integers, a square root operation table may be constructed to reduce the computational load. If a reciprocal table of the square root is used instead of the square root, the division in the Equation 4.8 can also be eliminated. The time needed to find a function value from a table depends on the table length l. Since it takes a search algorithm, the number of the comparison operations may take $\log_2 l$ with a binary search

algorithm, in the worst case[47]. Operation for calculating a surface normal takes about $\log_2 l$ comparisons, 5 multiplications and 2 additions per sample point on the surface. Therefore, for computing the overall surface with the surface normal, the number of calculations increases linearly with respect to the number of sample points.

In addition to the surface normal, a light source description is required to calculate shading. The light source description includes how many light sources, each sources' light patterns, their intensities and their directions. The illumination effect at a single point on surface or a face of a polygon will be

$$\vec{L} = \sum_{i=1}^{M} \vec{L}_i \tag{4.9}$$

where, $\vec{L}_i$'s are lightening effects on the point from each light source and consists of direction and intensity. The shading value I on the point is determined by the scalar product as,

$$I[n_x, n_y] = \text{cut}\left(-\vec{L}[n_x, n_y] \cdot \vec{N}[n_x, n_y]\right) \tag{4.10}$$

where,

$$\text{cut}(x) = \begin{cases} x & \textit{for } 0 \le x \\ 0 & \textit{otherwise} \end{cases}.$$

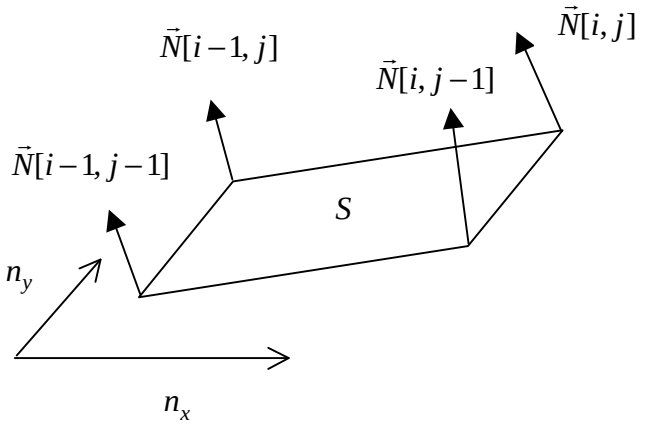


Figure 4.5 A quadrangle surface patch on the model in Example 4.1

Example 4.2 Let's shade the surface model in Example 4.1 using Lambertian Shading. Because the backward differences are used for partial differences, $\vec{N}[i,j]$ represents the surface normal of the quadrangle patch in Figure 4.5. Let's assume a single light source is used, and located far enough away from the model so that every part of the surface receives the same light intensity and same light direction for simplifying Equation 4.10. The shading value, $I[n_x, n_y]$ is

$$I[n_x, n_y] = \text{cut}\left(\frac{\nabla_{n_x} S \cdot \vec{L}_x + \nabla_{n_y} S \cdot \vec{L}_y - \vec{L}_z}{\sqrt{\left(\nabla_{n_x} S\right)^2 + \left(\nabla_{n_y} S\right)^2 + 1}}\right) \tag{4.11}$$

where, $\vec{L} = \left[\vec{L}_x, \vec{L}_y, \vec{L}_z\right]$. The Lambertian shading of Example 4.1 is shown in Figure 4.6 for $\vec{L} = [1,1,1]$. As expected, some parts of the shading of Figure 4.6 change abruptly at the edges of the quadrangle patches. This artifact may disappear when the number of the sample points on the surface increases. Increasing the number of the sample points has two positive effects on the shading quality. It makes the size of the quadrangle patches smaller and the partial differences are closer in value to the actual partial derivatives in the surface normal calculation.

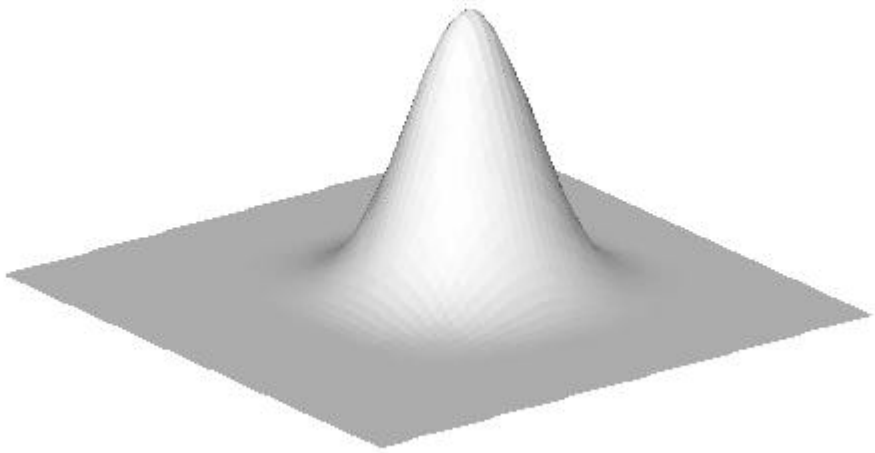


Figure 4.6 The Lambertian shading of the B-spline surface model in Example 4.1

4.4.2 Interpolated Shading

For Gouraud shading and Phong shading, the surface normals at ever corner of each quadrangle patch are required (see Figure 4.5). Gouraud shading obtains the shading values at all four corner of the patch from the surface normal at the corners. Then, it interpolates the shading values at all locations on each polygon bi-linearly using values from the four-corners. Phong shading first interpolates the surface normal for every point on the patch, and then evaluates the shading values from these surface normal values at each point on the patch. Thus, Phong shading produces much better quality shading but takes more computation. However, they have a common need. Both approaches need the surface normal at every sampled point on the surface. Also, they generate more accurate shading, as the surface normals are closer in value to their actual continuous domain counterparts. Fortunately, there is a simple way to compute these from the polynomial surface models.

When a surface model is built with B-splines, a set of polynomial segments, or polynomial splines, the basis functions of the model are always designed to be separable along the axes, even with complex three parameter models. Thus, a systematic approach to obtaining the analytic derivatives of the bases along one axis exists and the analytic partial derivatives can be found. For example, consider a B-spline surface model. With Equation 4.3 the new basis functions of the model can be written as

$$(x - x_i)_+^{k-1}(y - y_i)_+^{k-1} \tag{4.12}$$

Since the first derivative of $(x - x_i)_+^{k-1}$ is

$$(k-1)(x - x_i)_+^{k-2} \tag{4.13}$$

the partial derivative of Equation 4.12 is $(k-1)(x - x_i)_+^{k-2}(y - y_i)_+^{k-1}$ or $(x - x_i)_+^{k-1}(k-1)(y - y_i)_+^{k-2}$ for $k > 0$. The partial derivatives on the sample point grids can be obtained without error in the same manner, in which the surface was constructed. Of course, the process requires additional computation. Also, in the case of the set of polynomial segments surface model, the same idea can be used because the first derivative of a k^{th} order polynomial (Equation 3.1) is

$$P'(t) = \sum_{i=1}^{k-1} i a_i t^{i-1} \tag{4.14}$$

However, for both the B-spline model and a set of polynomial segments model, the process of obtaining the partial derivatives requires almost the same amount of computation for both. Equation 4.13 is only another one-sided power function and Equation 4.14 is just another polynomial with one order less than the original polynomials.

Theorem 4.1 When $f[n] = f(n\Delta t)$, i) the center difference $\Diamond f[n] = (f[n+1] - f[n-1])/2$ is bounded by the forward differences and the backward differences of $f[n]$, where $f(t)$

is a continuous smooth function and $\Delta t = 1$. ii) The values of the center difference of $f[n]$ are closer to $f'(n)$ either the forward or backward differences. See Appendix G for proofs.

Theorem 4.1 says that, the partial center difference values allow the estimation of the partial derivatives on the sampled points with less error than either the forward or backward differences. The center differences may be obtained by applying the center difference operation to the constructed surface or averaging two adjacent partial forward (or backward) differences along the partial difference axis. Either method requires a division by 2 for every sample point. Therefore, it is much faster to use averaging two adjacent partial forward (or backward) differences because they are automatically obtained during the surface construction using the impulse/summation approach and we do not have to apply an additional center difference operator at each sample point. For an integer processor, division by 2 can be implemented by one right shift with a possible half bit error (because right shift throws away the lowest order bit of the binary number for executing division by 2 and the thrown away bit can be used for rounding up the number). A half bit error may not significantly affect to the quality of the surface normal.

Example 4.3 Let's shade the surface model in Example 4.1 with Gouraud Shading. The center difference is used for the partial difference at every sample point. The same light model used in Example 4.2 is effective for this example, too. The shading result is shown in Figure 4.7.

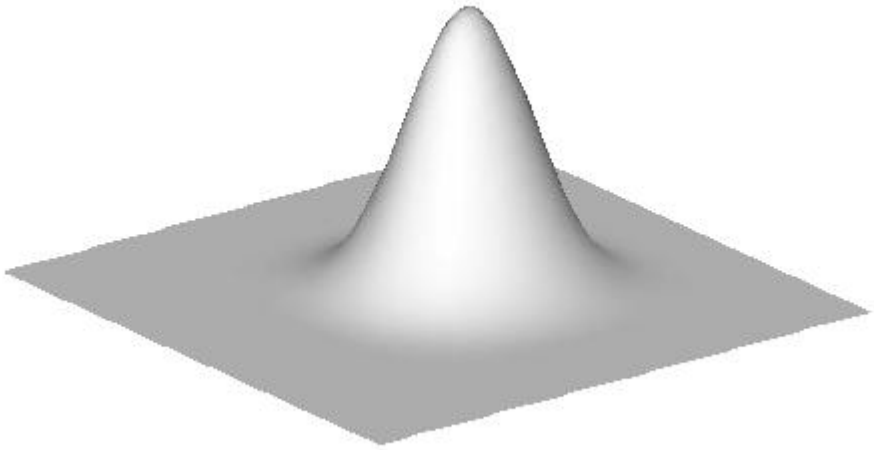


Figure 4.7 The Gouraud shading of the B-spline surface model in Example 4.1

4.4.3 Improving the Quality of Shading

It is visually obviously that the shading quality of Figure 4.7 is much better than that of Figure 4.6. This quality improvement is accompanied by higher computational cost. As was discussed previously, Gouraud shading uses bi-linear interpolation for each pixel of the shaded surface in the constructed image, while Lambertian shading assigns shading values to each pixel. When the same size image is formed with Lambertian shading and Gouraud shading, the numbers of pixels in the both images are the same. The bi-linear interpolation per pixel requires extra computation even using the same partial difference estimation scheme. Thus, if the number of pixels is much more than the number of patches in the model (usually, this is true in computer graphics), the difference in computation between Gouraud shading and Lambertian shading is enormous. From this, one may conclude that it is better from the computational complexity standpoint to increase the number of patches in the Lambertian shading rather than using Gouraud shading to obtain better quality shading.

To increase the number of patches on the model, requires decreasing the sampling period and padding zeros between the impulse patterns in the 4th derivative for the cubic B-spline model. In addition, scaling is required to keep the original geometry of the model and the partial differences. Assume the sampling period is cut in half and a cubic surface model is used. To generate the surface, the 4th repeated summation operator is needed for each axis. Applying the 4th repeated summation operator to a single impulse for both the original sample point and the half-period sample point generates the same shaped curves. But, the curve in the half-period sample case is 8 times larger because the physical points are not changed but the index number (variable) is doubled. For example, consider n^3. If the sampling period is doubled, the new index n_0 has the relationship with the old index as $n = 2n_0$. However, by applying the 4th repeated summation operator on n_0, n_0^3 is created instead of n^3. Thus, at every point on $n = 2n_0$, the function n_0^3 is always 2^3 times larger than the function n^3. This scaling factor can be generalized by

$$scaling\ \ factor = m_0^{k-1} \tag{4.15}$$

Where m_0 is the number of sampled point increase per single period and k is the order of the repeated summation operation used. When it is expanded into an R-dimensional space ($R = 2$, for the examples of this chapter), that

$$R - dimensional\ \ scaling\ \ factor = \left(m_0^{k-1}\right)^R \tag{4.16}$$

As it can be noted, the scaling factor increases rapidly with respect to R and m_0. Thus, the idea of increasing the number of patches is not suitable for an integer processor. As an example, the number of patches on the B-spline surface model in Example 4.1 is quadrupled and shaded by Lambertian shading. The shading result is shown in Figure 4.8 and it appears that the shading quality is comparable with Figure 4.7. In addition to the shading improvement, the image in Figure 4.8 provides more accurate shape information

than the one in Figure 4.7 because of larger number of patches. Table 4.1 summarizes the required resource comparison among the Lambertian shading Model, the Gouraud shading and Lambertian shading with 4 times as many patches using the impulse/summation approach.

	Any n, m and l		$n = 2500, m = 25,$ $l = 1000$	
	+ and -	× and /	+ and -	× and /
Lambertian	$3n + 10m - 12$	$3m - 6$	7738	69
Gouraud	$3n + 10m - 12 + 3l$	$3m - 6 + l$	10738	1069
Lambertian with four times more patches	$3n + 40m - 12$	$12m - 6$	8738	294

Table 4.1 Resource comparison among Lambertian shading Model, Gouraud shading and Lambertian shading with 4 times more patches when the impulse/summation approach is used where n is the number data points in the surface, m is the number of knots and l is the number of pixels in the projected image.

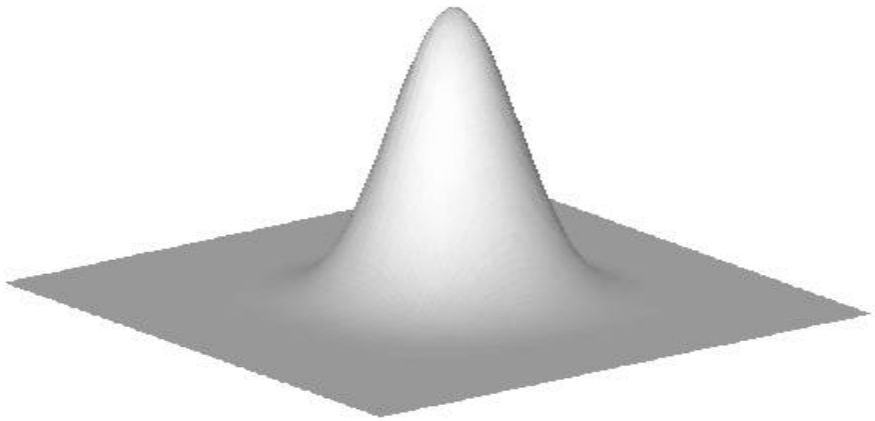


Figure 4.8 The Lambertian shading of the B-spline surface model in Example 4.1 with 4 times the number of patches

Chapter 5

Discrete Chebyshev Approximation

5.1 Introduction

This chapter contains the main subject of this dissertation. Section 5.2 discusses Chebyshev approximation in the discrete domain. The differences between the continuous domain and the discrete domain are shown in Section 5.2.1, 5.2.2 and 5.2.3. The most important changes are in Section 5.2.2. The absence of the derivative concept forces a new rule for determining a local extreme point. The new rule is a contribution of this dissertation which makes it possible to use the Reméz exchange algorithm in discrete Chebyshev approximation. In Section 5.3, we prove that discrete Chebyshev polynomial approximation is an optimal approximation in the discrete domain for certain well-defined, discrete functions. The proof (one of the new contributions) is given in Section 5.3.1. Another new contribution is related to the algorithm implementation. As discussed in Section 2.2.2, the Reméz exchange algorithm solves Equation 2.4 recursively. However, the square matrix in Equation 2.4 tends to be unbalanced and obtaining its inverse is difficult. The difficulty can be avoided using Cramer's rule for optimal error estimation and an interpolation scheme for the corresponding parameters. Section 5.4 is devoted to discrete, Chebyshev free-knot spline approximation. One of algorithms in Section 5.4.2 inserts knots during the approximation. Although its convergence and stability cannot be proved, the development contains examples showing the power of the new technique.

