2000

Multimedia applications of three-dimensional digital filters.

Steven Bruce McFadden
University of Windsor

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

NOTE TO USERS

The diskette is not included in this original manuscript. It is available for consultation at the author's graduate school library.

This reproduction is the best copy available.

UMI®

MULTIMEDIA APPLICATIONS OF THREE-DIMENSIONAL DIGITAL FILTERS

by

Steven B. McFadden

A Thesis
Submitted to the College of Graduate Studies and Research
through Electrical Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

April 2000

918468

APPROVED BY:

Dr. M. A. Sid-Ahmed (Supervisor)
Department of Electrical Engineering

Dr. J. J. Soltis
Department of Electrical Engineering

Dr. R. G. Gaspar
Department of Mechanical Engineering

Abstract

Digital signal processing has long been an extremely important field of study. One-dimensional and two-dimensional filters have applications in areas such as audio filtering or image processing respectively. As VLSI technology continues to increase, higher-dimensional digital filters are becoming more practical. This thesis investigates the application of Three-Dimensional (3-D) Digital Filters to the area of multimedia. Specifically, it investigates the use of 3-D Interpolation filters to increase the horizontal, vertical, and temporal resolution, or frame rate, of a moving image sequence.

The thesis begins by presenting the theory of digital interpolation in one dimension, and then extends that theory to three dimensions. Next the theory is presented for the design of a filter with appropriate characteristics for filtering a video image; i.e. near-linear phase and a steep transition band. After the basic theory is presented, a plan for implementing the filtering of a video image in software is presented along with the relevant file format information. Results from this implementation are shown next, and the thesis ends with a summary and conclusions

Acknowledgments

I wish to extend my thanks to my supervisor Dr. Sid-Ahmed, who planted the idea for me to pursue this degree, and then encouraged me throughout. I also wish to acknowledge my thesis committee members, Dr. Soltis, and Dr Gaspar, for their comments and advice on the thesis. Last but not least, I would like to thank my parents for their constant support of my continuing education.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Introduction

The purpose of this thesis is to investigate the multimedia application of three-dimensional (3-D) digital filters. Specifically, a digital 3-D interpolation filter is to be designed which performs inter-pixel and inter-frame interpolation, resulting in increased horizontal resolution, vertical resolution, and temporal resolution (frame rate) of a moving image sequence.

Digital video is a very common example of a moving digital image sequence, with each frame of video representing a separate two-dimensional (2-D) digital image. These images change as a function of time, and it is this temporal variation which represents the third dimension in digital video. This representation of digital video is depicted in Figure 1.1. In this figure, 'x' represents the horizontal axis, 'y' represents the vertical axis, and 't' represents the time axis. The term dt represents the inverse of the frame rate. The axis is drawn for the purpose of clarity, and the directions of positive and negative are arbitrary.

Though digital video signals such as the one shown in Figure 1.1 are inherently three-dimensional, 2-D digital filters are often used to filter such signals by processing each frame separately. This method is very practical since 2-D filters are less complex and require much less hardware than equivalent order 3-D filters. This complexity and hardware saving becomes more pronounced as the order of the filter increases. With

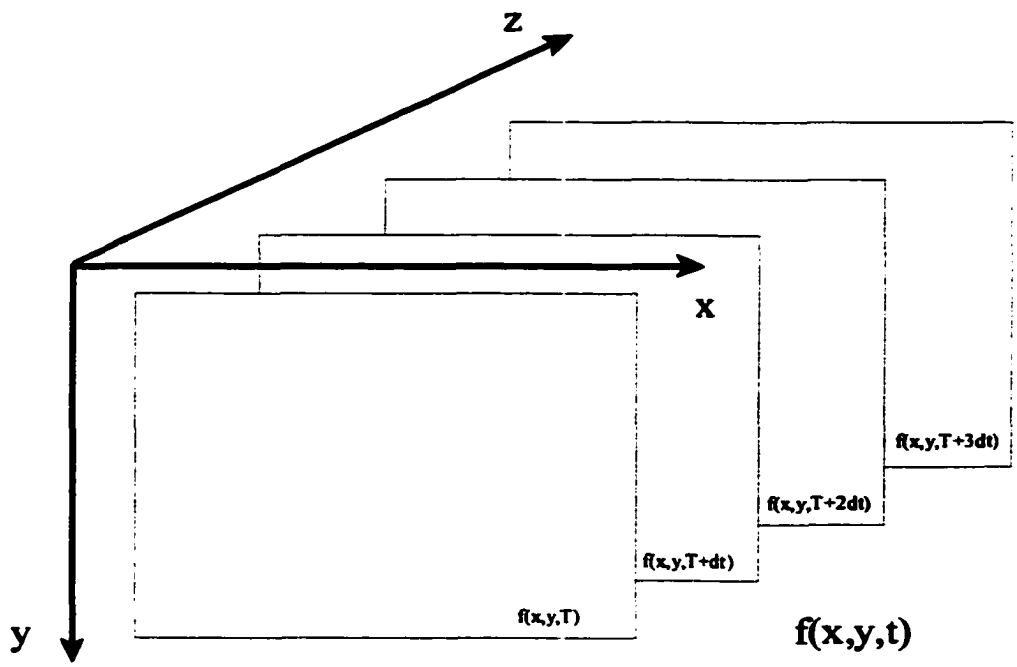


Figure 1.1: Representation of moving images

improvements in digital technology, 3-D filters are becoming more reasonable in terms of cost, and their benefits over 2-D filters are becoming more attractive. The particular benefit of interest in this thesis is the 3-D filter's ability to process temporal information in a moving picture sequence.

1.2 Digital Interpolation

Digital interpolation is a process by which a digital signal with a specific sampling rate is altered such that the frequency content of the signal remains unchanged while the sampling rate is increased. Subject to limitations specified in the Sampling Theorem, presented in Chapter 2, the sampling rate of a digital signal can be increased to any desired degree. This means that the original continuous signal is recoverable from the sampled signal. Interpolation has many applications in one-dimensional (1-D), 2-D, and 3-D digital signal processing. For example, it can be used as a method of data compression or used to improve the resolution of a signal. It may also be used to change the sampling rate of a signal for the purpose of scaling[1]. It is the improvement of signal resolution that this thesis is concerned with.

For an application such as digital audio, a 1-D digital interpolator can be used to increase the resolution of the signal, making it more closely represent the original continuous signal. In image processing applications, a 2-D digital interpolator can be used to increase the pixel resolution of a digital image in either, or both of, the horizontal and vertical directions (inter-pixel interpolation)[1]. This has the effect of making a digital image more

closely approximate the original image. The process of using digital interpolators to increase signal resolution can be extended to three dimensions. In the case of a digital video signal, the resolution of each frame can be increased in the 'x' and 'y' directions just as if it was processed using a 2-D interpolator. The 3-D interpolator has the added advantage of being capable of increasing the resolution along the time axis (the third dimension). This means that in addition to increased resolution in each frame of video, a 3-D interpolator can also increase the number of frames present in a video sequence (inter-frame interpolation). This three-dimensional interpolation of digital video is the primary goal of this thesis.

1.3 Three-Dimensional Digital Filters

A digital filter is a system that, when given a sequence of input numbers, produces a sequence of output numbers subject to a specified set of rules. Accordingly, a 3-D digital filter produces a three-dimensional array of numbers when given a three-dimensional input array. For example, when a 3-D filter is given the luminance values of a digital video sequence as an input, the output is usually an altered form of that digital video sequence. Filters of any dimension are traditionally divided into two categories: non-recursive filters and recursive filters. Non-recursive filters, also known as Finite Impulse Response (FIR) filters, produce an output which is a weighted average of present and previous input values. Recursive filters, also known as Infinite Impulse Response (IIR) filters, produce an output that is a weighted average of present and past input values as well as past output values. Each type of filter has its own advantages and disadvantages and these must be

weighed according to the individual application.

1.3.1 Three-Dimensional FIR Filters

If a causal 3-D FIR filter of order NxNxN is given an input $x(n_1,n_2,n_3)$, the output $y(n_1,n_2,n_3)$ can be expressed as

$$y(n_1,n_2,n_3) = \sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} h(n_1,n_2,n_3)x(n_1-i,n_2-j,n_3-k) \tag{1.1}$$

Examination of Equation 1.1 shows that the filter's output is a weighted function of past input values. The term $h(n_1,n_2,n_3)$ is known as the impulse response of the filter. The transfer function of the above filter is obtained by taking the z-transform of Equation 1.1 and is given as

$$H(z_1,z_2,z_3) = \sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} h(n_1,n_2,n_3)z_1^{-i}z_2^{-j}z_3^{-k} \tag{1.2}$$

Equation 1.2 can also be written as

$$H(z_1,z_2,z_3) = \frac{\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} h(n_1,n_2,n_3)z_1^{N-i}z_2^{N-j}z_3^{N-k}}{z_1^{N}z_2^{N}z_3^{N}} \tag{1.3}$$

Equation 1.3 shows that all poles of this filter are located at the origin. As a result of this constraint on pole placement the stability of the filter is guaranteed. Therefore, no design

effort is required to ensure the stability of an FIR filter.

Another advantage of FIR filters is the ease with which they can be designed to have linear phase response, and therefore constant group delays, over the entire baseband[2].

The main disadvantage of FIR filters is directly related to their inherent stability. As mentioned, non-recursive filters are always stable because the poles are constrained to the origin. However, this constraint also reduces the possible steepness of the transition band. As a result, higher order filters are required to obtain specified transition specifications. These higher order translate into a higher implementation cost for the filter.

1.3.2 Three-Dimensional IIR Filters

If a causal 3-D IIR filter of order NxNxN is given an input $x(n_1,n_2,n_3)$, the output $y(n_1,n_2,n_3)$ can be expressed as

$$y(n_1,n_2,n_3) = \sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} a(i,j,k)x(n_1-i,n_2-j,n_3-k) - \sum_{\substack{i=0\\(i+j+k)\neq 0}}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} b(i,j,k)y(n_1-i,n_2-j,n_3-k) \tag{1.4}$$

As Equation 1.4 shows, the present value of the output is a function of the present and past values of the input, as well as past values of the output. Note that FIR filters are actually a subset of IIR filters where all $b(i,j,k)$ coefficients are equal to zero. By taking the z-transform of Equation 1.4 and setting $b(0,0,0)$ equal to one, the transfer function of a 3-D IIR digital filter is obtained as

$$H(z_1,z_2,z_3) = \frac{\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} a(i,j,k) z_1^{-i} z_2^{-j} z_3^{-k}}{1+\sum_{\substack{i=0 \\ (i+j+k)\neq 0}}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} b(i,j,k) z_1^{-i} z_2^{-j} z_3^{-k}} \tag{1.5}$$

As Equation 1.5 shows, 3-D IIR filters do not have their poles constrained to the origin. This gives IIR filters a degree of design flexibility not available in FIR filters. A transition band specification that requires a high order FIR filter can be obtained using a much lower order IIR filter. The required order of an FIR design can be as much as five to ten times higher than that of an IIR filter satisfying the same specifications[3]. These lower orders can translate into lower implementation costs, and the cost difference is even more pronounced in the design of 3-D filters. This extra cost difference is a result of the fact that the number of coefficients in a 3-D filter is exponentially (by a power of three) higher than the number of coefficients in a 1-D filter design.

Despite its advantages, the IIR filter has a significant disadvantage compared to the FIR filter. This disadvantage is the IIR filter's lack of inherent stability. Since the filter output is dependent on past output values, it can grow to infinity even though the filter is given finite input values. This presents a challenge in designing these filters to be stable.

Another disadvantage of recursive filters is their inherent non-linear phase response. Designing a filter with a constant delay and prescribed loss specifications is usually very difficult to do[3]. In general, if an application requires constant delay characteristics, these

characteristics are achieved by cascading a filter that satisfies the magnitude response with a delay equalizer. In some applications, linear phase may not be of great importance. In image processing however, two-dimensional images are very sensitive to phase distortion[3]. By extension, since each frame of a video sequence can be look upon as a two-dimensional image, linear phase is very important in video processing.

1.3.3 3-D FIR Filter Design Methods

1.3.3.1 Design Using Integration

The design of 3-D FIR filters using integration is very simple and straight-forward. Given the filter's frequency response $H(\omega_1,\omega_2,\omega_3)$, the impulse response $h(n_1,n_2,n_3)$ can be obtained as

$$h(n_1,n_2,n_3) = \frac{1}{8\pi^3}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} H(\omega_1,\omega_2,\omega_3)e^{j(\omega_1 n_1+\omega_2 n_2+\omega_3 n_3)}d\omega_1 d\omega_2 d\omega_3 \qquad (1.6)$$

In general, calculation of this triple integral may be very difficult analytically. Therefore, Equation 1.6 is often calculated using numerical integration. This eliminates the need for an analytical solution, and it lends itself well to computer-aided analysis[1].

1.3.3.2 Design Using FFT and Window Functions

This design method is very similar to the one given in Section 1.3.3.1. Given a desired frequency response of a filter, the impulse response $h(n_1,n_2,n_3)$ can be obtained by use of the Three-Dimensional Inverse Discrete Fourier Transform (IDFT). The IDFT is discussed in Chapter 3. While simple and straight-forward, this design method is sub-

optimal due to the occurrence of Gibb's Oscillations in which ripples appear in the passband and stop-band of the filter's magnitude response. These ripples can be reduced by applying a window function to the impulse response. The most common window functions are Hann and Hamming windows, Blackman windows, and Kaiser windows. These are 1-D windows that can easily be extended to two and three dimensions for application to two- and three-dimensional impulse responses [1][4][5][6].

1.3.3.3 McClellan Transformation

Another technique for designing 3-D FIR filters is obtained by extending the McClellan Transformation to three dimensions. This technique involves determining transformation coefficients, and then designing a 1-D FIR filter to be transformed using these coefficients. A large number of coefficients may result from this method, but this number can be reduced by imposing symmetry constraints[6].

1.3.3.4 Linear Programming

Linear Programming is a popular method for designing multidimensional filters. It is an iterative process that measures the difference between the desired and designed frequency responses, often as a sum-of-square-error, and minimizes this difference. Linear programming is a computationally expensive design method, but is becoming more practical as processing power becomes more easily available. More is said about linear programming in the next section.

1.3.4 3-D IIR Filter Design Methods

1.3.4.1 Linear Programming [2][6][9]

The design of 3-D IIR filters using linear programming involves calculation of the filter's numerator coefficients $a(i,j,k)$ and denominator coefficients $b(i,j,k)$ such that the magnitude response and/or phase response of the designed filter approximates a desired response while maintaining stability in the filter. The transfer function given in Equation 1.5 can involve two subclasses: the separable product transfer function and the separable denominator, non-separable numerator transfer function. These transfer functions are given in Equation 1.7 and 1.8 respectively.

$$\begin{aligned} H(z_1,z_2,z_3) &= H_1(z_1)H_2(z_2)H_3(z_3) \\ &= \left(\frac{\sum_{i=0}^{N} a_1(i)z_1^{-i}}{\sum_{i=0}^{N} b_1(i)z_1^{-i}}\right)\left(\frac{\sum_{j=0}^{N} a_2(j)z_2^{-j}}{\sum_{j=0}^{N} b_2(j)z_2^{-j}}\right)\left(\frac{\sum_{k=0}^{N} a_3(k)z_3^{-k}}{\sum_{k=0}^{N} b_3(k)z_3^{-k}}\right) \end{aligned} \tag{1.7}$$

The separable product transfer function allows the filter to be designed as a cascade arrangement of three 1-D filters. In this way, stability is guaranteed by designing the 1-D filters to be stable. The major drawback of this design method lies in the fact that a spherical-symmetric specification cannot be obtained. A filter with a separable product transfer function will always have a cubic shaped magnitude response.

$$H(z_1,z_2,z_3) = \frac{\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} a(i,j,k)z_1^{-i}z_2^{-j}z_3^{-k}}{\left(\sum_{i=0}^{N} b_1(i)z^{-i}\right)\left(\sum_{j=0}^{N} b_2(j)z^{-j}\right)\left(\sum_{k=0}^{N} b_3(k)z^{-k}\right)} \tag{1.8}$$

The separable denominator, non-separable numerator transfer function has a denominator

like that of the separable product transfer function. As a result of this, the stability problem is reduced to that of the 1-D case. A filter having this transfer function can be designed by cascading a 3-D FIR filter with three 1-D all-pole IIR filters. The separable denominator, non-separable numerator transfer function is more flexible than the separable product transfer function and can be used to design filters with spherical-symmetric specifications[7][8].

The general transfer function of Equation 1.5 gives the most flexible results since the constraints of Equation 1.7 and 1.8 are removed. Unfortunately this design method does not share the ease in designing for stability that the other methods do.

1.3.4.2 Bilinear Transformation

Another method of designing 3-D digital IIR filters involves assigning a stable 3-variable polynomial in the denominator of an analog transfer function and applying the triple bilinear transformation. Unfortunately, not all analog filters will yield a stable digital filter upon application of the bilinear transformation[10]. There is a specific class of analog filters that will yield stable digital filters, and these analog filters have Very Strictly Hurwitz Polynomials (VSHP) as their denominators[11]. The use of VSHP denominators is used in the design of both 2-D filters and 3-D filters[6]

1.3.4.3 Modified Shank's Method

The final design method to be discussed here is another extension of a 2-D method. This

2-D method is known as Shank's Method[12] and is modified in [1] to provide a near-linear phase response. It uses a weighted error function that measures the difference between the desired magnitude response and the designed magnitude response. This error function is then minimized by taking the derivative with respect to each of the {a} and {b} coefficients and equating to zero. The resulting linear equations are then solved to obtain an ideal impulse response. This ideal impulse response is utilized in such a manner as to obtain a near-linear phase response for the filter. As a result of the advantages of IIR filters over FIR filters, and the obtainable near-linear phase characteristic, the Modified Shank's Method is used in this thesis. It is discussed in detail in Chapter 3.