5.2 Fundamental Concepts

For an approximation, changing the domain from continuous space to discrete space affects some parts of the four approximation elements of Section 2.2, (if not all of the elements). This is especially true in dealing with the characteristics of a best solution and the algorithms that determine the best solution. The lack of a continuity concept continuity in the discrete space is the basis for all of these differences.

5.2.1 Existence and Uniqueness

For a linear, discrete Chebyshev approximation, we can use Equation 2.1 to 2.4, after substituting a discrete variable for every continuous variable. The discrete version of Equation 2.3 is

$$D(n_1, n_2, \cdots, n_k) = \begin{bmatrix} f_1[n_1] & f_2[n_1] & \cdots & f_k[n_1] \\ f_1[n_2] & f_2[n_2] & \cdots & f_k[n_2] \\ \vdots & \vdots & \ddots & \vdots \\ f_1[n_k] & f_2[n_k] & \cdots & f_k[n_k] \end{bmatrix} \tag{5.1}$$

Where $b_s = n_1 < n_2 < \cdots < n_k = b_e$ and $f_i[n_i]$'s are the discrete basis functions of the system. When the determinant of Equation 5.1 is non-zero for all $b_s = n_1 < n_2 < \cdots < n_k = b_e$, the discrete basis functions form a discrete Chebyshev system. The Haar condition is also met and the full rank of the discrete version of the matrix in Equation 2.4 is guaranteed. Thus, there is always a unique, best Chebyshev approximation. When the determinant can be greater or equal to zero for any given condition, the basis functions form a weak Chebyshev system. Thus, there may be more than one solution, which is better than the other approximations.

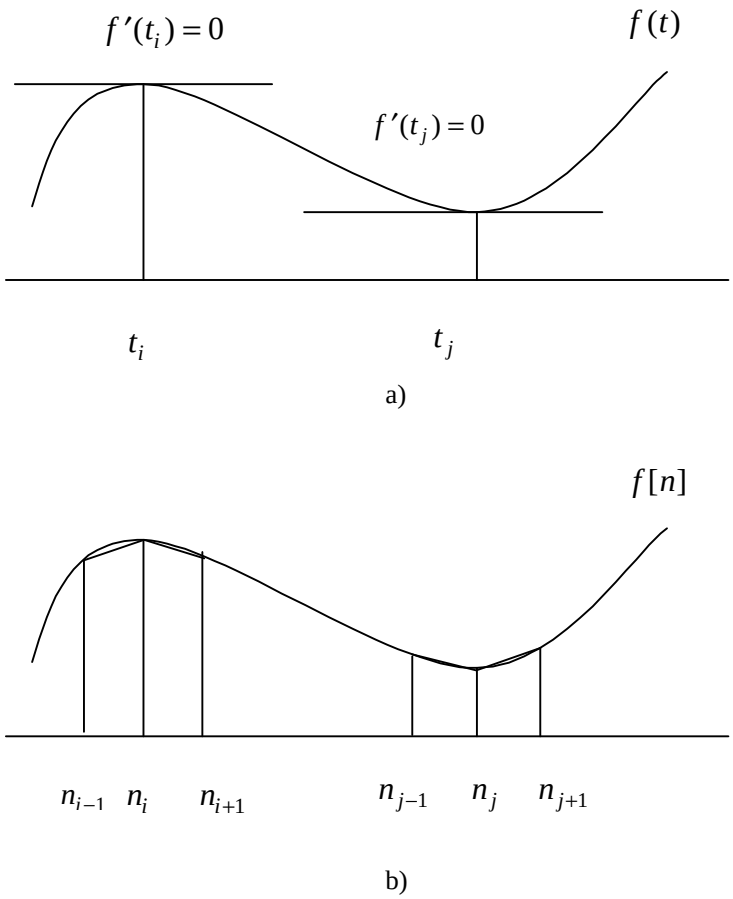


Figure 5.1 a) Extremes of a continuous function, b) Extremes of a discrete function

The independence of the Haar sub-matrix in a discrete Chebyshev system is stronger than its continuous counter part because the matrices in Equation 2.3 and 2.4 are closer to singular when in the continuous domain one of the t_i is approaching the value of an adjacent t_{i-1} or t_{i+1}. This forces a slow conversion of the approximation or a blocking iteration of the algorithm in finding a best solution for continuous Chebyshev approximation. The discrete nature prohibits this from happening in the case of discrete Chebyshev approximation. However, the lack of compactness in the domain obscures the existence of a best solution because it is possible that a best solution will fall between the sample points. Thus, additional considerations are required.

5.2.2 Characteristics

In terms of error alternations and their locations, a best solution's characteristics for a discrete Chebyshev system are the same as in the case of the continuous Chebyshev approximation. It means, the first three items of the best solution description in Section 2.2.2 are preserved. However, the last one must be changed because there is no derivative concept to justify local extremes at the alternation points in the discrete domain. Again, the difference concept is only a similar concept to the derivative concept of the continuous domain (whether it is the backward difference or the forward difference). As shown in a) of Figure 5.1, the local extremes of a continuous function are located at the zeros of the first derivatives. However, even if the continuous function is sampled, the local extremes of the discrete function may or may not be at the corresponding continuous function's extreme point locations. Usually, they are not co-located because the extreme points of the continuous function may not be selected during sampling. Also, there is a possibility that any of the first differences of a discrete function can be zero. From b) of Figure 5.1, it can be observed that the discrete extremes of $f[n]$ are accompanied by different polarities for the first backward difference $\nabla f[n]$ and the first forward difference $\Delta f[n]$ at these points. Thus, the discrete local extremes are obtained by checking the polarity changes in the two first differences. When the differences are zero, it is necessary to check both adjacent differences to determine whether it is an extreme point or not because there is the possibility that the point is at an inflection[48] point (this procedure is necessary for a extreme point checking of a continuous function as well). New rules to determine local extreme point n_{e_i} for a discrete error function $e[n] = f[n] - g[A,n]$ ($g[A,n]$ is the discrete counter part of an approximation function in Equation 2.1) can be written as

- $\nabla f[n_{e_i}]\Delta f[n_{e_i}] \le 0$:

  When $\nabla f[n_{e_i}] = 0$, $\nabla f[n_{e_i} - 1]\Delta f[n_{e_i}] \le 0$

  When $\Delta f[n_{e_i}] = 0$, $\nabla f[n_{e_i}]\Delta f[n_{e_i} + 1] \le 0$

- $e[n_{e_{i-1}}]e[n_{e_i}] \leq 0$: This guarantees the zero-crossings of the error function.

The above local extreme checking rule can be used in the discrete Reméz exchange algorithm.

Discreteness also affects the possible choice of extreme points. They can only be on the sampling grid because $f[n]$'s differences are only defined on the grid. It eliminates the possibility of finding better extreme point choices between the grid points. Every effect of discreteness is not adverse. Because the values on the points between the grid are ignored, even when the values on the continuous version of the error function exceed the allowed error at these points, as long as the values on the grid are bounded by this error, the approximation is acceptable as a valid solution. In addition, the continuity constraint of the original function also disappears with discreteness. Constraints are more relaxed than in the continuous approximation.

5.2.3 Algorithm

As long as the approximation basis functions form a Chebyshev system, the Reméz Multiple Exchange algorithm is effective for the discrete domain approximation. Of course, the new rule of Section 5.2.2 must be used for detecting new extreme points. The iteration of the algorithm is more stable in a discrete system than in a continuous system because there is no possibility of the "closeness problem" between two extremes occurring (as was discussed in Section 5.2.1).

Another advantage in discrete Chebyshev approximation occurs when the approximation basis functions form a weak Chebyshev system. The approximation can be written as a linear programming problem to minimize the maximum error with respect to the coefficients of the system[49]. Because the Chebyshev norm d of the error function is defined everywhere on the original function domain as

$$d = \max_{n} \left| f[n] - g[A, n] \right| \tag{5.2}$$

the number of constraints for the problem is large and it takes extensive computation to obtain the approximation. Many efficient algorithms have been developed for finding the solution when the problem size is large. Among them, the Simplex[38] method can be used for linear programming problems. Fixed knot polynomial spline and fixed knot B-spline approximation fit within these categories because they are linear weak Chebyshev systems. Because we want to discuss free knot splines here instead of fixed knot, the Simplex method cannot be used and the details of the Simplex method implementation for fixed knot splines is not discussed further.

5.3 Polynomial Approximation

Chebyshev polynomial approximation in the continuous domain has been well studied in the past. The power bases t^i of a polynomial form a linear Chebyshev system. A polynomial and its derivatives are well defined from $-\infty$ to ∞.

5.3.1 Polynomial Approximation Elements

In this section, the discussion of the uniqueness and existence of a best solution for a discrete polynomial approximation is the same as the discussion on characteristics and algorithm in Section 5.2. The matrix in Equation 5.1 for a discrete polynomial (or a continuous polynomial) guarantees uniqueness of any polynomial. No polynomial representation can have the same curve unless it has the same parameters as another representation. The uniqueness of each polynomial is preserved. Are the other aspects (existence and uniqueness of a best solution) of Chebyshev Approximation preserved without compactness of the discrete domain? Let's write the absolute value of the error function between the original function $f[n]$ and a k^{th} order discrete polynomial $p[n]$. Then,

$$|e[n]| = \left| f[n] - \sum_{i=0}^{k-1} a_i n^i \right| \tag{5.3}$$

where the a_i's are real coefficients for $i = 0,1,\cdots,k-1$ and n is defined on the approximation interval $[0, n_s]$ (without loss of generality). Because the Chebyshev norm is defined as the maximum of Equation 5.3 with respect to n,

$$\begin{aligned} &\left| f[0] - \sum_{i=0}^{k-1} a_i 0^i \right| \le d \\ &\left| f[1] - \sum_{i=0}^{k-1} a_i 1^i \right| \le d \\ &\quad\vdots \\ &\left| f[n_s] - \sum_{i=0}^{k-1} a_i n_s^{\,i} \right| \le d \end{aligned} \tag{5.4}$$

for $n_s > k$ (when $n_s \le k$, there is no need to search for an approximation because several polynomials interpolate the original function without any error) and a positive number d. In Equation 5.4, the original function values $f[n]$ and the power bases $n^0, n^1, \cdots, n^{k-1}$ are given fixed values for $n = 0,1,\cdots,n_s$. The problem, to be solved, can be restated as 'find a set of $a_0, a_1, \cdots, a_{k-1}$ such that we minimize the value of d with the given conditions of Equation 5.3. Let's transfer the problem to the coefficient space. The problem is not in a discrete domain any more because only the n's are integers while the other coefficients are real numbers. Although the dimension of the new system is expanded to k, the compactness of the domain is returned. After moving to the coefficient domain, each equation in Equation 5.3 represents a region (k dimensional volume) in the coefficient space. Each region V_n is defined as

$$\mathrm{V}_n = \left\{ a_0, a_1, \cdots, a_{k-1} \;\middle|\; \left| f[n] - \sum_{i=0}^{k-1} a_i n^i \right| \le d \right\} \tag{5.5}$$

for $n = 0,1,\cdots,n_s$ and a positive d. The size of each region V_n may increase with respect to d. Therefore, there exists a small enough d such that $\mathrm{V}_0 \bigcap \mathrm{V}_1 \bigcap \cdots \bigcap \mathrm{V}_{n_s} = \varnothing$ and a big enough d such that $\mathrm{V}_0 \bigcap \mathrm{V}_1 \bigcap \cdots \bigcap \mathrm{V}_{n_s} \neq \varnothing$. Thus, there is always a smallest d that satisfies $\mathrm{V}_0 \bigcap \mathrm{V}_1 \bigcap \cdots \bigcap \mathrm{V}_{n_s} \neq \varnothing$. In other words, a best solution or best solutions always exist. If $\mathrm{V}_0 \bigcap \mathrm{V}_1 \bigcap \cdots \bigcap \mathrm{V}_{n_s}$ has only a single point on the coefficient space as its element, the point represents a best polynomial approximation for $f[n]$, otherwise, there are more than one polynomial, which approximate $f[n]$ with the optimal Chebyshev norm error d. This may be considered as a proof of the solution existence.

Example 5.1 Consider a 2nd order Chebyshev polynomial approximation of $f[n]$, which has only three consecutive data [0, 0.4, 0.6] from $n = 0$. From Equation 5.5, the regions are written as

$$\begin{aligned} \mathrm{V}_0 &= \left\{ a_0, a_1 \middle| \, |a_0 - f[0]| \le d \right\} \\ \mathrm{V}_1 &= \left\{ a_0, a_1 \middle| \, |a_0 + a_1 - f[1]| \le d \right\} \\ \mathrm{V}_2 &= \left\{ a_0, a_1 \middle| \, |a_0 + 2a_1 - f[2]| \le d \right\} \end{aligned} \tag{5.6}$$

The regions are shown as Figure 5.2, when $d = 0.005$. Also, Figure 5.2 shows, the common region of V_0, V_1 and V_2 is a single point at the point $a_0 = 0.05$ and $a_1 = 0.3$. Thus, the best approximation of $f[n]$ is $0.05 + 0.3n$. Since the point is located above the center lines of V_0 and V_2, and is below the one of V_1, the errors between the approximation and the original are .05 at the data points $n = 0$ and $n = 2$, and -0.05 at $n = 1$. It indicates, the error alternation characteristic in a discrete approximation is also preserved. When there are more than three data sequences on $f[n]$, only three extreme points have the maximum absolute error between the best approximation and $f[n]$, and the remainders have smaller errors than the maximum absolute error.

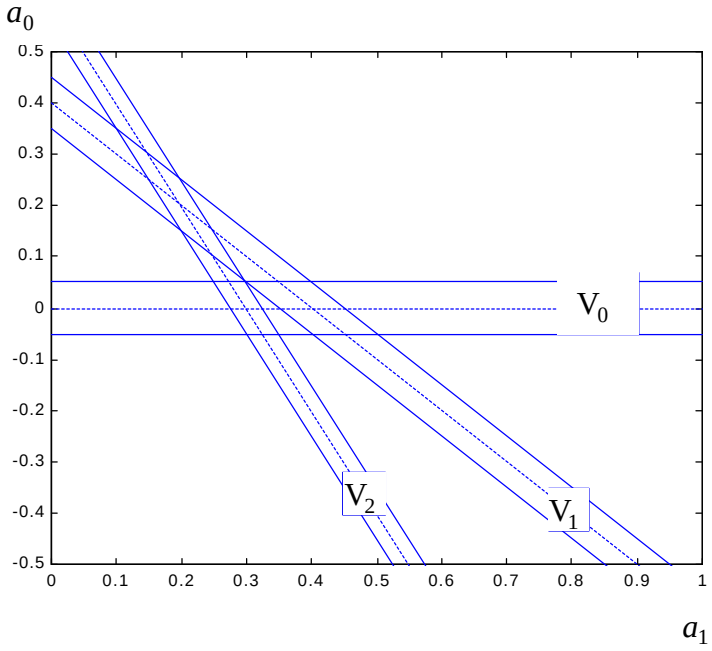


Figure 5.2 The region of V_0, V_1 and V_2 of Example 5.1 in the coefficient space: The dotted lines indicate the center of the regions.

Convexity of the solution space is required for the existence of a unique best solution for any problem. Let's add the maximum error d as one more dimension to the coefficient space. The new space has $k+1$ dimension and is continuous. Let's rewrite each equation in Equation 5.4 as two equations

$$f[n]-\sum_{i=0}^{k-1}a_i n^i - d \le 0$$
$$\textit{and} \qquad (5.7)$$
$$f[n]-\sum_{i=0}^{k-1}a_i n^i + d \ge 0$$

for $n = 0,1,\cdots,n_s$. When the equations of Equation 5.7 are equalities each equation represents two parallel half planes in the new $k+1$ dimensional space. Thus, each equation of Equation 5.7 represents a region with its boundary defined by its two half

planes. There will be a non-empty overlapping region specified by the equations in Equation 5.7. The common region is defined by the intersections of the half planes. Any region defined by intersections of half planes forms a convex hull[38]. When a minimum positive d is sought, if the solution space of the problem forms a convex hull in a continuous space, a best solution exists. Thus, there is always a unique best solution for a discrete Chebyshev polynomial approximation.

Example 5.2 Consider the 2nd order approximation in Example 5.1. The equations in Equation 5.7 are rewritten as

$$\begin{aligned} &a_0 - f[0] - d \le 0 \text{ and } a_0 - f[0] + d \ge 0 \\ &a_0 + a_1 - f[1] - d \le 0 \text{ and } a_0 + a_1 - f[1] + d \ge 0 \\ &a_0 + 2a_1 - f[2] - d \le 0 \text{ and } a_0 + 2a_1 - f[2] + d \ge 0 \end{aligned} \tag{5.8}$$

The regions of the above equations can be plotted in 3-D space and a best solution can be determined from the plot. The cross-sectional view of the 3-D plot is shown in Figure 5.3. The figure shows a unique minimal point at $d = 0.05$ and $a_0 = 0.05$ of the common region. The cross-sectional view at $a_0 = 0.05$ would show a similar plot and the minimal point would then be at $d = 0.05$ and $a_1 = 0.3$. These results are consistent with Example 5.1.

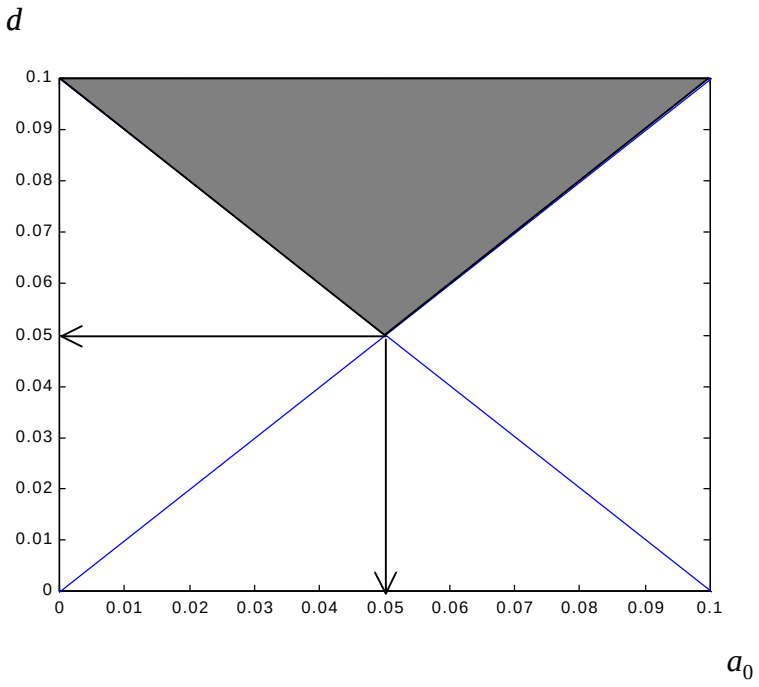


Figure 5.3 The cross-sectional view of the solution space in Example 5.2 at $a_3 = .3$: the shaded area indicates the cross-section of the common region

5.3.2 Algorithm

All of the aspects of the Reméz Multiple Exchange Algorithm application to a discrete polynomial approximation are discussed in the previous sections except the number of error alternations. Because the coefficients and the Chebyshev norm error form a $k+1$ dimensional space, $k+1$ error alternations must occur. Thus, it is valid to use the Reméz Multiple Exchange Algorithm for a discrete Chebyshev polynomial approximation problem.

Two essential parts of a Reméz multiple exchange algorithm are obtaining extrema and estimating coefficients and the L_∞ error norm. As discussed in Section 2.2.2, the coefficients and the error can be obtained by solving Equation 2.4. Let's rewrite Equation 2.4 for a k^{th} order discrete polynomial approximation. Thus,

$$\begin{bmatrix} 1 & n_{e_1} & \cdots & n_{e_1}^{k-1} & -1 \\ 1 & n_{e_2} & \cdots & n_{e_2}^{k-1} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & n_{e_{k+1}} & \cdots & n_{e_{k+1}}^{k-1} & (-1)^{k+1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \\ d \end{bmatrix} = \begin{bmatrix} f[n_{e_1}] \\ f[n_{e_2}] \\ \vdots \\ \vdots \\ f[n_{e_{k+1}}] \end{bmatrix} \tag{5.9}$$

Equation 5.9 looks very simple to solve. However, as the length of $f[n]$ increases and as k becomes higher, the matrix in Equation 5.9 approaches *ill-conditioned* [22] because it becomes an unbalanced matrix. Parks and McClellan[50] found a nice way around this problem in their FIR filter design schemes. They split the task into two. First, they calculate the error d analytically with their driven equation. Next, they interpolate the sums of the error and desired values at the local extremes rather than solving the matrix equation for the coefficients. Their strategy can be used in here with a slight modification. Because there is no analytical solution for the error, the error will be obtained by Cramer's rule[51]. Cramer's rule works with the matrix determinants rather than the matrix inverse so it provides a more accurate error estimation at each iteration. The coefficients are obtained by LaGrange's[22] interpolation formula as used by Parks and McClellan. Thus, an algorithm can be stated

Algorithm-1

Step 1. Initial guesses on the set of local extreme points $n = n_{e_i}$ for $i = 2,3,\cdots,k$ (the first and the last extreme points are fixed at $n_{e_1} = b_s$ and $n_{e_{k+1}} = b_e$).

Step 2. Obtain an estimated error d from the set of the local extremes n_{e_i} with Equation 5.9 and Cramer's rule.

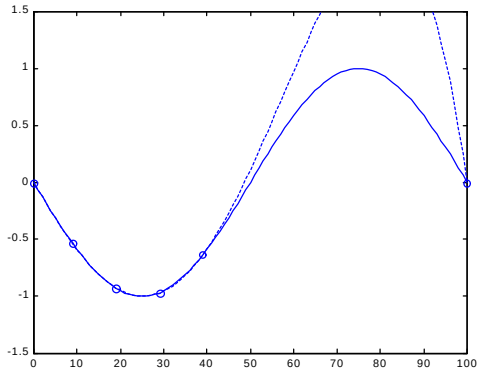
Step 3. Interpolate the sums of the error and desired values at the local extremes with LaGrange's interpolation.

Step 4. Obtain the error function $e[n]$ between $f[n]$ and $p[n]$ ($p[n]$ is an interpolated polynomial).

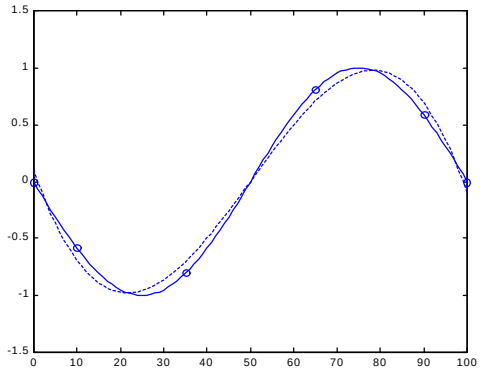
Step 5. Obtain a new set of local extreme points from $e[n]$ with the rule in Section 5.2.2.

Step 6. When the extreme points converge, stop. Otherwise go to Step. 2.

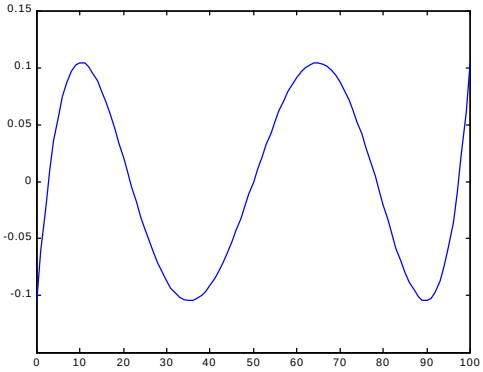
Example 5.3 A non-polynomial function $f[n]=\sin\left(2p\left(n-50\right)/100\right)$ is defined on $n=[0,100]$. Let's approximate $f[n]$ with both a 5th and 7th order discrete Chebyshev polynomial approximation. The 5th order solution starts with the extremes [0, 10, 20, 30, 40, 100]. The initial guesses for the approximation and the first interpolation are shown in a) of Figure 5.4. After the 4th iteration, the extremes are converged at [0, 11, 36, 66, 91, 100] and the error function is confined with Chebyshev norm $d=0.1047$ with 6 alternations as shown in b) and c) of Figure 5.4. The 7th order approximation result after converged is shown in Figure 5.5. As in the case of the 5th order solution, the approximation is found as expected. The maximum error d is 0.0068. The approximation and $f[n]$ are not distinguishable in Figure 5.5. The two approximations demonstrate, first, that the convergence is very fast using the Reméz Exchange Algorithm, and, second, the Chebyshev norm errors are reduced significantly as the order of the approximation increases. Also, Algorithm-1 works over a wide range of n. The 7th order makes the smallest non-zero element in the matrix of Equation 5.9 to be 1 and the largest element to be 10^{14}.



a)

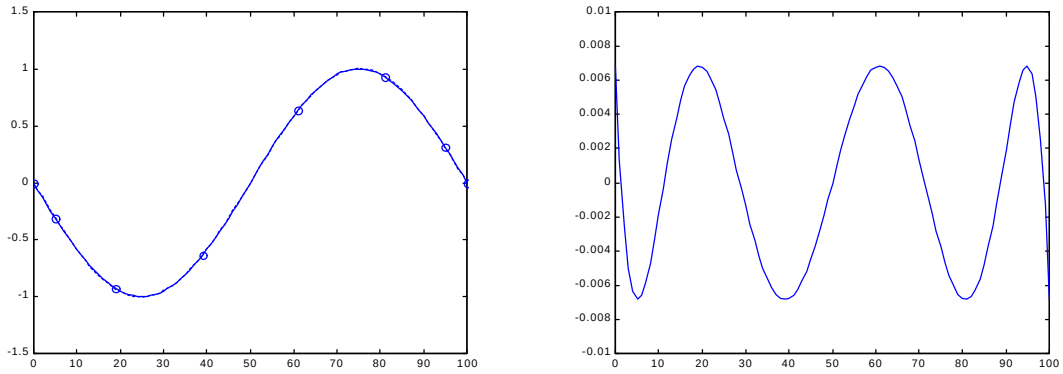


b)



c)

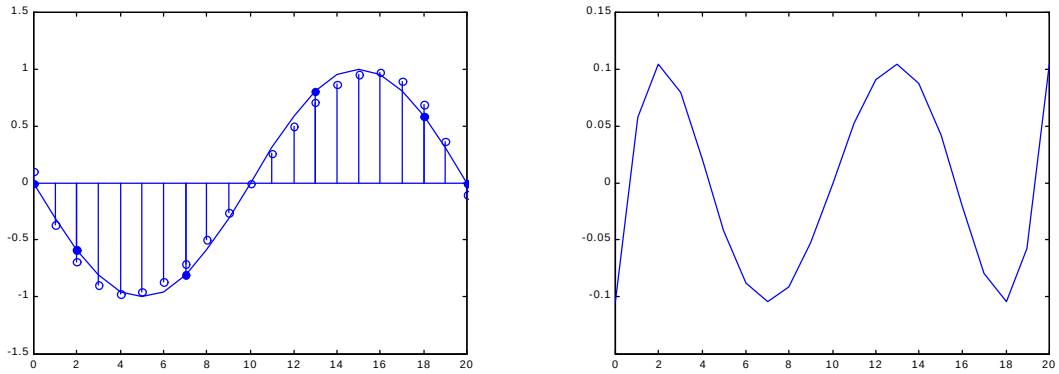
Figure 5.4 a) Initial extreme points and the first interpolation, b) The extreme points and the interpolation after convergence of the 5^{th} order approximation and c) Its error function after convergence: dotted lines indicate approximation, solid lines indicate $f[n]$ and o's mark extrema.



a) b)

Figure 5.5 a) The extreme points and the interpolation of the 7th order polynomial approximation after convergence and b) The error function after convergence: dotted lines indicate approximation, solid lines indicate $f[n]$ and o's mark extremes.

Example 5.4 The non-polynomial function $f[n]=\sin\left(2p\left(n-10\right)/20\right)$ is defined on $n=[0,20]$. Let's approximate $f[n]$ with a 5th order discrete Chebyshev polynomial approximation as in Example 5.3. The difference in this example and Example 5.3 is only in the sampling periods. In this example, the same continuous sine curve is sampled 5 times less frequently over the same period as in Example 5.3. However, almost the same approximation is achieved as is shown in Figure 5.6. The maximum error d is identical with Example 5.3 because Algorithm-1 tries to find the same local extremes in the error function no matter how many data points iterations. Of course, a little phase shift in the sampling could change the result. The maximum error would be less than .1047 because the current sampling produces some data points at the same peak of the continuous sine function and slight change in the sampling point will make a peak less than the peak of the sine. This example demonstrates that Algorithm-1 can be used in situations, which have a small number of discrete data points. $f[n]$ in this example has only 21 data points.



a) b)

Figure 5.6 a) The extreme points and the best approximation of the 5th order polynomial approximation after converged and b) Its error function after converged: stems indicate the approximation, solid lines indicate $f[n]$ and filled o's mark extremes.

5.4 Free Knot Polynomial spline and B-spline

Because two splines contain one-sided power functions as basis functions in both the continuous and discrete domains, the free knot case generates a non-linear system. Each time the knot locations are changed, the basis functions also change shape. Therefore, approximation with free knot polynomial splines or B-splines is a non-linear optimization process. Even though the system is non-linear, if the parameters of the system and its error form a convex space, there is always a best solution for any problem in the non-linear system[16].

5.4.1 One-Sided Power Function

With a one-sided power function $(t-h)_+^n$, Equation 2.5 can be written as

$$(t-h_1)_+^n - (t-h_2)_+^n \geq n(h_1-h_2)(t-h_2)_+^{n-1} \qquad (5.10)$$

where h, h_1 and h_2 are real numbers. Equation 5.10 is true for any h_1 and h_2, and the equality can hold for some cases. Thus, unfortunately, the one-sided power function is semi-convex with respect to its parameter h (ref. Equation 3.8). Semi-convexity does not provide many alternatives in finding an approximation algorithm. Any derivative method, such as Gauss-Newton, pushes the new solution h near the zero point of the basis function, when the current solution is located on the right side of a previous value of h. When the new solution h moves to the left side, there is no way for a derivative algorithm to keep the iteration running. Thus, it is very difficult to use a derivative method in an algorithm to a best approximation with a free knot polynomial spline or a free knot B-spline. This situation is not improved by moving to a discrete system because the non-linearity of the bases or some of bases still remains. On other hand, a "shaping method", such as the Reméz Exchange Algorithm, will work because it is looking for an equiripple error solution for a given number of alternations. Of course, the approximation may not be a best approximation unless the correct number of alternations is used and the proper conditions of the approximation are given.

5.4.2 Algorithms

There are many difficulties in developing a Reméz Exchange Algorithm for free knot spline approximation. First, there is no definite number of error alternations for a best approximation. Second, no good relationship can be found between the knot locations and the extrema locations. Third, no good estimation of the optimal error can be found for the given spline and the original function when we have a smaller than optimal number of alternations.

To overcome these difficulties, some restrictions need to be imposed on the algorithm. Two algorithms are represented in this section. Both of them are Reméz type algorithms[17]

(since they are not producing optimal errors, it is not correct to call them Reméz Exchange Algorithms).

The restrictions of the first algorithm are

- The extreme points and the knots are set at the same locations.
- The minimax error will be estimated by an average of the extreme values of the error function and the system will have a fixed number of knots

The first restriction makes the determinant of the matrix in Equation 2.4 non-zero. If the matrix becomes singular, it is impossible to keep iterating the algorithm. The first restriction also resolves the problem of knot assigning from the extremes. It is heuristic approach and makes the approximation far from the best. However, it will converge as long as a large the enough number of alternations is set. The algorithm can be stated

Algorithm-2

Step 1. Initial guesses on the set of local extreme points $n = n_{e_i}$ for $i = 2,3,\cdots,k$ (the first and the last extreme points are fixed at $n_{e_1} = b_s$ and $n_{e_{k+1}} = b_e$), and estimated error d.