1.4 Video Formats

The moving image sequences used in this thesis are found in digital video files. This section briefly discusses some of the digital video formats in common use today.

1.4.1 H.263 Video Standard

The H.263 standard is a video coding standard published by the International Telecom Union (ITU). It is specifically designed to accommodate low bit-rate applications where bandwidth is limited. In particular, this video format has become standard in the field of video telephony. The coding algorithm is a hybrid of inter-picture prediction, transform coding, and motion compensation. In essence, this standard is primarily a compression algorithm designed to allow higher frame rate video to be sent over low-bandwidth channels. The ITU H.263 Recommendation is available from the International Telecom Union.

1.4.2 MPEG Video Standard

The MPEG-1 video standard is officially known as ISO/IEC Standard, Coded Representation of Picture, Audio and Multimedia/hypermedia Information, ISO 11172. MPEG-2 is a related standard and since this discussion relates equally to both, they will be commonly referred to as the MPEG video standard. The MPEG video standard is the adopted standard for the emerging application of High Definition TeleVision (HDTV). It has three types of frames: I-Frames, P-Frames, and B-Frames. I-Frames, or Intra-picture frames, are coded only using information present in the picture itself. P-Frames, or Predicted frames, are coded using the nearest previous I-Frame or P-Frame. B-Frames, or Bidirectional frames, are frames that use both a past and future frame as a reference. [http://www.c-cube.com/technology/mpeg.html#MPEG Overview] Like the H.263 standard, MPEG is primarily a compression algorithm. The MPEG standards are available from the International Standards Organization.

1.4.3 Microsoft Windows AVI Standard

The Microsoft Windows Audio Video Interleaved (AVI) format is a common video file format used to hold video sequences on Personal Computers (PCs) running the Microsoft Windows operating system. Unlike the H.263 and MPEG formats, the AVI is unsuitable for transmitting video data, and is not used in applications such as video-telephony or HDTV. It is a very simple video format which can often be found in an uncompressed form. The Microsoft Windows Application Program Interface (API) contains numerous functions for the manipulation of AVI files, and is well documented, allowing easy access

and manipulation of the raw video data.

1.4.4 Comparison

Since this thesis is concerned with applications of 3-D filters, one may at first assume that either the H.263 or MPEG standards would be an appropriate choice to use, since both are commonly used in real-world applications. However, as mentioned earlier, these formats are essentially compression standards. The application of using 3-D filters to increase video resolution is concerned not with compressed data, but rather with raw data. Any filtering algorithms developed to work on the raw data should also work with compressed formats. One need only decompress the data before filtering. For this reason, plus the wide availability of AVI files and AVI tools, the AVI format is preferable for the purposes of this thesis, since the raw data is more easily accessible than in the other formats. An added advantage to using this format arises from the fact that any PC running Microsoft Windows is capable of playing an AVI file.

1.5 Current Applications of 3-D Filters

Digital filters are widely used in the processing of 1-D and multidimensional signals. 1-D digital filters are commonly used in the area of speech or music processing. Other examples can be found in[13][14]. Due to the increased complexity and hardware cost of 2-D filters, they are not used as often as 1-D filters. Some applications of 2-D filters include image processing and seismic signal processing[15]. Three-dimensional filters are even more complex and expensive than 2-D filters, and are therefore even less used. Their

use is becoming more practical as VLSI technology continues to improve. These filters are currently used in the field of geophysics[6].

1.6 Thesis Organization

This thesis is divided into six chapters. Chapter 2 discusses the process of digital interpolation. It begins with an introduction to the sampling theorem, and then gives an explanation of interpolation in one dimension. Two methods of interpolation are discussed: interpolation using zero-padding, and interpolation using sample replication. These methods are then extended to three dimensions.

In Chapter 3 of this thesis, recursive filter design using the Modified Shank's method is discussed in detail. The three-dimensional Fast Fourier Transform is also developed.

The theory discussed in chapters two and three is tied together in Chapter 4 to outline the process used to create a three-dimensional digital interpolator (in software). The process of using this interpolator to increase the horizontal resolution, vertical resolution, and frame rate of an AVI video file is also given. Relevant details of the AVI format and the related BMP file format are provided.

The results of the thesis are given in Chapter 5. Plots are provided showing the characteristics of the designed 3-D filter, and frames of the filtered AVI file are shown side-by-side with frames from the original file to compare the resolution and quality.

A summary and conclusions are provided in the final chapter. A computer program written in Microsoft Visual C++ was used to test the theory of this thesis and produce the results found in Chapter 5. The source code for this thesis is found in Appendix A.

Chapter 2: Digital Interpolation

2.1 The Sampling Theorem

The sampling theorem states that any bandlimited continuous signal x(t) with frequency spectrum $X(j\omega) = 0$ for $|\omega| \geq \omega_s/2$, where $\omega_s = 2\pi/T$ and T is the sampling period, can be uniquely determined from its discrete values x(nT), where n is an integer[3]. This means that any signal sampled at greater than twice its highest frequency component can be reconstructed to any desired degree of accuracy.

A graphical description of the sampling theorem is given in Figure 2.1 and Figure 2.2.

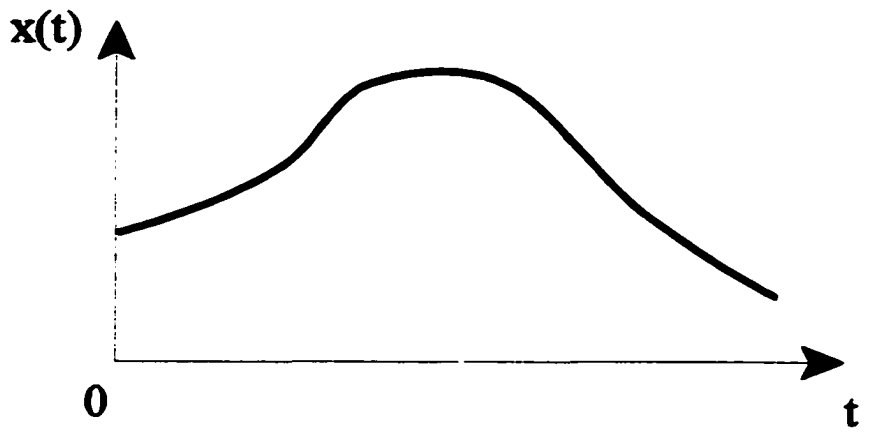


Figure 2.1(a): Continuous time signal x(t)

Figure 2.1(a) shows a one-dimensional continuous time signal denoted as x(t). If this signal is now sampled by multiplying it with the unit pulse train p(nT) shown in Figure 2.1(b), then the discrete time signal x(nT) shown in Figure 2.1(c) is obtained.

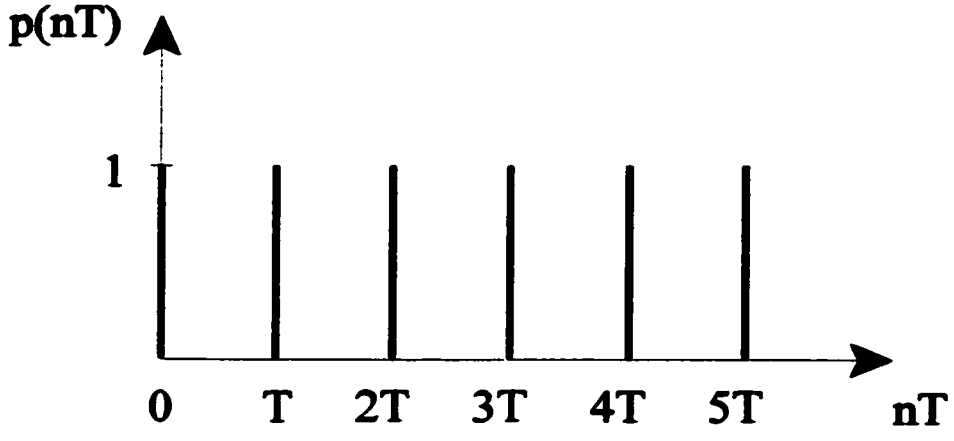


Figure 2.1(b): Unit pulse train p(nT)