Step 2. Interpolate the sums of the error and desired values at the local extremes with polynomial spline interpolation.

Step 3. Obtain the error function $e[n]$ between $f[n]$ and the spline.

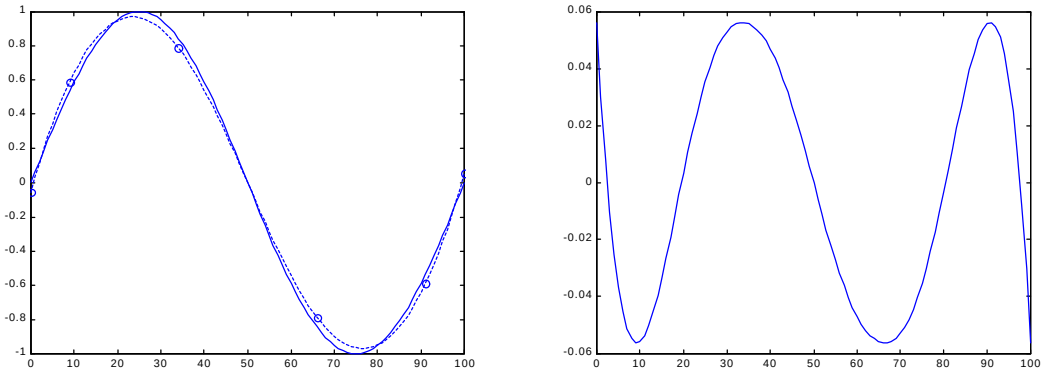
Step 4. Obtain new set of local extreme points from $e[n]$ with the rule in Section 5.2.2.

Step 5. When the extreme points converge, stop. Otherwise, go to Step 6.

Step 6. Set d with the current absolute average of the extreme values, and go to Step 2.

Example 5.5 A non-polynomial function $f[n] = \sin\left(2p\left(n-50\right)/100\right)$ is defined in the same way as in Example 5.3. Let's approximate $f[n]$ with a 4^{th} order discrete Chebyshev polynomial spline approximation by using Algorithm-2. 6-error-alternations

are picked to compare with the 5th order polynomial Chebyshev approximation in Example 5.3. The approximation result is shown in Figure 5.7. After the 20th iteration, the extremes are converged at [0, 10, 35, 67, 92, 100] and the error function is confined with Chebyshev norm $d = 0.0561$ as shown in Figure 5.7. Although the algorithm cannot guarantee a best approximation, the approximation error here is about twice as small as the one from the 5th order polynomial Chebyshev approximation in Example 5.3. Polynomial splines are more effective than polynomials for the same degree of the approximation problem. With a better error estimation method, the approximation can be improved beyond the result of this algorithm. However, the algorithm converges much more slowly than in Example 5.4. Thus, the algorithm may not be suitable to a real time approximation application.



a) b)

Figure 5.7 a) The extreme points and the result of the 4th order polynomial spline Chebyshev approximation with Algorithm-2, and b) Its error function after converged: dotted lines indicate approximation, solid lines indicate $f[n]$ and o's mark extremes.

The restrictions of the second algorithm are

- The extreme points and the knots are set at the same locations.
- The minimax error will be given but the number of knots may vary.

The difference between the first algorithm and the second is in the second restriction about number of knots and the estimation of the error. Since there is no feedback on error, the number of the knots and their locations are the only controlling factors used to achieve equiripple approximation. The second algorithm utilizes the fact that when the fluctuations of the original function are more frequent than the polynomial order of the approximation function there is a greater number of the local extrema than the number of basis functions used in the approximation. For example, consider an original function $f[n]$ as in Example 5.5 and the approximation function is a second order spline with only two knots at both ends. In this case, there will be four extreme points in its error function and the four points will produce four constraint equations. There are more constraints than are needed to determine the second order spline with two knots. However, the extra extrema will provide a hint for a new knot insertion in the second algorithm. For this particular algorithm, the extra extrema defines the new knot insertion point because of the first restriction. However, in some cases, the algorithm may not converge because there is a possibility that no extra extrema occurs without forming an equiripple error. When this happens, the algorithm may oscillate between one set of knots and the other during iteration. Although this algorithm cannot guarantee convergence, its knot insertion capability gives a good solution when there is no error alternation number guidance. Also, the algorithm is closer to a Reméz Exchange Algorithm than Algorithm-2 (except the error estimation procedure). The algorithm can be stated as

Algorithm-3

Step 1. Set $n_{e_i} = [b_s, round\left(\left(b_s + b_e\right)/2\right), b_e]$ as the initial local extremes (the first and the last extreme points are fixed at $n_{e_1} = b_s$ and $n_{e_{k+1}} = b_e$).

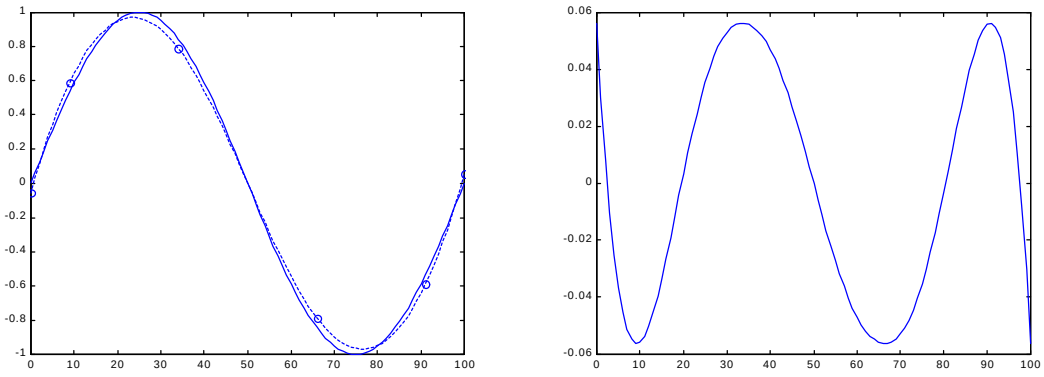
Step 2. Interpolate the sums of the error d and desired values at the local extremes with polynomial spline interpolation.

Step 3. Obtain the error function $e[n]$ between $f[n]$ and the spline.

Step 4. Obtain new set of local extreme points from $e[n]$ with the rule in Section 5.2.2.

Step 5. When the extreme points converge, stop. Otherwise, go to Step 2.

Example 5.6 The non-polynomial function $f[n] = \sin\left(2p\left(n-50\right)/100\right)$ is defined as in Example 5.3. Let's approximate $f[n]$ with a 4th order discrete Chebyshev polynomial spline approximation using Algorithm-3. The algorithm starts with the initial extremes at [0, 50, 100] and the given error $d = 0.0561$. When it converges, the approximation result is exactly the same as the one in Example 5.5 as shown in Figure 5.8 because the error d is set to be 0.0561 as in Example 5.5. Consequently, the convergence of the extremes at the same locations [0, 10, 35, 67, 92, 100] with Example 5.5 is the obvious result. However, the initial knot location change may lead to another result, which converges to different extremes and error, or oscillates between two sets of knots.



a) b)

Figure 5.8 a) The extreme points and the result of the 4th order polynomial spline Chebyshev approximation with Algorithm-3, and b) Its error function after convergence: dotted lines indicate approximation, solid lines indicate $f[n]$ and o's mark extremes.

Chapter 6

FIR Filter Design

6.1 Introduction

This chapter provides, the theoretical basis (in Section 6.2) to use the impulse/summation approach for implementing digital filters and design examples for polynomial approximated FIR filters (in Section 6.3) and polynomial spline approximated FIR filters (in Section 6.3). By proving Theorem 6.1, the mathematical validity for using repeated summation in the implementation of an approximated FIR filter is established. The theorem states, the convolution operator and the repeated summation operator are interchangeable. Also, at the end of Section 6.4, the computational complexity comparison is summarized with Table 6.1.

6.2 Motivation and Theories

A FIR filter impulse response is a discrete function, which has non-zero values on an upper and lower bounded interval. Conventionally, it is implemented with a series of delays, multipliers and adders in a custom VLSI design or as a software program. The number of the non-zero filter values indicates the number of delays, the number of multipliers and the number of adders required for its implementation. Among the elements of the filters, the multiplier is the largest element on a chip and also the most power consuming. Reducing the number of multipliers is directly connected to low power VLSI design and cost reduction for its production.

Chebyshev Approximation of an FIR filter response with a polynomial or spline provides a way of reducing the number of multipliers and the number of adders required for implementation using the impulse/summation scheme. However, it may produce artifacts. The artifacts can be controlled within a reasonable level by increasing the order of the polynomial or the number of the spline knots.

6.2.1 Convolution

In a discrete linear shift invariant system, applying a filter to the signal is achieved by convolution between the filter sequence $h[n]$ and the signal sequence $x[n]$. The convolution operator[52] $*$ is defined as

$$h[n]*x[n]=\sum_{i=-\infty}^{\infty}h[i]x[n-i] \tag{6.1}$$

The impulse/summation scheme may provide a way of reducing the complexity of a filter implementation. We must first show that the convolution operator $*$ and the repeated summation operator $(\cdot)_{[k]}$ are interchangeable.

Theorem 6.1 Operator $*$ and operator $(\cdot)_{[k]}$ are interchangeable. Thus, whenever $h[n]=(I[n])_{[k]}$,

$$h[n]*x[n]=(I[n]*x[n])_{[k]} \tag{6.2}$$

for any $x[n]$. See Appendix H. for a proof.

6.2.2 End of Approximated Filter

Filter implementation with a polynomial or spline impulse response requires strictly zero values out side of the approximated interval because the signal sequences to be filtered are continuously fed to the filter. The repeated summation operator may run for a very long time or endlessly. This requirement can be achieved by the segmentation of a polynomial as described in Section 3.3.1 for the case of a polynomial filter approximation and the case of a polynomial spline filter approximation. Because the B-spline has only finite length, if B-spline approximation is applied to a filter, this finite support requirement is achieved automatically.

6.3 Polynomial Approximated Filter

After approximating a filter response with a Chebyshev polynomial approximation, a polynomial segment is produced. Let's assume the segment is defined on $[b_s, b_e]$ of the discrete domain. Two one-sided polynomials can be obtained from the segment: one starts from b_s and the other starts from b_e as shown in Figure 3.1. Because the segment can be expressed by the subtraction of the last one-sided polynomial from the first one, the segment can be constructed with Implementation-3 by supplying two coefficient-weighted impulse sequences into each of the Implementation-3 inputs. The coefficients are from two one-sided polynomials. The first impulse on each difference represents the one beginning at b_s and the second represents the one beginning at b_e. The gaps between two impulses on every difference are $b_e - b_s$. For the polynomial approximated filter implementation, the inputs of Implementation-3 are acting as filters, which have only two non-zero coefficients. The filter implementation can be achieved simply by feeding the signal into each of the two-coefficient filters. An example of a k^{th} order Fast Polynomial Approximated FIR Filter is shown in Figure 6.1. Since every h_i is a FIR filter, each requires two multipliers, one adder and $b_e - b_s$ delays for its implementation. For the entire filter implementation, $2k$ multipliers, $k(b_e - b_s + 1)$ delays and $3k+1$ two-input adders are required. The overall hardware complexity for its implementation is

about $2kw^2 + (k(b_e - b_s + 7) + 2)w$ when w represents the hardware complexity of an adder. Of course, the above overall complexity measure is approximate because the assumptions, the hardware complexity of a multiplier is w^2 and the one of a delay is about w, are very rough. A conventional implementation of an arbitrary FIR filter requires $b_e - b_s$ multipliers, $b_e - b_s - 1$ delays and $b_e - b_s - 1$ two-input adders. The overall hardware implementation complexity for the conventional approach is roughly $(b_e - b_s)w^2 + 3(b_e - b_s - 1)w$. However, there are four types[53] (odd length odd symmetry, odd length even symmetry, even length odd symmetry and even length even symmetry) of symmetry FIR filters. While an odd symmetry FIR filters require about same number of multipliers as the arbitrary FIR filter, an even symmetry filter can be implemented using $(b_e - b_s)/2$ multipliers ($(b_e - b_s - 1)/2$ multipliers for an odd length symmetry FIR filter) when the length of the filters is the same as that of the arbitrary FIR filter[54]. Therefore, its overall hardware complexity for the implementation is approximately $w^2(b_e - b_s)/2 + 3(b_e - b_s - 1)w$.

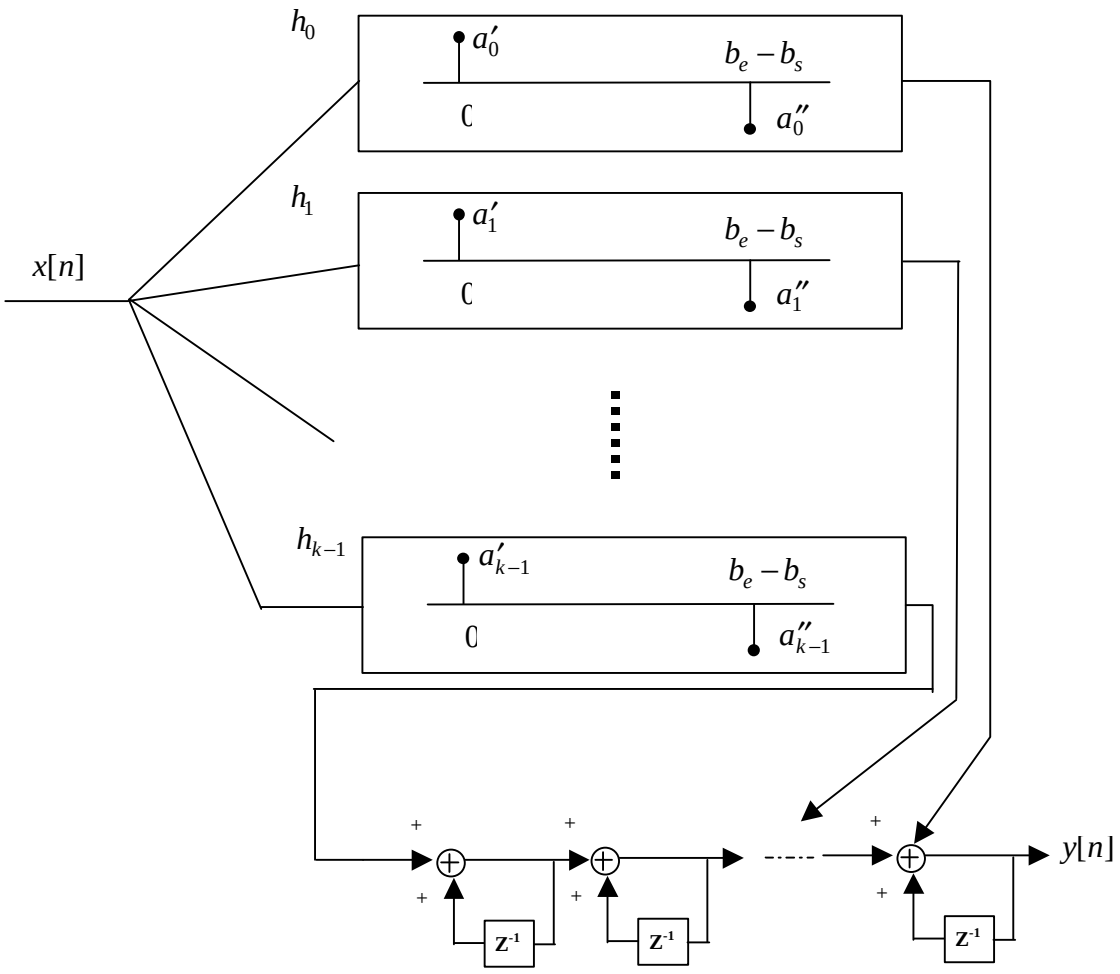


Figure 6.1 A Fast Polynomial Approximated FIR Filter

6.4 Polynomial Spline Approximated Filter

Since a polynomial spline can be considered as a combination of a polynomial and a B-spline as it is discussed in Chapter 3, a polynomial spline approximated FIR filter can also be constructed in almost same manner as the polynomial approximated filter in Section 6.3. Actually, the one-sided power function portion of the filter is integrated into the polynomial approximated filter implementation. From Equation 3.10, 3.12, 3.14 and the transformation in Appendix B, the necessary coefficients can be obtained for the polynomial part and the one-sided power function part. A 4^{th} order polynomial spline

approximated filter is shown in Figure 6.2 as an example. Because of the higher order polynomial's oscillation tendency, the required number of parameters and the minimal requirement of the second derivative smoothness, usually a 4th order polynomial spline is used to approximate a function. h_B of Figure 6.2 generates the necessary impulses for one-sided power function portions of the polynomial spline approximated filter. It has two parts, which are cascaded. The first filter h_{coeff} consists of weighted discrete sample functions, which are located at knots. The second filter $h_{one-sided}$ depends on the order of the polynomial spline. It has the same sequences as the k differences $\nabla^k \tilde{b}_k[n]$ of the one-sided discrete power function (see Table 3.2 for proper sequences). The details of h_B are shown in Figure 6.3 for a 4th order polynomial spline approximated filter implementation. If a B-spline approximated filter exists, it can be implemented by eliminating h_1, h_2, h_3 and h_4 from Figure 6.2. Since the polynomial portion is identical with Section 6.3 and the order is 4, the polynomial portion takes only 8 multiplies, $4(b_e - b_s + 1)$ delays and 13 two-input adders for its implementation. When m knots are used for the spline, the one-sided discrete power function requires $m+1$ multipliers, $b_e - b_s$ delays and $m+3$ adders. Total $m+9$ multipliers, $5(b_e - b_s)+1$ delays and $m+16$ adders are needed. Therefore, it takes approximately $(m+9)w^2 + (5(b_e - b_s)+m+17)w$ of the overall hardware complexity to implement the hardware for a polynomial spline approximated FIR filter. The required computational resources are heavily dependent on the number of knots in the approximation. Table 6.1 summarizes the required resource comparisons among conventional FIR filter implementations, polynomial approximated FIR filter implementations, polynomial spline approximated and FIR filter implementations.

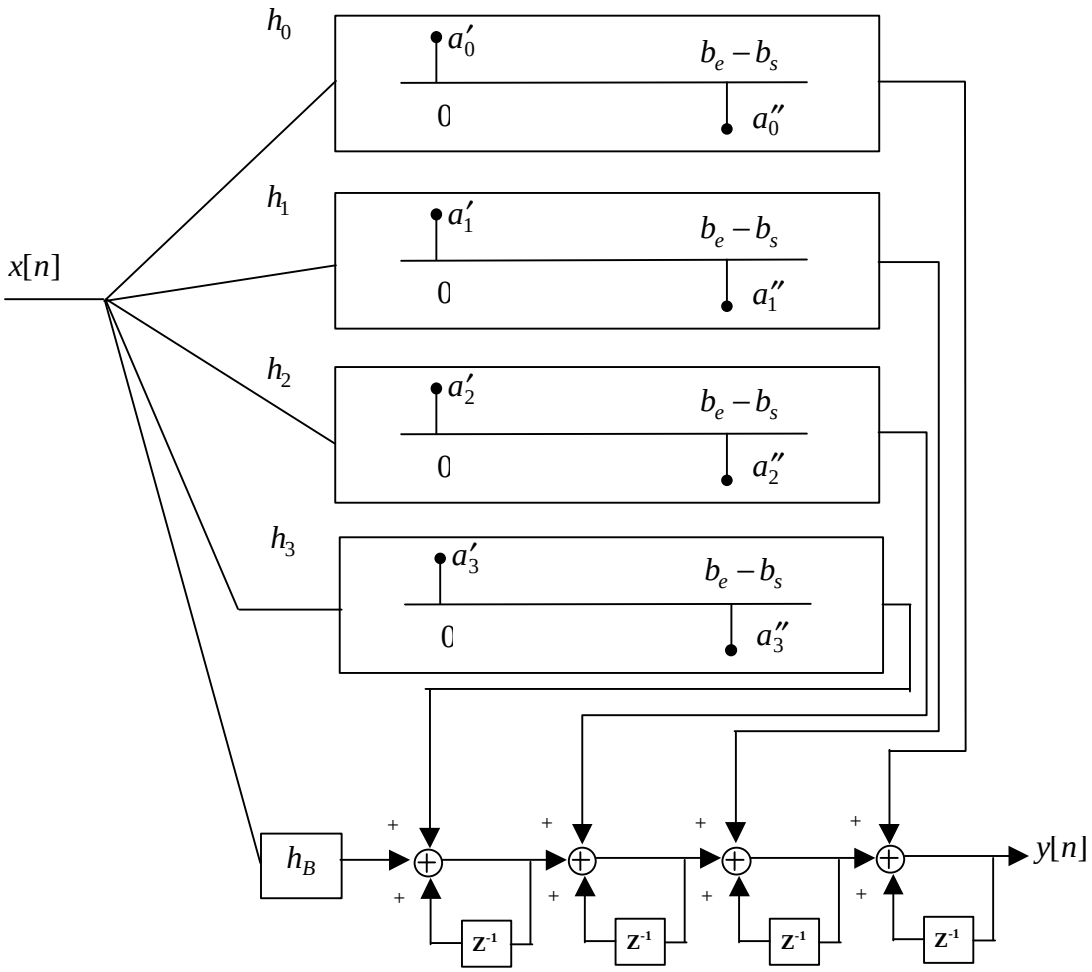


Figure 6.2 A Fast Polynomial Spline Approximated FIR Filter

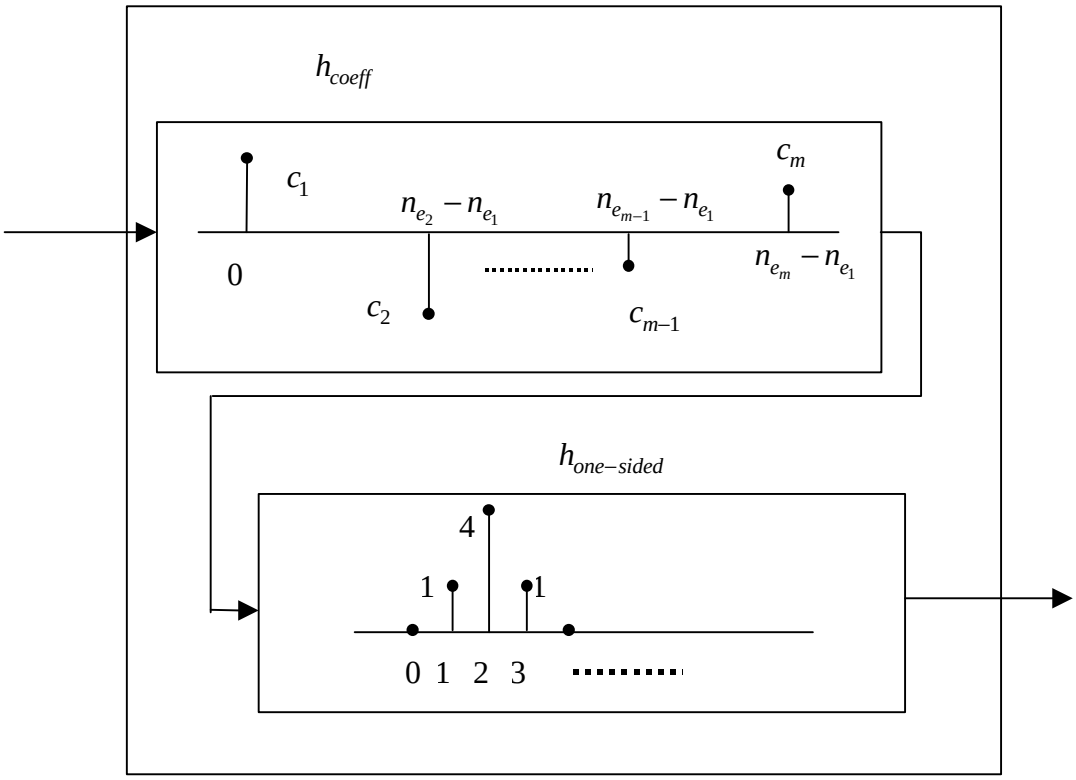


Figure 6.3 An example of h_B for the 4th order polynomial spline approximated filter

	No. Of Delays	No. Of Mult.	No. Of Add.	Overall HW Cmplx.*
Convent. (Even Symm.)	$b_e - b_s - 1$	$b_e - b_s$ $(b_e - b_s)/2$	$2(b_e - b_s - 1)$	$(b_e - b_s)w^2 + 3(b_e - b_s - 1)w$ $w^2(b_e - b_s)/2 + 3(b_e - b_s - 1)w$
Polynomial Approxi.	$k(b_e - b_s + 1)$	$2k$	$6k + 2$	$2kw^2 + (k(b_e - b_s + 7) + 2)w$
Poly.Spline Approx.	$5(b_e - b_s) + 1$	$m + 9$	$m + 16$	$(m + 9)w^2 + (5(b_e - b_s) + m + 17)w$

* w represents the hardware complexity of an adder. Roughly, the hardware complexity of a multiplier is about w^2 and the one of a delay is about w. Thus, the overall complexity measures in the table are representing the complexity very roughly.

Table 6.1 Required resource comparison among conventional FIR filter implementation, polynomial approximated FIR filter implementation, polynomial spline approximated and FIR filter implementation

6.5 Low Pass FIR filter Examples

In this section, two optimum lowpass FIR filters[50,52,55] are approximated using Chebyshev polynomial approximation and Chebyshev polynomial spline approximation. The specification of the first optimum lowpass FIR filter is the passband[52] frequency $f_p = 0.001$, the stopband[52] frequency $f_s = 0.025$, equal error weights and the filter duration $N = 65$. The filter is a very narrow-band high quality lowpass filter. The optimum lowpass FIR filter with this specification can be obtained with the Parks-McClellan algorithm[50,52,55], and the impulse response and the frequency response of the filter are shown in Figure 6.4. The impulse responses and the frequency responses of its polynomial approximated filter and polynomial spline approximated filter are shown in Figure 6.5 and 6.6. The optimum filter coefficients are very slow varying except at both ends. Even though we have no continuity concept in the discrete domain, the abrupt changes act as singular points in a continuous function. Thus, if the abrupt changes are not there, both approximated filters can achieve the desired error with fewer knots or a lower order of the polynomial. This analysis is applicable to the second filter because it has the abrupt change at both edges (see Figure 6.5 and 6.6). Also, without the abrupt changes at the edges, the optimum filter would have a better high frequency characteristic because abrupt changes in sequences appear as high frequency components in the frequency domain. The second lobes of both FIR filter frequency responses are higher than that of the Optimal FIR filter. Increasing the number of knots for the splines or using higher order polynomials may reduce the second lobes.

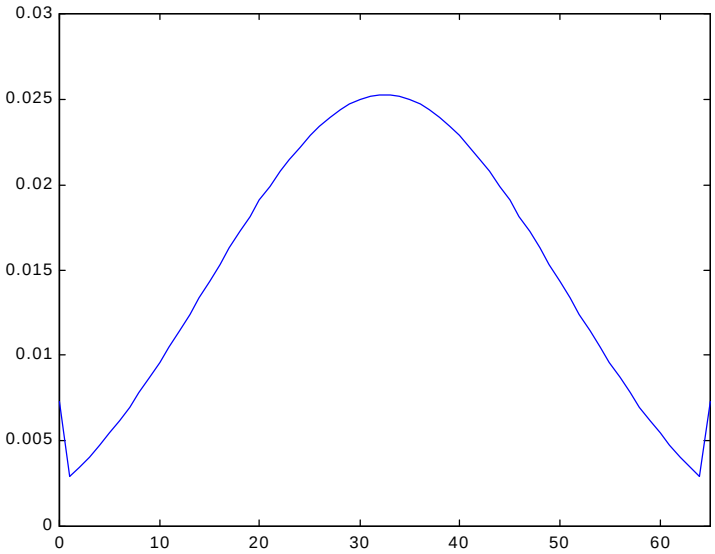
The specification of the second optimum lowpass FIR filter is the passband frequency $f_p = 0.0404$, the stopband frequency $f_s = 0.0556$, equal error weights and the filter duration $N = 99$. The impulse response and the frequency response of the filter are shown in Figure 6.7. The impulse responses and the frequency responses of its polynomial approximated filter and polynomial spline approximated filter are shown in Figure 6.8 and 6.9.

From Table 6.2, for the slow varying filter (the first filter), the polynomial spline approximated filter has better error characteristics than the polynomial approximated filter at similar approximation level in the sense of the alternation. However, for the relatively fast changing filter (the second), there is not much difference between the two, although the polynomial spline approximated filter requires less computation (see Table 6.1). This is because of the smaller number of data points on the discrete function compared to the high variation sequence values. Thus, more knots for a spline or higher order polynomials are required to achieve a smaller error approximation of the relatively fast variation filter sequences.

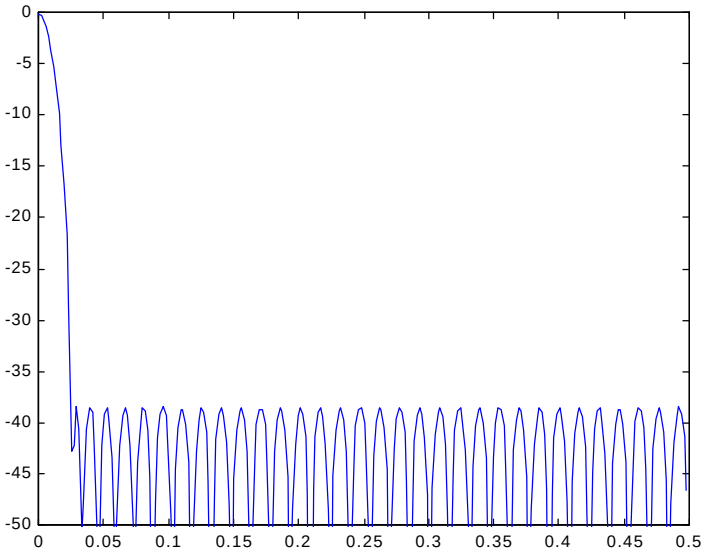
Filter Spec.	Filters	Passband Error(dB)	Stopband Error(dB)	No. Of Delay	No. Of Mult.	No. Of Add.	Overall HW Cmplx.*
$f_p = 0.001$ $N = 65$ $f_s = 0.025$	Opt. Filter	.33	-38.417	64	65	64	$65w^2 + 128w$
	Poly.	.34	-29.056	726	22	34	$22w^2 + 748w$
	Spline	.34	-37.486	321	20	27	$20w^2 + 321w$
$f_p = .0404$ $f_s = .0556$ $N = 99$	Opt. Filter	.36	-31.771	98	99	98	$99w^2 + 196w$
	Poly.	.37	-19.106	1882	38	58	$38w^2 + 1882w$
	Spline	.36	-19.960	491	30	37	$30w^2 + 491w$

* w represents the hardware complexity of an adder. Roughly, the hardware complexity of a multiplier is about w^2 and the one of a delay is about w. Thus, the overall complexity measures in the table are representing the complexity very roughly.