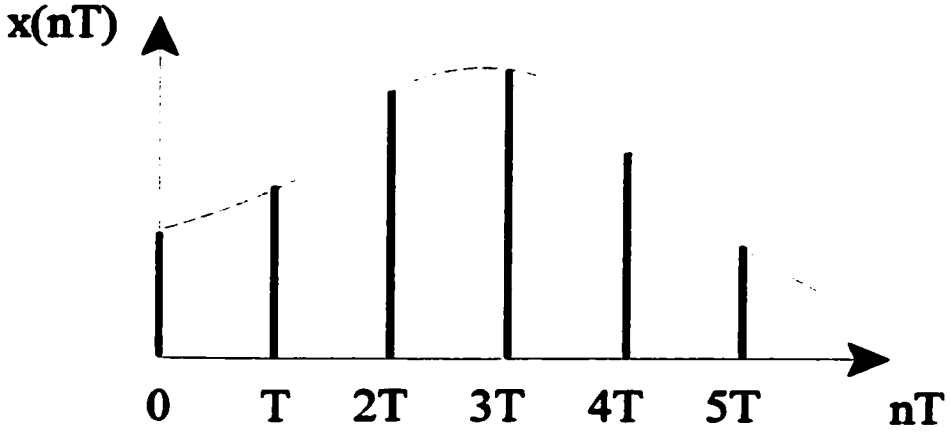


Figure 2.1(c): Discrete time signal x(nT)

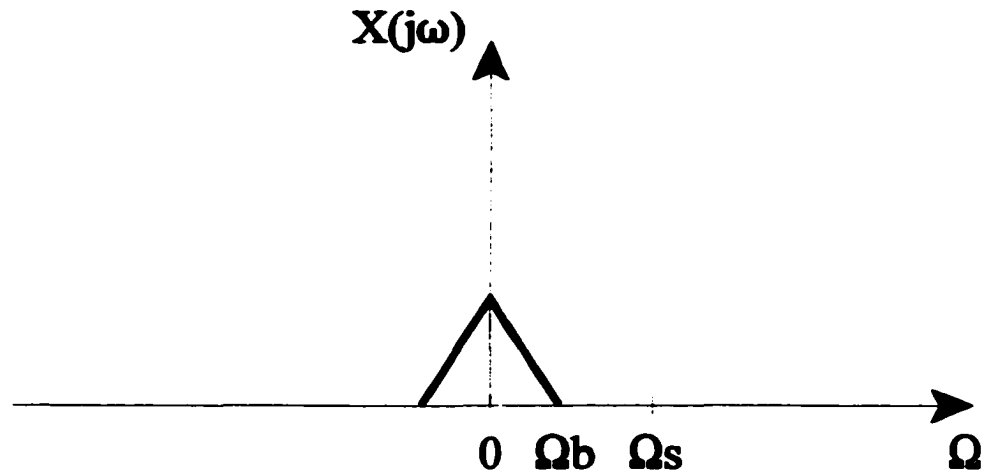


Figure 2.2(a): Frequency spectrum of continuous time signal x(t)

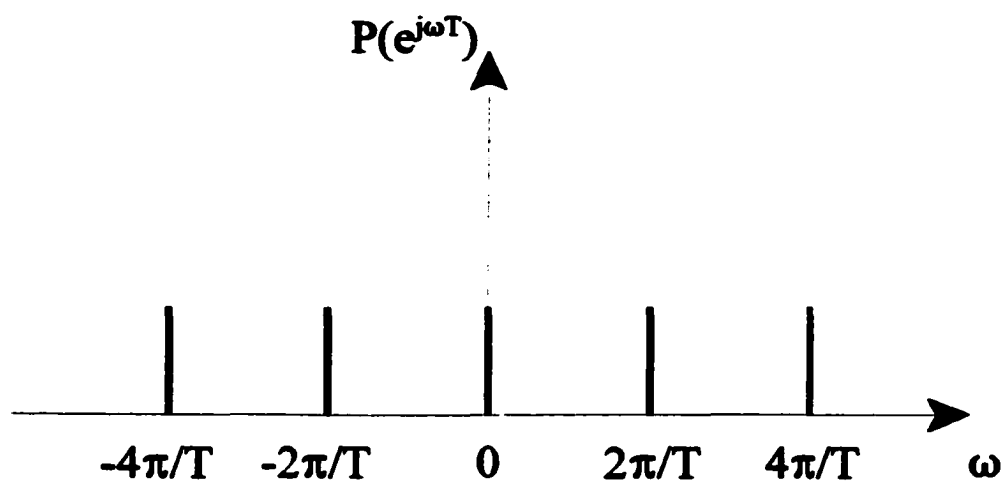


Figure 2.2(b): Frequency spectrum of unit pulse train p(t)

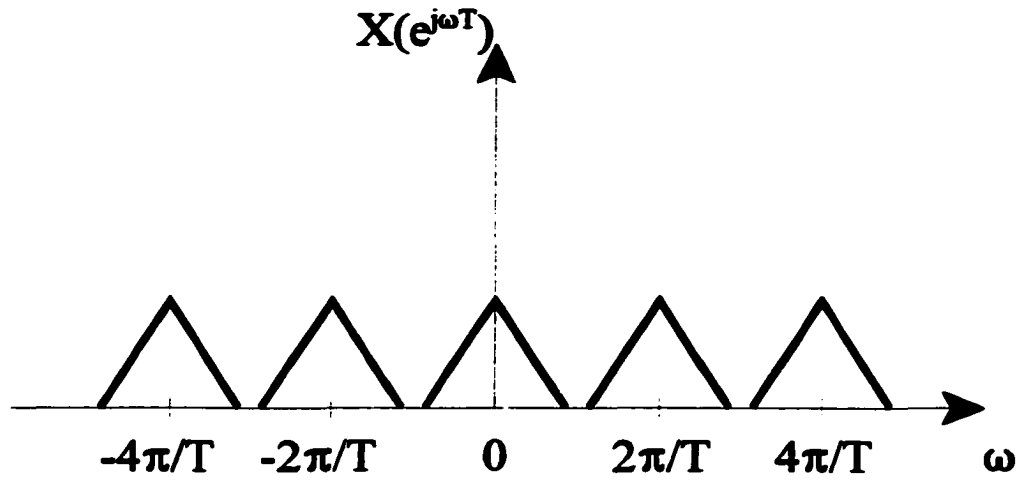
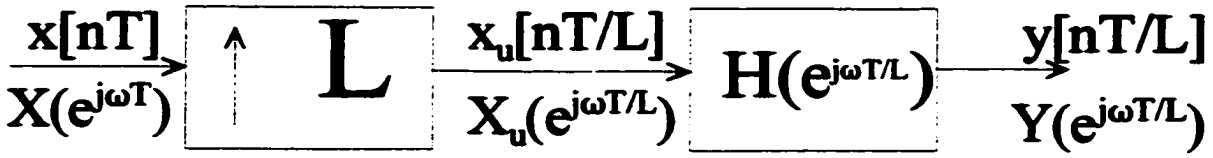


Figure 2.2(c): Frequency spectrum of discrete time signal x(nT)

Figure 2.2 illustrates this process in the frequency domain. Figure 2.2(a) shows the frequency spectrum $X(j\omega)$ of the continuous time signal x(t). Figure 2.2(b) shows the frequency spectrum $P(e^{j\omega T})$ of the unit pulse train p(nT). The frequency spectrum $X(e^{j\omega T})$ of the discrete time signal x(nT) is shown in Figure 2.2(c). The spectrum $X(e^{j\omega T})$ is obtained by convolving $X(j\omega)$ with $P(e^{j\omega T})$, since multiplication in the time domain is equivalent to convolution in the frequency domain. By examination of Figure 2.2(c), it can be seen that as long as ω_s is greater than twice ω_b there will be no overlap between the frequency spectrum "images". Therefore the original spectrum of the continuous time signal has not been distorted by sampling, and all the information about the signal is retained.