Table 6.2 The implementation required resource comparison among the filters

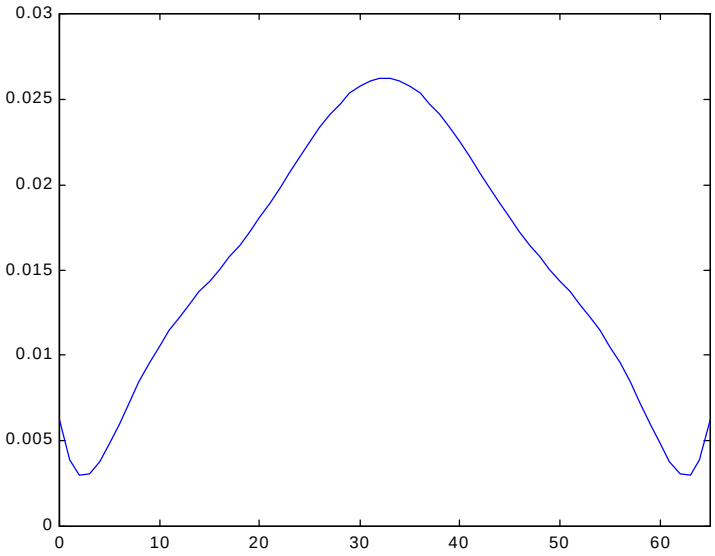


a)

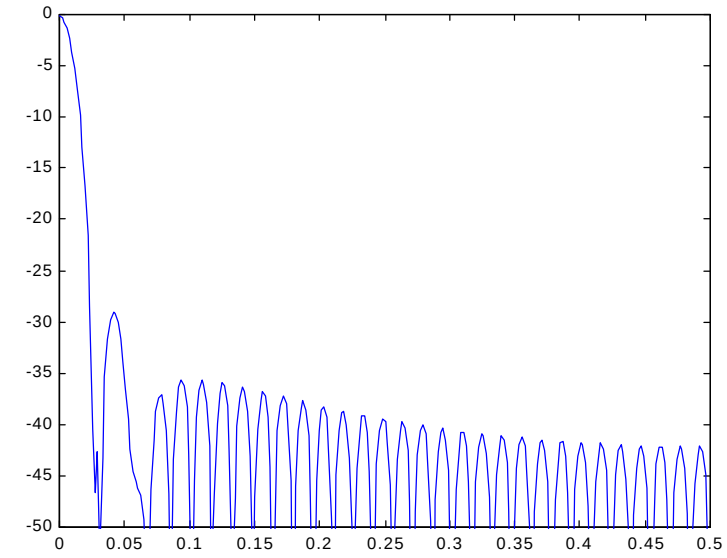


b)

Figure 6.4 The first optimal lowpass filter a) its impulse response and b) its frequency response in dB

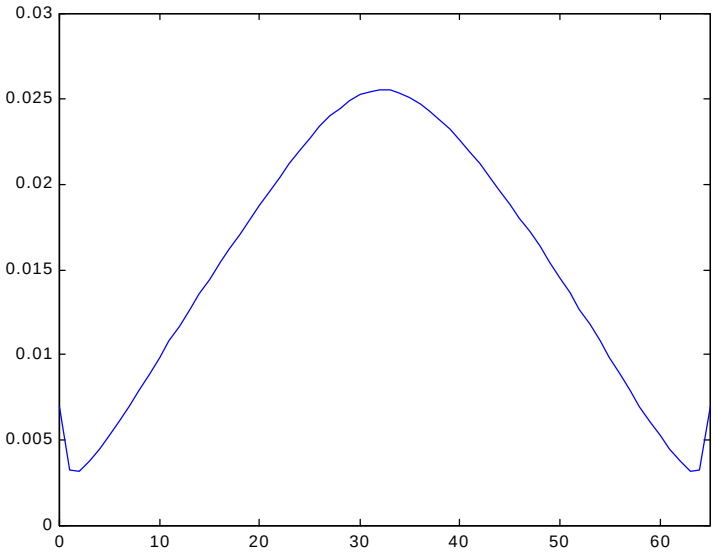


a)

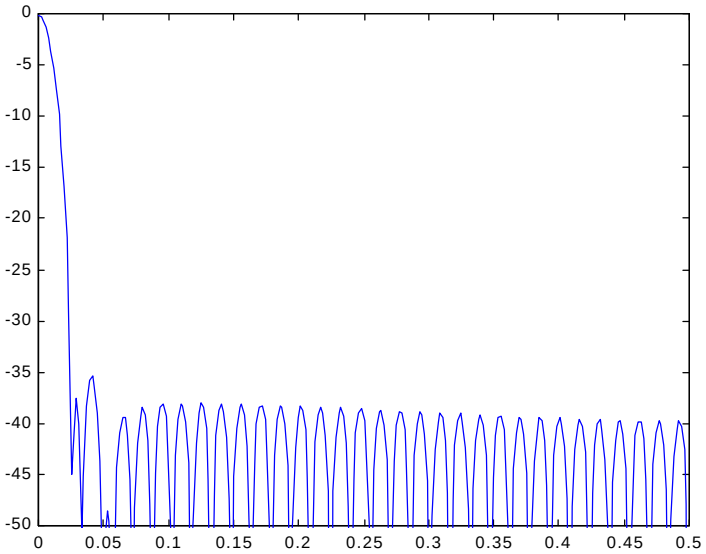


b)

Figure 6.5 The 11th order polynomial approximated filter of the first optimal lowpass filter a) its impulse response and b) its frequency response in dB

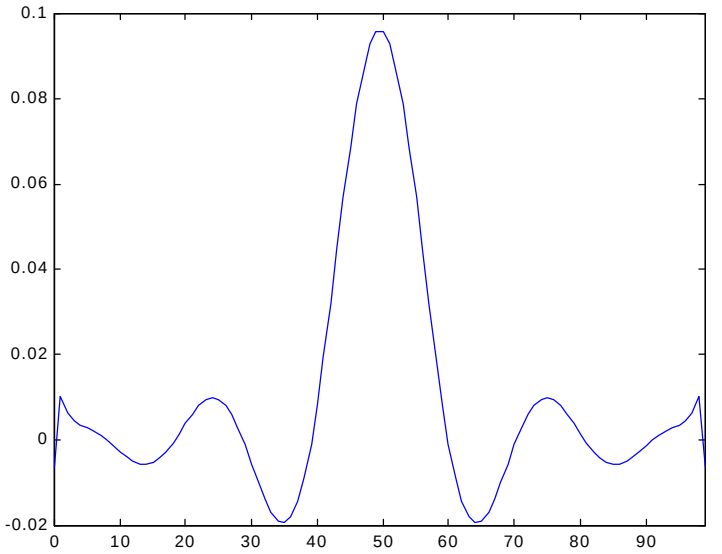


a)

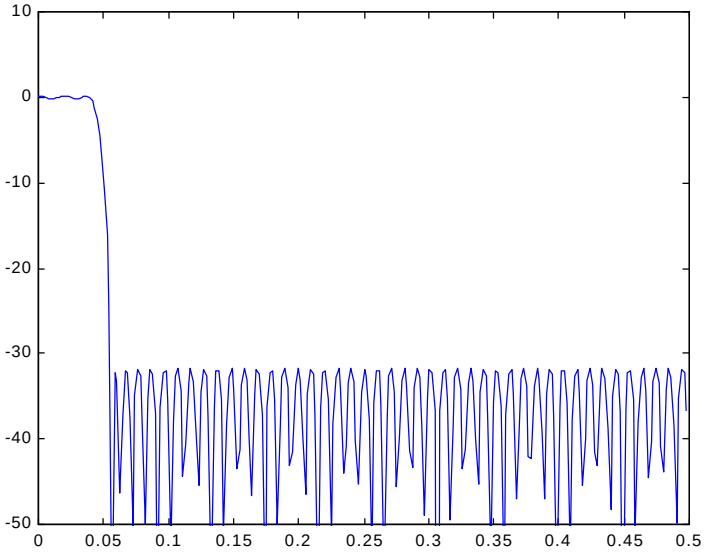


b)

Figure 6.6 The 11 knot of the 4^{th} order polynomial spline approximated filter of the first optimal lowpass filter a) its impulse response and b) its frequency response in dB

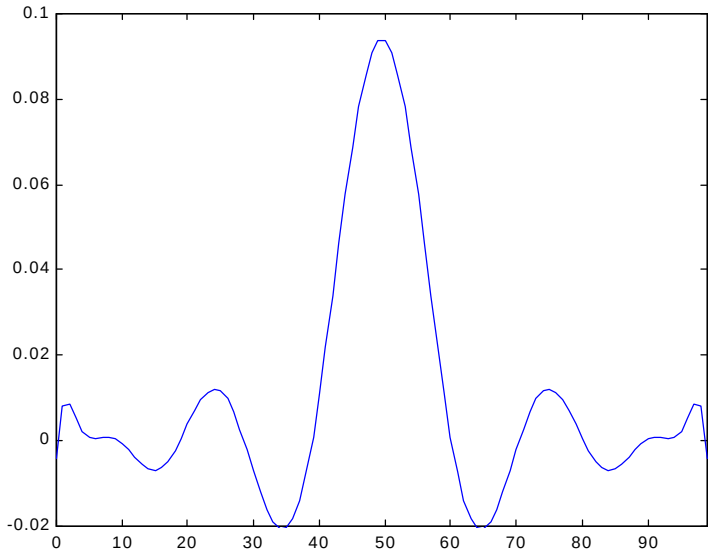


a)

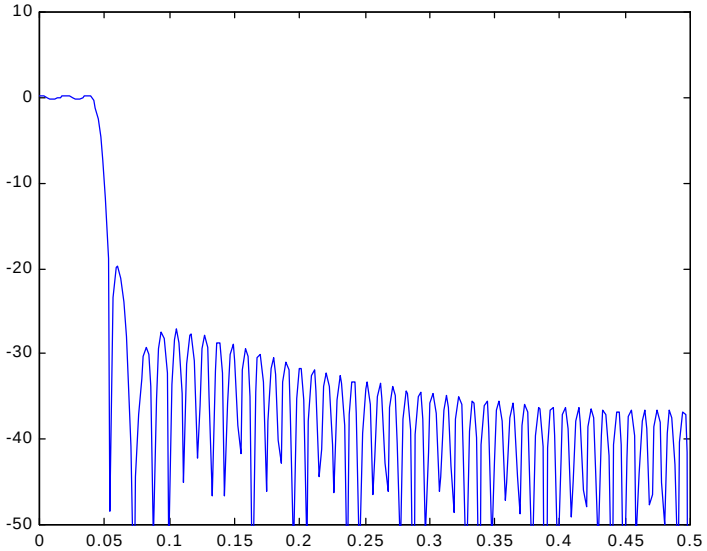


b)

Figure 6.7 The second optimal lowpass filter: a) its impulse response and b) its frequency response in dB

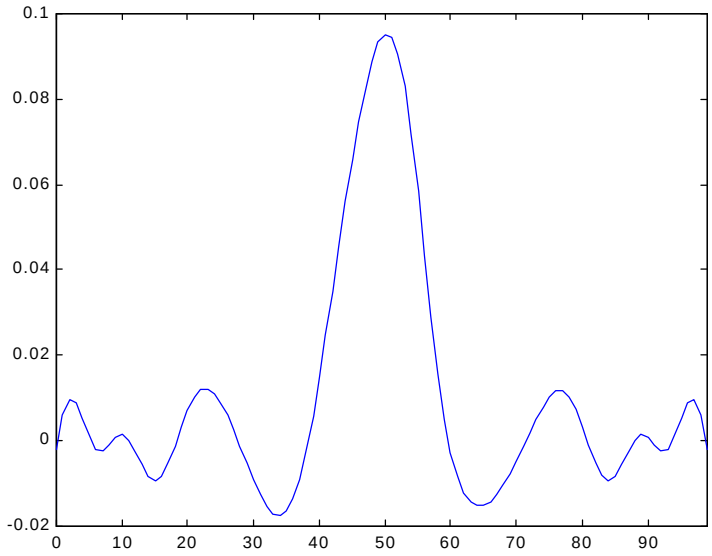


a)

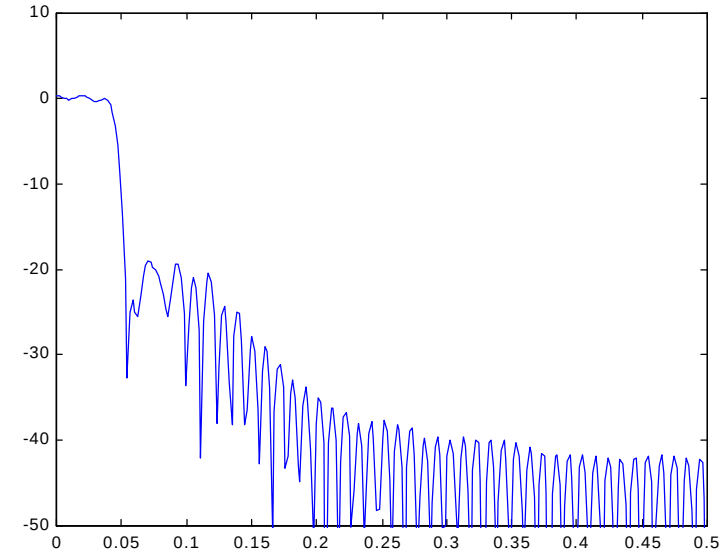


b)

Figure 6.8 The 19th order polynomial approximated filter of the first optimal lowpass filter a) its impulse response and b) its frequency response in dB



a)

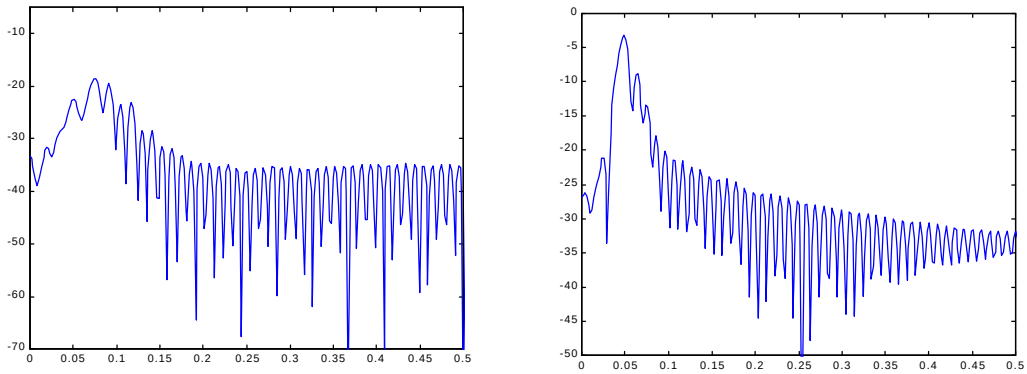


b)

Figure 6.9 The 21 knot of the 4th order polynomial spline approximated filter of the first optimal lowpass filter a) its impulse response and b) its frequency response in dB

6.6 Error Spectrum

The filter's frequency spectrum is the sum of the original filter's spectrum and that of the error. Since any Chebyshev approximation generates equiripple error solutions, the approximated filters overall frequency spectrum is controllable to some degree. The frequency behavior of an equiripple error is similar to a sinusoidal function. Because it is alternating around zero, the DC component in the frequency spectrum is very small and the number of alternations can be taken as the main frequency of the error spectrum. The higher order (or more knots) for any Chebyshev approximation not only lowers the maximum error, but also pushes the main frequency of the spectrum higher (see Figure 6.10 for an example). This may not give full freedom of control but gives some control on the overall filter spectrum.



a) b)

Figure 6.10 a) 21 knot – 4th order polynomial spline error spectrum in dB. Main frequency is around 0.08 and b) 11 knot – 4th order polynomial spline

Chapter 7

Conclusion

7.1 Summary

This dissertation focuses on applications of various impulse/summation algorithms developed by Ferrari[5], Üstüner[21], Sankar[33] and Wang[34]. A new version of the impulse/summation algorithm was developed in Chapter 3 for the computation of polynomials. It is based upon Üstüner's one-sided discrete factorial functions instead of the one-sided discrete power functions and is computationally more efficient for higher order polynomials. To reduce the implementation complexity, a new one-sided discrete factorial function class is obtained by normalization of the original class. The implementation issues of the impulse/summation algorithms for calculating polynomials, polynomial segments, B-splines, and polynomial splines are discussed and examples are shown.