2.2 One-Dimensional Interpolation

Digital interpolation in one dimension can be achieved by combining an upsampler with a



Upsampler - inserts L-1 samples between each pair of samples x[nT] and x[nT+1]

Low Pass Filter - transition band centered about π/L

Figure 2.3: Block diagram of digital interpolation system

lowpass filter as shown in Figure 2.3. [3][16] First consider the operation of the upsampler. If an upsampler using zero-padding is given an input x(nT), then its output $x_u(nT')$ can be expressed as

$$x_u(nT') = \begin{cases} x(^{nT}/_L) & \text{for } n=0, \pm L, \pm 2L, \ldots \\ 0 & \textit{otherwise} \end{cases} \qquad \text{where } T' = {}^{T}/_{L} \tag{2.1}$$

which can also be written as

$$x_u(nT) = \sum_{k=-\infty}^{\infty} x(kT)\delta(nT - kLT) \tag{2.2}$$

By applying the z-transform to Equation 2.2 and substituting $z = e^{j\omega T'}$, Equation 2.3 is obtained.

$$
\begin{aligned}
X_u(e^{j\omega T'}) &= \sum_{n=-\infty}^{\infty}\left[\sum_{k=-\infty}^{\infty} x(kT)\delta(nT-kLT)\right]e^{-j\omega nT'} \\
&= \sum_{k=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} x(kT)\left[\delta(nT-kLT)e^{-j\frac{\omega nT}{L}}\right] \\
&= \sum_{k=-\infty}^{\infty} x(kT)e^{-j\omega kT} \\
&= X(e^{j\omega T})
\end{aligned}
\tag{2.3}
$$

Equation 2.3 shows that the frequency spectrum of $x_u(nT')$ is identical to the frequency spectrum of $x(nT)$. Since $T' = T/L$, $\omega_s' = L\omega_s$ and therefore the location of the sampling frequency has been changed. Now in the range $-\omega_s'/2 \le \omega \le \omega_s'/2$ there are L

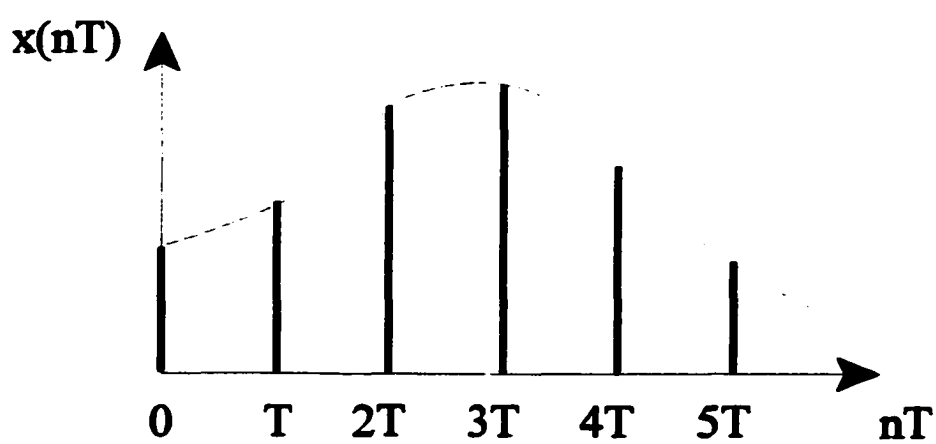


Figure 2.4(a): Discrete time signal x(nT)

images of $X(e^{j\omega T})$. An interpolated signal $x_i(nT')$ with an increased sampling rate can be obtained from $x_u(nT')$ by using a lowpass filter to remove the extra images. The

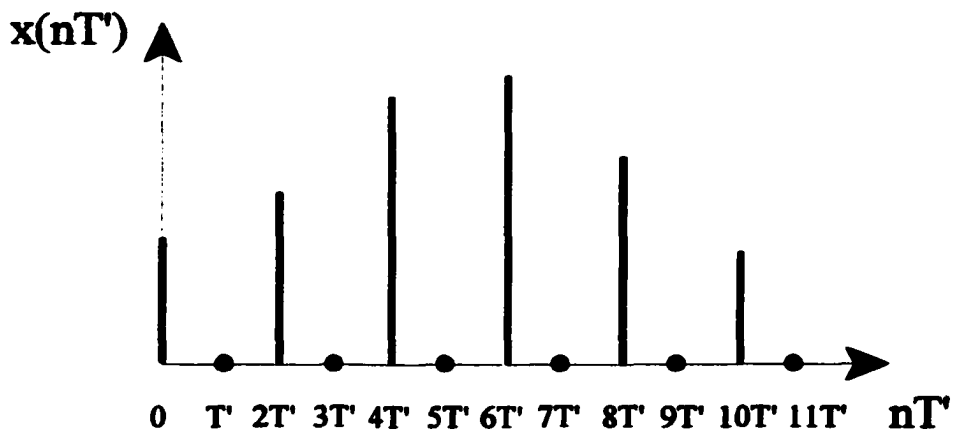


Figure 2.4(b): Discrete time signal x(nT) with zero padding

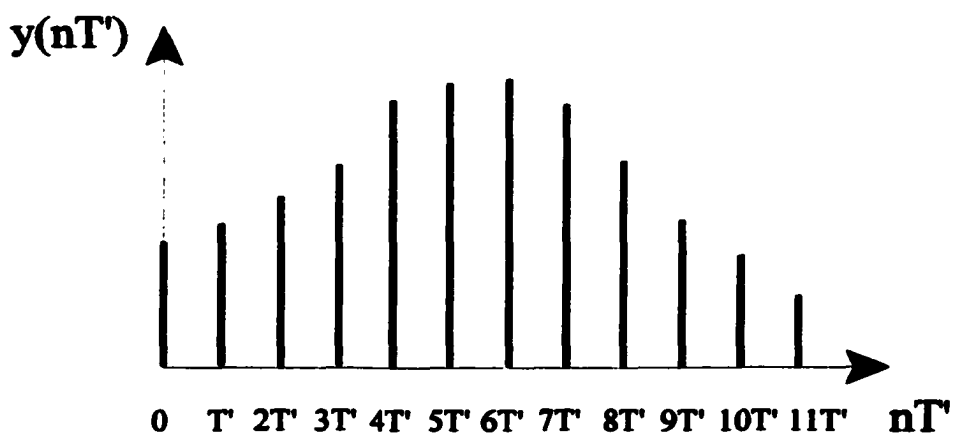


Figure 2.4(c): Interpolator output y(nT)

combination of the upsampler with the lowpass filter is called an interpolator. The interpolation process is shown graphically in Figure 2.4.

The method presented above will theoretically yield a perfect reconstruction of the original signal, assuming an ideal low-pass filter is used. There is a variation of the above method which does not yield a perfect reconstruction, but is better suited to a hardware implementation. In this variation, the upsampler uses sample replication instead of zero padding. Now if the upsampler is given an input x(nT), and L is assumed to be equal to two, then the output $x_u(nT')$ can be expressed as

$$x_u(nT') = \sum_{k=-\infty}^{\infty} x(kT)\delta(nT - 2kT) + \sum_{k=-\infty}^{\infty} x(kT)\delta(nT - (2k+1)T) \qquad (2.4)$$

Examination of Equation 2.4 shows that the upsampled signal consists of the original signal added to a time-shifted image of itself. Application of the z-transform to the original and time-shifted signals gives

$$X_u(e^{j\omega T'}) = X(e^{j\omega T}) + e^{-j\omega\frac{T}{2}} X(e^{j\omega T}) \qquad (2.5)$$

By evaluating Equation 2.5 at various values of ω, it can be noted that this modified upsampler using sample replication has a slight low-pass filtering effect on the signal. The justification for this slight distortion lies in the hardware implementation of the upsampler. To use the zero- padding method, the original signal must be fed into the upsampler and a zero sample must be explicitly inserted between each sample. With the sample replication, the upsampling process is much easier as it only requires the filter to run at twice the speed, or sample rate, of the incoming signal.

2.3 Three-Dimensional Interpolation

The procedure described in Section 2.2 will interpolate a one-dimensional signal such as an audio signal. To interpolate a three-dimensional video file, the procedure must be extended to three dimensions. Following the same method as before, the output of the upsampler $x_u(n_1T_1', n_2T_2', n_3T_3')$ can be expressed as

$$x_u(n_1T_1', n_2T_2', n_3T_3') = \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}\sum_{k_3=-\infty}^{\infty} x(k_1T_1, k_2T_2, k_3T_3)\delta(n_1T_1 - k_1L_1T_1, n_2T_2 - k_2L_2T_2, n_3T_3 - k_3L_3T_3) \quad \textbf{(2.6)}$$

For simplicity, let $L=L_1=L_2=L_3$ and $T=T_1=T_2=T_3$. Application of the z-transform and substitution of $z_1 = e^{j\omega_1 T'}, z_2 = e^{j\omega_2 T'}, z_3 = e^{j\omega_3 T'}$ gives

$$\begin{aligned}
X_u(e^{j\omega_1 T}, e^{j\omega_2 T}, e^{j\omega_3 T}) &= \sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty}\sum_{n_3=-\infty}^{\infty}\left[\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}\sum_{k_3=-\infty}^{\infty} x(k_1T, k_2T, k_3T)\delta(n_1T - k_1LT, n_2T - k_2LT, n_3T - k_3LT)\right]e^{-j\omega_1 n_1 T}e^{-j\omega_2 n_2 T}e^{-j\omega_3 n_3 T} \\
&= \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}\sum_{k_3=-\infty}^{\infty}\sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty}\sum_{n_3=-\infty}^{\infty} x(k_1T, k_2T, k_3T)\left[\delta(n_1T - k_1LT, n_2T - k_2L_2T, n_3T - k_3LT)e^{-j\frac{\omega_1 n_1 T}{L}}e^{-j\frac{\omega_2 n_2 T}{L}}e^{-j\frac{\omega_3 n_3 T}{L}}\right] \\
&= \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}\sum_{k_3=-\infty}^{\infty} x(k_1T, k_2T, k_3T)e^{-j\omega_1 k_1 T}e^{-j\omega_2 k_2 T}e^{-j\omega_3 k_3 T} \\
&= X(e^{j\omega_1 T}, e^{j\omega_2 T}, e^{j\omega_3 T})
\end{aligned}$$

(2.7)

As in the one-dimensional case, the frequency spectrum of the upsampled signal is identical to that of the original signal, while the sampling rates have been increased in each of the three dimensions. Now in the range

$(-\omega_{s1}/2 \le \omega_1 \le \omega_{s1}/2), (-\omega_{s2}/2 \le \omega_2 \le \omega_{s2}/2), (-\omega_{s3}/2 \le \omega_3 \le \omega_{s3}/2)$ there are L^3 images of the

original spectrum. The three-dimensional interpolated signal $x_i(n_1T', n_2T', n_3T')$ with an increased sampling rate in each dimension can be recovered by applying a lowpass filter with a cubic response to remove the unnecessary images.