Using by-products of the impulse/summation algorithm, the first differences, a new approach to reducing the computation requirements for surface normals at the sampled grids is presented in Chapter 4. Simple shading models could lead to the use of the impulse/summation algorithm in computer graphics. The surface normal requires a square root function for the normalization. Use of a search table simplifies the square root function implementation with a minimum cost in memory. The idea can be extended for the elimination of a division from the calculations. The center differences can be used to calculate surface normal for the sophisticate shading methods (Gouraud and Phong) and improves the shading quality. It is shown in the same chapter that, when the impulse/summation algorithm is used for shading, by reducing the sample data grid size,

we obtained results comparable to the more the sophisticated shading methods while at the same time significantly decreasing the computation. Changing the sample data grid size affects the coefficients of the impulse/summation algorithm. The relationship between the size change and coefficient change is the ratio of the size increase to the dimensionality.

For polynomials, it is proven that there is always a unique best Chebyshev approximation in the discrete domain. Some modifications of the usual Reméz Exchange Algorithm are required to obtain the approximation for polynomials and polynomial splines in the discrete domain. In Chapter 6, Chebyshev discrete polynomial and Chebyshev discrete polynomial splines are used to reduce the implementation complexity for FIR filters.

7.2 Future Work

All examples in this dissertation are implemented using Matlab. As shown in Section 3.5.1 and 3.5.2, the speed of the algorithm can be increased by orders of magnitude with its VLSI implementation. The algorithm speed can be measured as $\boldsymbol{O}(n)$, where n is the length of input data stream.

Chebyshev discrete polynomial approximation has a unique best approximation and an algorithm exists to find the solution. While this is not the case for Chebyshev discrete polynomial spline approximation, significant improvements are expected over the pure polynomial case. Therefore, even if a best solution may not exist, it is worth looking for because, while the results of polynomial spline approximation shown in Chapter 5 are not optimal, the solution is better than the optimal result obtained in polynomial approximation. Finally we proved that there is no way to obtain a best approximation solution for polynomial spline.

Chebyshev approximation can be effective only in a system, which has a single path for moving from one point to another in the system space[14]. Thus, it is not applicable to systems of higher order dimensionality than two, unless the function is completely

separable into independent functions for each axis (this constraint satisfies the only one path requirement). This prevents its usage for optimal surface approximation in computer graphics. However, approximation of a surface may be worth seeking in many applications, which do not require accuracy, e.g., graphic art. For wireframe computer graphics, the optimal approximation can be sought using Chebyshev approaches because a curved line in a 3-D space can be expressed with three independent functions of the 3-D coordinates in addition to a one-dimensional parametric space. If the errors of the line can be confined within a voxel of the 3-D space, the line appears on a display device without any errors. This idea may be expanded to a compression algorithm for a 3-D object, whose data are collected with a voxel map, into a wireframe-object without any loss of information.

Appendix A

Proof for Theorem 3.1

To have a unique a_i' for any order $k > 0$ of the polynomial $P(t)$, the transform between the bases of the set of x^i function and the bases of the set of $(x-h)^i$ function should be a Similarity Transformation. By binomial series expansion,

$$(t-h)^i = t^i + \binom{i}{1} t^{i-1}(-h) + \binom{i}{2} t^{i-2}(-h)^2 + \cdots + \binom{i}{i-1} t(-h)^{i-1} + (-h)^i \qquad \text{(A.1)}$$

Thus,

$$\begin{bmatrix} (t-h)^0 \\ (t-h)^1 \\ (t-h)^2 \\ \vdots \\ (t-h)^{k-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -h & 1 & 0 & & 0 \\ (-h)^2 & -2h & 1 & & 0 \\ \vdots & & & \ddots & \vdots \\ (-h)^{k-1} & & \cdots & & 1 \end{bmatrix} \begin{bmatrix} t^0 \\ t^1 \\ t^2 \\ \vdots \\ t^{k-1} \end{bmatrix} \qquad \text{(A.2)}$$

$$T' = AT$$

Since A is a triangular matrix with ones for the diagonal elements the determinant of A is non-zero. Thus, there always exists a unique A^{-1} for any real number h. Therefore, A is a similarity transformation operator.

Appendix B

Coefficient Relationship

Equation 3.12 and 3.12 are identical on $[t_1, t_m]$. Thus,

$$\sum_{i=0}^{k-1} a_i t^i + \sum_{i=2}^{m-1} c_i (t-t_i)_+^{k-1} = \sum_{i=0}^{k-2} a_i' (t-t_1)_+^i + \sum_{i=1}^{m-1} c_i' (t-t_i)_+^{k-1} \tag{B.1}$$

From above equation, it is clearly that $c_2' = c_2, c_3' = c_3, \cdots, c_{m-1}' = c_{m-1}$ and

$$\sum_{i=0}^{k-1} a_i t^i = \sum_{i=0}^{k-2} a_i' (t-t_1)_+^i + c_1' (t-t_1)_+^{k-1} \tag{B.2}$$

Equation B.2 can be written with a matrix equation as below

$$CT = C'T' \tag{B.3}$$

where $C = [a_0, a_1, \cdots, a_{k-1}]$, $T = [t^0, t^1, \cdots, t^{k-1}]^{\mathrm{T}}$, $C' = [a_0', a_1', \cdots, a_{k-2}', c_1']$ and $T' = [(t-t_1)^0, (t-t_1)^1, \cdots, (t-t_1)^{k-1}]^{\mathrm{T}}$. If we substitute h with t_1 in Equation A.2, the base of the right side of Equation B.2 and the base of Equation A.2 are same. Thus,

$$\begin{aligned} & CT = C'AT \\ \Rightarrow\ & C = C'A \\ \Rightarrow\ & C' = CA^{-1} \\ \Rightarrow\ & C'^{\mathrm{T}} = A^{-\mathrm{T}} C^{\mathrm{T}} \end{aligned} \tag{B.4}$$

Therefore,

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{k-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -t_1 & 1 & 0 & & 0 \\ (-t_1)^2 & -2t_1 & 1 & & 0 \\ \vdots & & & \ddots & \vdots \\ (-t_1)^{k-1} & & \cdots & & 1 \end{bmatrix}^{-\mathrm{T}} \begin{bmatrix} a_0' \\ a_1' \\ \vdots \\ a_{k-2}' \\ c_1' \end{bmatrix} \tag{B.5}$$

Appendix C

Getting Coefficients of the One-Sided Polynomial Expression for polynomial segment

By Section 3.3.1, Equation 3.20 can be written into

$$s_i(t) = O_{i,f}(t) - O_{i,b}(t) \tag{C.1}$$

where $O_{i,f}(t)$ is one of the two one-sided polynomials for $s_i(t)$, which starts at knot t_i and $O_{i,b}(t)$ is the other one-sided polynomials for $s_i(t)$, which starts at knot t_{i+1}. With the transformation A defined in Appendix A, the coefficients f_{ij}'s and b_{ij}'s for $O_{i,f}(t)$ and $O_{i,b}(t)$ can be obtained from a_{ij}'s of Section 3.3.4. Since $S(t)$ is the sum of $s_i(t)$'s

$$\begin{aligned} S(t) = {} & O_{1,f}(t) - O_{1,b}(t) + O_{2,f}(t) - O_{2,b}(t) + \\ & \cdots + O_{m-1,f}(t) - O_{m-1,b}(t) \end{aligned} \tag{C.2}$$

$S(t)$ can be rearranged into

$$\begin{aligned} S(t) = {} & O_{1,f}(t) - (O_{1,b}(t) - O_{2,f}(t)) - (O_{2,b}(t) - O_{3,f}(t)) - \\ & \cdots - (O_{m-2,b}(t) - O_{m-1,f}(t)) - O_{m-1,b}(t) \end{aligned} \tag{C.3}$$

By the definition of $O_{i,f}(t)$ and $O_{i,b}(t)$,

$$
\begin{aligned}
S(t) = & \sum_{j=0}^{k-1} f_{1,j}(t-t_1)_+^j \\
& - \sum_{i=1}^{m-2} \left(\sum_{j=0}^{k-1} (b_{ij} - f_{i+1,j})(t-t_i)_+^j \right) - \sum_{j=0}^{k-1} b_{m-1,j}(t-t_m)_+^j
\end{aligned}
\tag{C.4}
$$

Since Equation C.4 and 3.21 are the same,

$$
\begin{aligned}
a'_{1,j} &= f_{1,j} \\
a'_{i,j} &= -(b_{i-1,j} - f_{i,j}) \\
a'_{m,j} &= -b_{m-1,j}
\end{aligned}
\tag{C.5}
$$

for $i = 2,3,\cdots,m-1$ and $i = 0,1,\cdots,k-1$.

Appendix D

Proof of Theorem 3.2

Before we prove Theorem 3.2, it is necessary to prove that the repeated summation operator is linear because any discrete function $f[n]$ can be expressed with the summations of weighted impulses as

$$f[n] = \sum_{i=-\infty}^{\infty} a_i d[n-i] \tag{E.1}$$

For example, $\nabla^4 \tilde{b}_4[n] = d[n-1] + 4d[n-2] + d[n-3]$ and $\tilde{b}_4[n] = (n)_+^3$.

Let's $F_1[n] = (f_1[n])_{[k]}$ and $F_2[n] = (f_2[n])_{[k]}$ where any discrete functions $f_1[n]$ and $f_2[n]$ and apply the repeated summation operator on $af_1[n] + bf_2[n]$, a and b are real numbers. Then,

$$\begin{aligned} y[n] &= (af_1[n] + bf_2[n])_{[k]} \\ &= \sum_{r_k=0}^{n} \cdots \sum_{r_2=0}^{r_3} \sum_{r_1=0}^{r_2} (af_1[n] + bf_2[n]) \\ &= \sum_{r_k=0}^{n} \cdots \sum_{r_2=0}^{r_3} \left(a \sum_{r_1=0}^{r_2} f_1[n] + b \sum_{r_1=0}^{r_2} f_2[n] \right) \\ &= a \sum_{r_k=0}^{n} \cdots \sum_{r_2=0}^{r_3} \sum_{r_1=0}^{r_2} f_1[n] + b \sum_{r_k=0}^{n} \cdots \sum_{r_2=0}^{r_3} \sum_{r_1=0}^{r_2} f_2[n] \end{aligned} \tag{E.2}$$

By the definition of the repeated summation,

$$y[n] = aF_1[n] + bF_2[n] \tag{E.3}$$

Therefore the repeated summation is a linear operator.

Since it is linear operator, it is sufficient to prove the shift invariance of the operator on a impulse for the proof of its shift invariant property. Thus, consider a k^{th} one-sided factorial function $b_k[n]$ and $b_k[n-n_0]$. Let's apply the forward difference to $b_k[n-n_0]$. Then,

$$\Delta b_k[n-n_0] = b_k[n-n_0+1] - b_k[n-n_0] \tag{E.4}$$

By the definition of the one-sided factorial function (Equation 3.33),

$$\begin{aligned}
\Delta b_k[n-n_0] &= \left(\frac{(n-n_0+1)!}{k!(n-n_0+1-k)!} - \frac{(n-n_0)!}{k!(n-n_0-k)!} \right) && \text{for } n-n_0 \geq 0 \\
&= \frac{n!(n-n_0+1-(n-n_0+1-k))}{k!(n-n_0+1-k)!} && \text{for } n-n_0 \geq 0 \\
&= \left(\frac{(n-n_0)!}{(k-1)!(n-n_0-(k-1))!} \right) && \text{for } n-n_0 \geq 0 \\
&= b_{k-1}[n-n_0]
\end{aligned} \tag{E.5}$$

Equation E.5 shows the same recursive relationship between $b_k[n]$ and $b_{k-1}[n]$ for $b_k[n-n_0]$'s. Thus, the repeated summation can generate $b_k[n-n_0]$ from $b_{-1}[n-n_0] = d[n-n_0+1]$. Therefore, the repeated summation is a shift invariant operator.

Appendix E

Transformation between One-sided Power Function Bases and One-sided Factorial Function Bases

From Equation 3.26,

$$\begin{bmatrix} b_0[n] \\ b_1[n] \\ b_2[n] \\ b_3[n] \\ b_4[n] \\ \vdots \\ b_{k-1}[n] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & -1/2 & 1/2 & 0 & & \cdots & 0 \\ 0 & 2/6 & -3/6 & 1/6 & 0 & \cdots & 0 \\ 0 & -6/24 & 11/24 & -6/24 & 1/24 & & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \ddots \end{bmatrix} \begin{bmatrix} n^0 \\ n^1 \\ n^2 \\ n^3 \\ n^4 \\ \vdots \\ n^{k-1} \end{bmatrix} \tag{F.1}$$

$$B = T\tilde{B}$$

and Equation 3.5 can be re-written

$$P[n] = A\tilde{B} = A'B \tag{F.2}$$

where $A = [a_0, a_1, \cdots, a_{k-1}]$ and $A' = [a_0', a_1', \cdots, a_{k-1}']$. Thus,

$$\begin{aligned} & A\tilde{B} = A'T\tilde{B} \\ \Rightarrow \quad & A = A'T \\ \Rightarrow \quad & A' = AT^{-1} \\ \Rightarrow \quad & A'^{T} = T^{-T}A^{T} \end{aligned}$$

where

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 2 & 0 & & \cdots & 0 \\ 0 & 1 & 6 & 6 & 0 & \cdots & 0 \\ 0 & 1 & 14 & 36 & 24 & & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \ddots \end{bmatrix}$$

Appendix F

Surface Normal

As discussed in Section 4.3, the surface normal $\vec{N}$ at the point P and the plane T, which passes P, are perpendicular each other because T is the tangent plane of surface S at P. Also, the cross product of any two vectors from T will generate a vector, which is perpendicular with the plane T, if the two vectors are not aligned with the same direction. Let's pick $\vec{T}_x$ and $\vec{T}_y$. Since they are on T and obviously are pointing different directions,

$$\vec{N} = \frac{\vec{T}_x \times \vec{T}_y}{\left|\vec{T}_x \times \vec{T}_y\right|} \tag{F.1}$$

From b) of Figure F.1,

$$\begin{aligned} \vec{T}_x &= \lim_{\Delta x \to 0} [\Delta x \cdot \vec{x}, 0 \cdot \vec{y}, \Delta S \cdot \vec{z}] \\ &= \lim_{\Delta x \to 0} \left[1 \cdot \vec{x}, 0 \cdot \vec{y}, \frac{\Delta S}{\Delta x} \cdot \vec{z} \right] \\ &= \left[1 \cdot \vec{x}, 0 \cdot \vec{y}, \frac{\partial S}{\partial x} \cdot \vec{z} \right] \end{aligned} \tag{F.2}$$

Similarly,

$$\vec{T}_y = \left[0, 1, \frac{\partial S}{\partial y} \right] \tag{F.3}$$

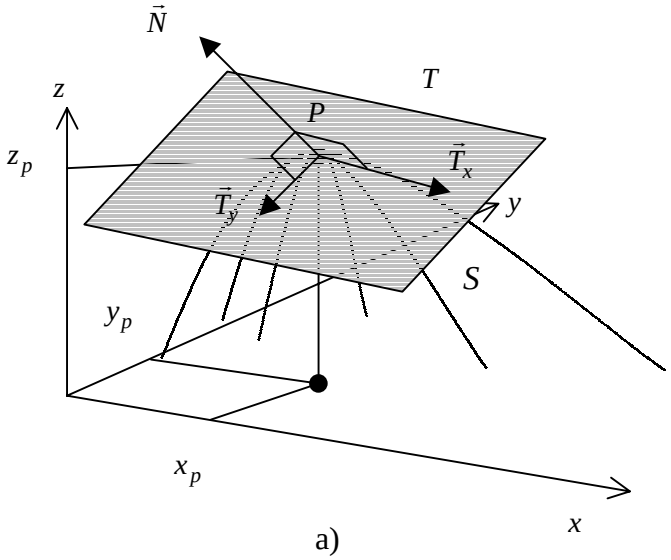
From Equation F.2 and F.3,

$$\vec{T}_x \times \vec{T}_y = \left[-\frac{\partial S}{\partial x} \cdot \vec{x}, -\frac{\partial S}{\partial y} \cdot \vec{y}, 1 \cdot \vec{z} \right] \tag{F.4}$$