As in the one dimensional case, there is the option of using sample (pixel) replication instead of zero-padding. In this case, the output of the upsampler will be

$$\begin{aligned} x_u(n_1T_1', n_2T_2', n_3T_3') = & \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}\sum_{k_3=-\infty}^{\infty} x(k_1T_1, k_2T_2, k_3T_3)\delta(n_1T_1 - 2k_1T_1, n_2T_2 - 2k_2T_2, n_3T_3 - 2k_3T_3) \\ & + \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}\sum_{k_3=-\infty}^{\infty} x(k_1T_1, k_2T_2, k_3T_3)\delta(n_1T_1 - (2k_1+1)T_1, n_2T_2 - (2k_2+1)T_2, n_3T_3 - (2k_3+1)T_3) \end{aligned} \tag{2.8}$$

The output of the three-dimensional upsampler is the input added to a shifted version of itself. Transforming Equation 2.8 to the frequency domain gives

$$\begin{aligned} X_u(e^{j\omega_1T_1'}, e^{j\omega_2T_2'}, e^{j\omega_3T_3'}) = & X(e^{j\omega_1T_1}, e^{j\omega_2T_2}, e^{j\omega_3T_3}) \\ & + e^{-j\left(\frac{\omega_1T_1}{2}+\frac{\omega_2T_2}{2}+\frac{\omega_3T_3}{2}\right)} X(e^{j\omega_1T_1}, e^{j\omega_2T_2}, e^{j\omega_3T_3}) \end{aligned} \tag{2.9}$$

The result of Equation 2.9 shows that there is again a slight low-pass filtering effect on the signal when pixel replication is used. The justification for this method again lies in the hardware implementation. By using pixel replication, there is no zero-insertion required between each pixel, line, and frame. Instead, the result can be obtained by running the filter at eight times the original speed (a factor of two for each dimension).

Chapter 3: Filter Design

3.1 Introduction

As mentioned in the introduction, one of the goals of this thesis involves the design of a stable recursive filter with a near-linear phase response. A method for designing such filters already exists in two dimensions. This method will now be extended to three dimensions.

3.2 Modified Shank's Method for 2-D Filter Design[1]

The following three-dimensional method for designing stable recursive filters with near-linear phase response is based on the filter design technique known as Shank's Method[12]. Although this method is a spatial design method, or a space-time design method once extended to three dimensions, the derivation is given in the frequency domain for purposes of clarity.

As given in Chapter 1, the transfer function of a three-dimensional recursive filter is described by

$$H(z_1,z_2,z_3) = \frac{\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} a_{ijk} z_1^{-i} z_2^{-j} z_3^{-k}}{1+\sum_{\substack{i=0\\(i+j+k)\neq 0}}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} b_{ijk} z_1^{-i} z_2^{-j} z_3^{-k}} \tag{3.1}$$

Note that b_{000} is arbitrarily set to equal 1. By substituting $z_1 = e^{j\omega_1 T}$, $z_2 = e^{j\omega_2 T}$,

and $z_3 = e^{j\omega_3 T}$, the frequency response of the filter is obtained as

$$H(\omega_1,\omega_2,\omega_3) = \frac{A(\omega_1,\omega_2,\omega_3)}{B(\omega_1,\omega_2,\omega_3)} \tag{3.2}$$

where

$$A(\omega_1,\omega_2,\omega_3) = \sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} a_{ijk} e^{j\omega_1 iT} e^{j\omega_2 jT} e^{j\omega_3 kT} \tag{3.3}$$

and

$$B(\omega_1,\omega_2,\omega_3) = 1 + \sum_{\substack{i=0 \\ (i+j+k)\neq 0}}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} b_{ijk} e^{j\omega_1 iT} e^{j\omega_2 jT} e^{j\omega_3 kT} \tag{3.4}$$

By now letting $H^d(\omega_1,\omega_2,\omega_3)$ represent the desired frequency response of the filter, an error function can be defined as

$$\varepsilon(\omega_1,\omega_2,\omega_3) = H^d(\omega_1,\omega_2,\omega_3) - \frac{A(\omega_1,\omega_2,\omega_3)}{B(\omega_1,\omega_2,\omega_3)} \tag{3.5}$$

When transformed back to the space-time domain, Equation 3.5 becomes

$$\varepsilon(n_1,n_2,n_3) = h^d(n_1,n_2,n_3) - h(n_1,n_2,n_3) \tag{3.6}$$

where $h^d(n_1,n_2,n_3)$ is the desired impulse response and $h(n_1,n_2,n_3)$ represents the

impulse response of the designed filter. $h(n_1,n_2,n_3)$ is described by

$$h(n_1,n_2,n_3)=\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N}a_{ijk}\delta(n_1-i,n_2-j,n_3-k)-\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N}b_{ijk}h(n_1-i,n_2-j,n_3-k) \quad (3.7)$$

Forming the L_2 norm using the error function in Equation 3.6 gives

$$Q=\sum_{n_1=0}^{M-1}\sum_{n_2=0}^{M-1}\sum_{n_3=0}^{M-1}\varepsilon^2(n_1,n_2,n_3) \quad (3.8)$$

where $M \times M \times M$ points are taken from the impulse response for this computation. The design of the filter now consists of determining the values of the $\{a\}$ and $\{b\}$ coefficients such that the expression in Equation 3.8 is minimized. This is done by taking the derivative of Q with respect to each coefficient and equating the resulting equations to zero. This results in Equation 3.9 and Equation 3.10.

$$\frac{\partial Q}{\partial a_{xyz}}=2\sum_{n_1=0}^{M-1}\sum_{n_2=0}^{M-1}\sum_{n_3=0}^{M-1}\varepsilon(n_1,n_2,n_3)\frac{\partial\varepsilon(n_1,n_2,n_3)}{\partial a_{xyz}} \quad (3.9)$$
$$x,y,z=0,1,2,\ldots,N$$

and

$$\frac{\partial Q}{\partial b_{xyz}}=2\sum_{n_1=0}^{M-1}\sum_{n_2=0}^{M-1}\sum_{n_3=0}^{M-1}\varepsilon(n_1,n_2,n_3)\frac{\partial\varepsilon(n_1,n_2,n_3)}{\partial b_{xyz}} \quad (3.10)$$
$$x,y,z=0,1,2,\ldots,N \qquad (x+y+z)\neq 0$$

Equation 3.9 generates $(N+1)\times(N+1)\times(N+1)$ nonlinear equations in $\{a\}$, while Equation 3.10 generates $(N+1)\times(N+1)\times(N+1)-1$ nonlinear equations in $\{b\}$. There are $2(N+1)\times(N+1)\times(N+1)-1$ filter coefficients, and these generated equations form a complete set from which the coefficients can be solved.

While solvable, the above system of equations is highly nonlinear. To avoid this non-linearity, reconsider the error equation of Equation 3.5 in the following form

$$\hat{\varepsilon}(\omega_1,\omega_2,\omega_3) = B(\omega_1,\omega_2,\omega_3)H^d(\omega_1,\omega_2,\omega_3) - A(\omega_1,\omega_2,\omega_3) \tag{3.11}$$

where $\varepsilon(\omega_1,\omega_2,\omega_3)B(\omega_1,\omega_2,\omega_3)$ has been replace by a "weighted" error term $\hat{\varepsilon}(\omega_1,\omega_2,\omega_3)$. Now transformation of Equation 3.11 to the space-time domain results in the error equation

$$\hat{\varepsilon}(n_1,n_2,n_3) = \sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} b_{ijk}h^d(n_1-i,n_2-j,n_3-k) - \sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} a_{ijk}\delta(n_1-i,n_2-j,n_3-k) \tag{3.12}$$

Forming the L_2 norm again, but now using the modified error term results in

$$Q = \sum_{n_1=0}^{M-1}\sum_{n_2=0}^{M-1}\sum_{n_3=0}^{M-1} \hat{\varepsilon}^2(n_1,n_2,n_3) \tag{3.13}$$

It should now be clear why this derivation began in the frequency domain, despite the fact that it is a space-time design method. By beginning the derivation in the frequency domain, it is clearly shown that the error being minimized in Equation 3.13 is a "weighted" error,

and not the true error.

This new error Q can be minimized by differentiating with respect to each of the filter coefficients, and setting the resulting equations equal to zero. Differentiating Q with respect to the $\{a\}$ coefficients gives

$$\frac{\partial Q}{\partial a_{xyz}} = 2\sum_{n_1=0}^{M-1}\sum_{n_2=0}^{M-1}\sum_{n3=0}^{M-1}\left[\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} a_{ijk}\delta(n_1-i,n_2-j,n_3-k) - \sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} b_{ijk}h^d(n_1-i,n_2-j,n_3-k)\right]\delta(n_1-x,n_2-y,n_3-z) = 0 \quad (3.14)$$
$$x,y,z = 0,1,2,\ldots,N$$

This reduces to

$$a_{n_1n_2n_3} = \sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} b_{ijk}h^d(n_1-i,n_2-j,n_3-k) \quad (3.15)$$
$$n1,n2,n3 = 0,1,2,\ldots,N$$

As a result of Equation 3.15, Equation 3.13 can be rewritten as

$$Q = \sum_{n1=N+1}^{M-1}\sum_{n2=N+1}^{M-1}\sum_{n3=N+1}^{M-1}\left[\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} b_{ijk}h^d(n_1-i,n_2-j,n_3-k)\right]^2 \quad (3.16)$$

since $a_{n_1n_2n_3} = 0$ for $N+1 \le n_1,n_2,n_3 \le M-1$. Now minimize Q by differentiating with respect to the $\{b\}$ coefficients and equating the resulting equations to zero. This gives

$$\frac{\partial Q}{\partial b_{xyz}} = 2\sum_{n_1=N+1}^{M-1}\sum_{n_2=N+1}^{M-1}\sum_{n_3=N+1}^{M-1}\left[\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} b_{ijk}h^d(n_1-i,n_2-j,n_3-k)\right]h^d(n_1-x,n_2-y,n_3-z)=0 \quad (3.17)$$
$$x,y,z = 0,1,2,\ldots,N$$

which reduces to

$$\sum_{\substack{i=0\\(i+j+k)\neq 0}}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} b_{ijk}\sum_{n_1=N+1}^{M-1}\sum_{n_2=N+1}^{M-1}\sum_{n_3=N+1}^{M-1}h^d(n_1-i,n_2-j,n_3-k)h^d(n_1-x,n_2-y,n_3-z) = -\sum_{n_1=N+1}^{M-1}\sum_{n_2=N+1}^{M-1}\sum_{n_3=N+1}^{M-1}h^d(n_1,n_2,n_3)h^d(n_1-x,n_2-y,n_3-z) \quad (3.18)$$

These equations generate $(N+1)\times(N+1)\times(N+1)$ linear equations in $\{a\}$ and $(N+1)\times(N+1)\times(N+1)-1$ linear equations in {b}. This set of linear equations can easily be solved for the filter coefficients.

3.3 Designing the 3-D Recursive Filter[1]

Once the desired impulse response is generated, a decision must be made as to how this response should be utilized. There are four possible options for the utilization of the impulse response:

1. Use the eight cubes of the impulse response (entire large cube shown in Figure 3.1) with the origin being at the center of the array.
2. Shift the axis such that the entire impulse response is in the cube where $n_1, n_2, n_3 \geq 0$.
3. Take the impulse response from only one cube of the array where $n_1, n_2, n_3 \geq 0$ (only utilize 1/8 of the impulse response). This option is depicted by the dotted

cube in Figure 3.1.

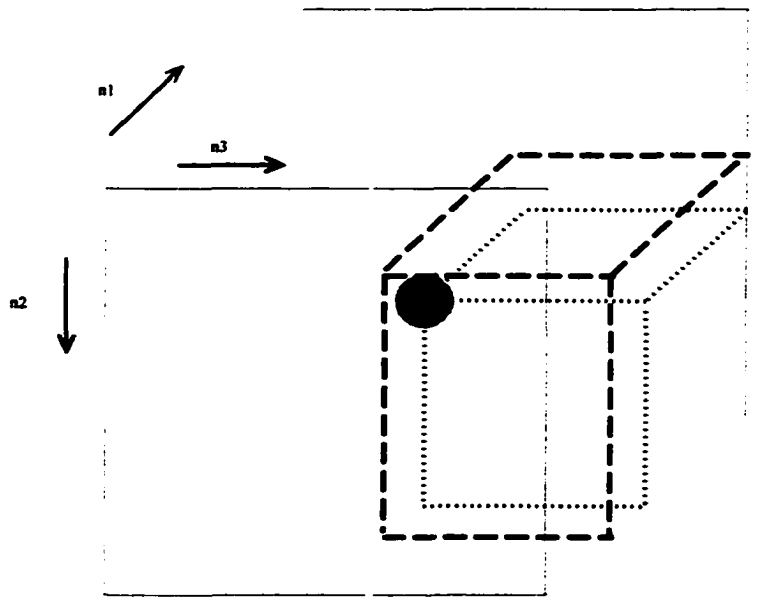


Figure 3.1: Utilization of Impulse Response

4. Shift the axes by an amount that is large enough to include the largest components of the impulse response in the cube $n_1, n_2, n_3 \geq 0$ and use this cube. This option is depicted by the dashed line in Figure 3.1.

The first option cannot be used since a filter's impulse response must be zero in the range $(n_1 < 0)$, $(n_2 < 0)$, and $(n_3 < 0)$ for the filter to be causal.

The second option will provide a causal filter, but the order of the filter will need to be at least $(M/2 \times M/2 \times M/2)$. The need for such a high order is a result of the large delay of the

impulse response in all directions. The third option will also provide a causal filter since values of $h(n_1, n_2, n_3)$ are only used when $(n_1 \geq 0)$, $(n_2 \geq 0)$, and $(n_3 \geq 0)$. Due to symmetry, the complete magnitude spectrum can be obtained by using only the one cube. However, while the magnitude spectrum will be preserved, the phase characteristic will not be preserved. Since linear phase is very important in video processing applications, this method is unsatisfactory.

In the fourth option, two characteristics of the impulse response are made use of :

i) The impulse response decays rapidly away from the origin.

ii) A shift in the impulse response in the space-time domain corresponds to the addition of a linear phase to the frequency domain.

Since the larger values of the impulse response (shown as the sphere in the middle of cube of Figure 3.1) are being used (which have the largest influence on the magnitude and phase response), and a linear shift is being added to a filter originally specified as zero-phase, this method can be used to design filters with near-linear phase characteristics. By using this method, most of the characteristics of the original desired frequency response are preserved without the large delay that would exist if the entire impulse response were used. It has been found[1] that the "shift" specified in Option 4 is best set to N-1 in each dimension.

3.4 Three-Dimensional Inverse Fast Fourier Transform

Since the method discussed above is spatio-temporal, it requires the ideal impulse response before the error function can be formed. Usually however, a filter's specification

is given in terms of its frequency response. Therefore a method of transforming the frequency response into the impulse response is required. The most straight-forward way of doing this is through the use of the Three-Dimensional Inverse Discrete Fourier Transform (IDFT), given by

$$X(k_1,k_2,k_3) = \frac{1}{N_1N_2N_3}\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}\sum_{n_3=0}^{N_3-1} x(n_1,n_2,n_3)e^{j\frac{2\pi}{N_1}(n_1k_1+n_2k_2+n_3k_3)} \quad (3.19)$$

If a filter with a cubic response is used, then the assignment $N_1=N_2=N_3=N$ can be made for simplicity and Equation 3.19 can be rewritten as

$$X(k_1,k_2,k_3) = \frac{1}{N^2}\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}\left[\frac{1}{N}\sum_{n_3=0}^{N-1} x(n_1,n_2,n_3)e^{j\frac{2\pi}{N}n_3k_3}\right]e^{j\frac{2\pi}{N}(n_2k_2+n_3k_3)} \quad (3.20)$$

Now let

$$G(n_1,n_2,k_3) = \frac{1}{N}\sum_{n_3=0}^{N-1} x(n_1,n_2,n_3)e^{j\frac{2\pi}{N}n_3k_3} \quad (3.21)$$

$$for\ n_1,n_2 = 0,1,2,\cdots,N-1$$

Equation 3.21 is essentially the one-dimensional IDFT of the n_2^{th} row of the n_1^{th} "frame". This is shown graphically in Figure 3.2(a). Using Equation 3.21, the following can be stated

$$H(n_1,k_2,k_3) = \frac{1}{N}\sum_{n_2=0}^{N-1} G(n_1,n_2,k_3)e^{j\frac{2\pi}{N}n_2k_2} \tag{3.22}$$

$$for\ n_1,k_3 = 0,1,2,\cdots,N-1$$

Equation 3.22 is the one-dimensional IDFT of the resulting k_3^{th} column of the n_1^{th} "frame". This is shown graphically in Figure 3.2(b). Now consider the following

$$X(k_1,k_2,k_3) = \frac{1}{N}\sum_{n_1=0}^{N-1} H(n_1,k_2,k_3)e^{j\frac{2\pi}{N}n_1k_1} \tag{3.23}$$

$$for\ k_2,k_3 = 0,1,2,\cdots,N-1$$

Equation 3.23 is essentially the one-dimensional IDFT of each resulting "depth vector" of the array. This is shown graphically in Figure 3.2(c). What has been accomplished above is the breakdown of the Three-Dimensional Inverse Discrete Fourier Transform into multiple One-Dimensional Inverse Discrete Fourier Transforms. This method can be used to take advantage of the computational efficiency of the One-Dimensional Fast Fourier Transform when the desired impulse response is formed for the modified Shank's method presented earlier.