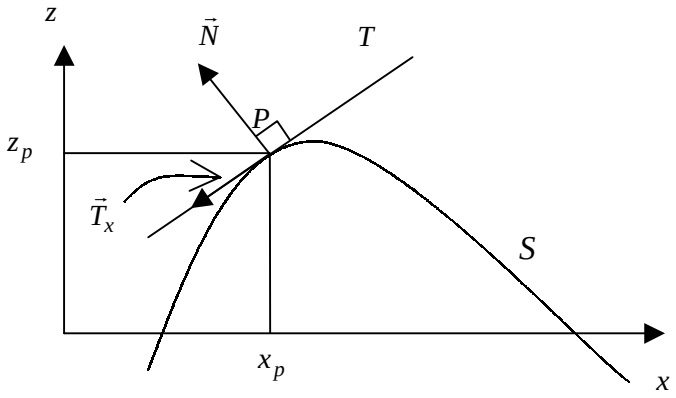
Thus,

$$\vec{N} = \frac{\left[-\frac{\partial S}{\partial x} \cdot \vec{x}, -\frac{\partial S}{\partial y} \cdot \vec{y}, 1 \cdot \vec{z} \right]}{\sqrt{\left(\frac{\partial S}{\partial x} \right)^2 + \left(\frac{\partial S}{\partial y} \right)^2 + 1}} \tag{F.5}$$

at the point P.



a)



b)

Figure F.1 Surface and tangent plane example: a) 3-D view and b) cross-sectional view at $y = y_p$

Appendix G

Proofs for Theorem 4.1

i) From Theorem 4.1,

$$\begin{aligned} \Diamond f[n] &= \frac{(f[n+1]-f[n-1])}{2} \\ &= \frac{(f[n+1]-f[n]+f[n]-f[n-1])}{2} \\ &= \frac{\Delta f[n]+\nabla f[n]}{2} \end{aligned} \tag{G.1}$$

Thus, the center difference of $f[n]$ is an average of $\Delta f[n]$ and $\nabla f[n]$. An average of two numbers is always between the two. Thus, $\Delta f[n] \le \Diamond f[n] \le \nabla f[n]$, when $\Delta f[n] \le \nabla f[n]$, otherwise, $\nabla f[n] \le \Diamond f[n] \le \Delta f[n]$. Therefore, the statement i) in Theorem 4.1 is true.

ii) The statement ii) in Theorem 4.1 can be re-stated as, always

$$\left| f'(n) - \Diamond f[n] \right| \le \left| f'(n) - \Delta f[n] \right| \tag{G.2}$$

and

$$\left| f'(n) - \Diamond f[n] \right| \le \left| f'(n) - \nabla f[n] \right| \tag{G.3}$$

for any smooth function $f(t)$.

By Taylor Expansion,

$$f(n+1) = f(n) + f'(n) + \frac{f''(n)}{2!} + \frac{f'''(n)}{3!} + \cdots$$

and (G.4)

$$f(n-1) = f(n) - f'(n) + \frac{f''(n)}{2!} - \frac{f'''(n)}{3!} + \cdots$$

Since $\Delta t = 1,\ f[n] = f(n)$. Thus,

$$\Delta f[n] = f(n+1) - f(n) = f'(n) + \frac{f''(n)}{2!} + \frac{f'''(n)}{3!} + \cdots$$

$$\Rightarrow \quad |f'(n) - \Delta f[n]| = \left| \frac{f''(n)}{2!} + \frac{f'''(n)}{3!} + \cdots \right|$$

and (G.5)

$$\Diamond f[n] = f(n+1) - f(n-1) = f'(n) + \frac{f'''(n)}{3!} + \frac{f^{[5]}(n)}{5!} + \cdots$$

$$\Rightarrow \quad |f'(n) - \Diamond f[n]| = \left| \frac{f'''(n)}{3!} + \frac{f^{[5]}(n)}{5!} + \cdots \right|$$

Since

$$f'(n) - \Delta f[n] = (f'(n) - \Diamond f[n]) + \frac{f''(n)}{2!} + \frac{f^{[4]}(n)}{4!} + \cdots$$

$|f'(n) - \Diamond f[n]| \le |f'(n) - \Delta f[n]|$ is always true. Similarly, it can be proved for $|f'(n) - \Diamond f[n]| \le |f'(n) - \nabla f[n]|$. Therefore, the center difference values of $f[n]$ are always closer to $f'(n)$ than two the other differences' values.

Appendix H

Proofs for Theorem 6.1

Since $x[n]$ is a sequence of numbers, $x[n]$ can be broken into a sum of the impulses weighted by values of $x[n]$ at the impulse location. Thus, $x[n]$ can be written as

$$x[n] = \sum_{i=-\infty}^{\infty} x[i]d[n-i] \tag{H.1}$$

From Equation I.1,

$$(I[n] * x[n])_{[k]} = \left(I[n] * \left(\sum_{i=-\infty}^{\infty} x[i]d[n-i] \right) \right)_{[k]} \tag{H.2}$$

Since the convolution is a linear shift invariant operator[52],

$$(I[n] * x[n])_{[k]} = \left(\sum_{i=-\infty}^{\infty} x[i]\left(I[n] * d[n-i]\right) \right)_{[k]} \tag{H.3}$$

Also, from Theorem 3.2, the repeated summation is a linear shift invariant operator. Thus,

$$(I[n] * x[n])_{[k]} = \sum_{i=-\infty}^{\infty} x[i]\left(\left(I[n] * d[n-i]\right)\right)_{[k]} \tag{H.4}$$

Since $I[n]*d[n-i]=I[n-i]$,

$$\begin{aligned}(I[n]*x[n])_{[k]} &= \sum_{i=-\infty}^{\infty} x[i]\left(I[n-i]\right)_{[k]} \\ &= \sum_{i=-\infty}^{\infty} x[i]h[n-i] \\ &= x[n]*h[n]\end{aligned} \tag{H.5}$$

The convolution operator is commutative[56]. Therefore,

$$h[n]*x[n]=(I[n]*x[n])_{[k]} \tag{H.6}$$

References

1 Schoenberg, I. J., "Contributions to the problem of approximation of equidistant data by analytic functions", Quarterly of Applied Mathematics, 4, pp49-99, 1946.

2 Zadunaisky, P. E. and Lafferriere, G., "On an iterative improvement of the approximate solution of some ordinary differential equations Computers & Mathematics with Applications, vol.6, no.1, p.147-54, 1980.

3 Prenter P. M., "Splines and Variational Methods", John Wiley & Sons, Inc., New York, 1975.

4 Lancaster P., and Salkauskas, K., "Curve and Surface Fitting: An Introduction ", Academic Press, 1986.

5 Ferrari, L. A., Silbermann, M. J. & Sankar, P. V., "An Efficient Algorithm of the Implementation of General B-Splines", CVGIP, vol. 56 no. 1, pp102-105, Jan 1994.

6 Ross, K. A., "Elementary Analysis: The Theory of Calculus", Springer-Verlag, New York, 1980.

7 Jupp D. L. B., "Approximation to data by splines with free knots", SIAM Journal on Numerical Analysis, Vol. 15, no. 2, p.328-43, April 1978.

8 Cox, M. G., "Curve fitting with piecewise polynomials", Journal of the Institutes of Mathematics and its Application, vol. 8, no.1, p.36-52, August 1971.

9 Hu, Y., "An algorithm for data reduction using splines with free knots", IMA Journal of Numerical Analysis, vol. 13, no.3, p.365-81, July 1993.

10 Rice, J. R., "The Approximation of functions", Vol. I, Addison-Wesley Publishing Company, INC., London, 1969.

11 Rivlin, T. J., "Chebyshev Polynomials", John Wiley & Sons, INC., New York, 1990.

12 Rice, J. R., "The Approximation of functions", Vol. II, Addison-Wesley Publishing Company, INC., London, 1969.

13 Kalin, S. & Studden, W. J., "Tchbycheff Systems: With Applications In Analysis and Statistics", John Wiley & Sons, Inc., New York, 1966

14 Watson, A. G., "Approximation Theory and Numerical Methods", John Wiley & Sons, LTD., New York, 1980.

15 Snyder, M. A., "Chebyshev Methods in Nummerical Approximation ", Prentice-Hall, Englewood Cliff, 1966.

16 Gass, S. I., "Linear Programming: Methods and Applications", McGraw-Hill, New York, 1971.

17 Nürnberger, G. and Sommer, M., "A Remez Type Algorithm for Spline Function", Numer. Math., 41, p117-146, 1983.

18 Braess, D., "Chebyshev Approximation by Spline Functions with Free Knots", Numer. Math., 17, p357-366, 1971.

19 Karon, J. M., "Computing Improved Chebyshev Approximations By the Continuation Method. I: Description of an Algorithm", SIAM J. Number. Analy., V13, No.6, December, 1978.

20 Cox, M. G., "A survey of numerical methods for data and function approximation. In The state of the art in numerical analysis: proceedings of the Conference on the State of the Art in Numerical Analysis", Academic Press, Inc. 1977.

21 Üstüner, K. F., "Discrete Polynomial Splines: Application in Computer Graphics and Digital Signal Processing", Dissertation, University of California, Irvine, 1989.

22 Yakowitz, S. & Szidarovszky, F., "An Introduction to Numerical Computations", Macmillan Publishing Company, New York, 1989.

23 Schumaker, L. L., "Spline Functions: Basic Theory", John Wiley & Sons, Inc., New York, 1981.

24 Mangasarian, O. L. & Schumaker, L. L., "Discrete Splines via Mathematical Programming", SAIM J. Control, 9, pp. 174-183, 1971.

25 Ferrari, L. A., Silbermann, M. J. & Sankar, P. V., "An Efficient Algorithm of the Implementation of General B-Splines", CVGIP, vol. 56 no. 1, pp102-105, Jan 1994.

26 de Boor, C., "A Practical Guide to Splines", Springer-Verlag, New York, 1978.

27 Ferrari, L. A., Park, J. H., Healey, A. and Leeman, S., "Interpolation Using a Fast Spline Transform(FST)", IEEE Transaction of Circuit and Systems, will appear in Aug., 1999

28 Jones, R. C. & Karlovitz, L. A., "Equioscillation under Nonuniqueness in the Approximation of Continuous Functions", Journal of Approximation Theory 3 pp.138-145, 1970.

29 Cox, M. A., "The Numerical Evaluation of B-Splines", Journal of Inst. of Mathematics and Its Application, 10, pp134-149, 1972.

30 de Boor, C., "On Calculating B-splines", Journal of Approximation Theory, 6, pp50-62, 1972.

31 Bartels, B. H., Beatty, J. C., and Barsky, B. A., "An Introduction to Splines For Use in Computer Graphics & Geometric Modeling ", Morgan Kaufmann Publisher, Inc., Los Altos, CA, 1987.

32 de Boor, C., "On uniform Approximation By Splines", J. Approximaton Theory, 1, p219-235, 1668.

33 Sankar, P. V., Silvermann, M. J., and Ferrari, L.A., "Curve and Surface Generation and Refinement Based on a High Speed Derivative Algorithm", CVGIP, vol. 56 no. 1, pp94-101, Jan 1994.

34 Wang, S. Y., "High-Speed, Interactive Computer Graphics Using Spline Functions", Dissertation, University of California, Irvine, 1991.

35 Israel, K., "Computer arithmetic algorithms", Englewood Cliffs, N.J., Prentice Hall, 1993.

36 Refling, J. P., "Pipelined Implementation of B-splines and Beta-splines for Computer Graphics and other Discrete Application", Dessertation, University of California, Irvine, May 31, 1994.

37 Brualdi, R. A., "Introductory Combinatorics", North Halland, New York, 1977.

38 Sedgewick, R., "Algorithms in C", Addison-Wesley Inc., New York, 1990.

39 Foley, J. D., Dam, A.V., Feiner, S. K., and Hughes, J. F., "Computer Graphics: Princeples and Practice", 2nd edition, Addison-Wesley Inc., 1990.

40 Schoenberg, I. J. "Spline functions, convex, and mechanical quadrature", Bull. Amer. Math. Soc., 64, pp352-357, 1958.

41 Böhm, W., "Efficient Evaluation of Splines", Computing 33, pp171-177, 1984.

42 Horn, B. K., "Robert Vision", The MIT Press, McGraw-Hill, Comp., 1986.

43 Schroeder, W., Martin, K, and Lorensen, B., "The Visualization Toolkit", 2nd Edition, Prentice Hall, 1998.

44 Yang, Y. and Yuille, A., "Sources from shading", Proceedings 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Los Alamitos, p.534-9, 1991.

45 Bergman, L., Fuchs, H., Grant, E. and Spach, S., "Image rendering by adaptive refinement", Computer Graphics, v20, no.4, p.29-37, Aug. 1986.

46 Deering, M., Winner, S., Schediwy, B., Duffy, C. and Hunt, "The Triangle Processor and Normal Vector Shader: a VLSI System for High Performance Graphics", Computer Graphics, v22, no.4, p.21-30, Aug. 1988.

47 Gusfield, D., "Algorithms On Strings, Trees, and Sequences: Computer Science and Computational Biology", Cambridge University Press, New York, 1997.

48 Bittinger, M. L., "Calculus: A Modeling Approach", Addison-Wesley Inc., 1984.

49 Powell, M. J. D., "Approximation Theory and Methods", Cambridge University Press, New York, 1981.

50 Parks, T. W. and McClellan, J. H., "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase", IEEE Transaction on Circuit Theory, v.CT-19, no.2, p189-194, March, 1972.

51 Wiberg, D. M., "Theory and Problems of State Space and Linear Systems", Schaum's Outlines Series, McGraw-Hill, 1971.

52 Oppenheim, A. V. and Schafer, R. W., "Discrete-Time Signal Processing", Prentice Hall, 1989.

53 Roberts, R.A. and Mullis, C. T., "Digital Signal Processing", Addison-Wesley Inc., 1987.

54 Oppenheim, A. V. and Schafer, R. W., "Digital Signal Processing", Prentice Hall, 1975.

55 McClellan, J. H. and Parks, T. W., "A United Approach to the Design of Optimum FIR Linear-Phase Digital Filters", IEEE Transaction on Circuit Theory, v.CT-20, no.6, p697-701, November, 1973.

56 Kreyszig, E., "Advanced Engineering Mathematics, Sixth Edition", John Wiley and Sons, 1988.

Vitae

Jae Hong Park

September 1, 1964	Born in Seoul, Korea.
1987	B.S. in Textile Engineering, Seoul National University, Seoul Korea
1992	M.S. in Electrical Engineering, California State University, Fullerton, California, U.S.A.
1999	Ph.D in Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia

Publications

- Park, J. H., and Ferrari, L. A., "*The 2-5-2 Spline Functions*", Accepted in 4th International AFA Conference on Curves and Surfaces.
- Ferrari, L. A., Park, J. H., Healey, A. and Leeman, S., "*Interpolation Using a Fast Spline Transform(FST)*", IEEE Transaction of Circuit and Systems, vol. 46, no. 8, August 1999.
- Bay, J. S. and Park, J. H. "*Manual for Fundamentals of Linear State Space Systems*", McGraw-Hill, 1998.
- Ferrari, L. A. and Park, J. H. "*An Efficient Spline Basis for Multi-Dimensional Application: Image-Interpolation*", IEEE International Symposium on Circuits and System, vol. 1, pp.757-760, 1997.
- Hwang, C. and Park, J. H., "*Time Domain Mixed Pulse Signal Recognition and Application*", International Society for Mini and Microcomputers Conference in Long Beach, California, December 16-18, 1991.