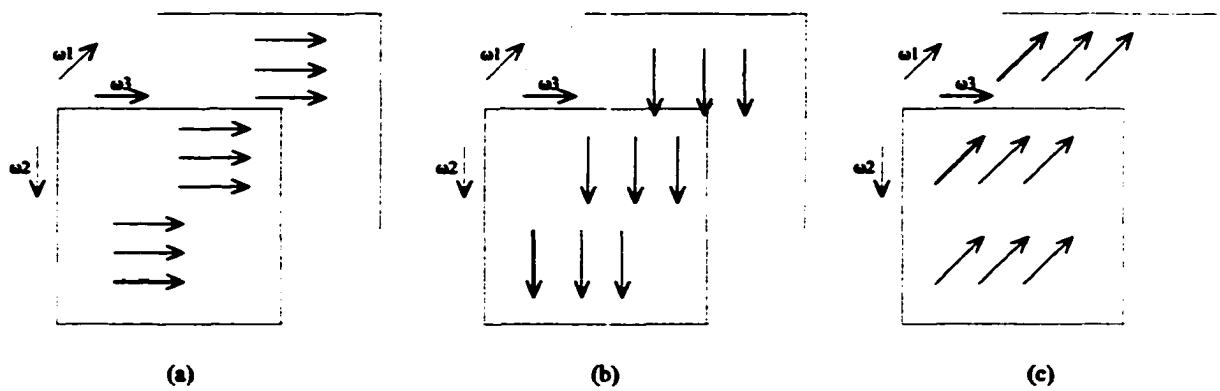


Figure 3.2: Three-Dimensional Fast Fourier Transform

Chapter 4: Implementation

4.1 Introduction

With the basic theory presented, the actual process of using a three-dimensional filter to improve video resolution is now tested. This is done by extracting the raw data from the video file, filtering it, and reconstructing the file for playback comparison. This process is illustrated in Figure 4.1. As mentioned in the introduction of this thesis, there are many functions available in the Microsoft SDK to modify AVI files. These functions are grouped under the AVIFile library. One particular function of interest extracts an individual frame of video as a Microsoft BMP bitmap image. Since four of the six steps (as outlined in Figure 4.1) involve AVI or BMP file manipulation, this chapter begins with an overview of the AVI and BMP file formats. The rest of the chapter discusses each step of the flowchart in Figure 4.1. Code snippets used in the Filter3D program dealing with the AVI codec are given at the end of the chapter, and described throughout the chapter.

4.2 AVI and BMP File Formats

4.2.1 AVI File Format

The AVI format is a sub-format of the Microsoft Resource Interchange File Format (RIFF). This format is based on the Electronic Arts Interchange File Format (IFF)[17] which is a general purpose data storage format for associating and storing multiple types of data. As the name implies, an Audio Visual Interleaved (AVI) file can contain both Audio and Video data.

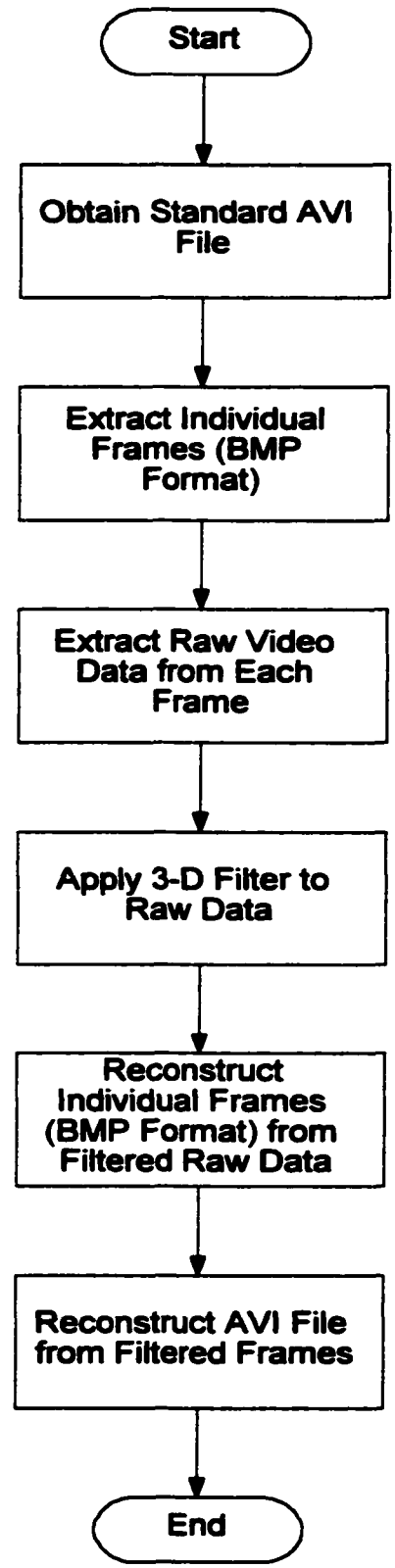


Figure 4.1: Implementation

While the IFF format uses tagged blocks of data called chunks, the AVI format handles its information as data streams. Data streams broadly refer to the components of a time-based file, either audio or video in the case of AVI files. This thesis is concerned only with the video stream of a file, and audio streams are ignored when reading the AVI files. Each AVI file consists of one file header, one or more stream headers, and the file data. The structures AVIFILEINFO and AVISTREAMINFO hold the file header and stream header respectively. The following structure definitions are taken directly from the Microsoft Developer Studio help files.

The **AVIFILEINFO** structure contains global information for an entire AVI file.

```
typedef struct {
    DWORD dwMaxBytesPerSec;
    DWORD dwFlags;
    DWORD dwCaps;
    DWORD dwStreams;
    DWORD dwSuggestedBufferSize;
    DWORD dwWidth;
    DWORD dwHeight;
    DWORD dwScale;
    DWORD dwRate;
    DWORD dwLength;
    DWORD dwEditCount;
    char  szFileType[64];
} AVIFILEINFO;
```

Members

dwMaxBytesPerSec

Approximate maximum data rate of the AVI file.

dwFlags

Applicable flags. The following flags are defined:

AVIFILEINFO_HASINDEX

The AVI file has an index at the end of the file. For good performance, all AVI files should contain an index.

AVIFILEINFO_MUSTUSEINDEX

The file index contains the playback order for the chunks in the file. Use the index rather than the physical ordering of the chunks when playing back the data. This could be used for creating a list of frames for editing.

AVIFILEINFO_ISINTERLEAVED

The AVI file is interleaved.

AVIFILEINFO_WASCAPTUREFILE

The AVI file is a specially allocated file used for capturing real-time video. Applications should warn the user before writing over a file with this flag set because the user probably defragmented this file.

AVIFILEINFO_COPYRIGHTED

The AVI file contains copyrighted data and software. When this flag is used, software should not permit the data to be duplicated.

dwCaps

Capability flags. The following flags are defined:

AVIFILECAPS_CANREAD

An application can open the AVI file with with the read privilege.

AVIFILECAPS_CANWRITE

An application can open the AVI file with the write privilege.

AVIFILECAPS_ALLKEYFRAMES

Every frame in the AVI file is a key frame.

AVIFILECAPS_NOCOMPRESSION

The AVI file does not use a compression method.

dwStreams

Number of streams in the file. For example, a file with audio and video has at least two streams.

dwSuggestedBufferSize

Suggested buffer size, in bytes, for reading the file. Generally, this size

should be large enough to contain the largest chunk in the file. For an interleaved file, this size should be large enough to read an entire record, not just a chunk.

If the buffer size is too small or is set to zero, the playback software will have to reallocate memory during playback, reducing performance.

dwWidth

Width, in pixels, of the AVI file.

dwHeight

Height, in pixels, of the AVI file.

dwScale

Time scale applicable for the entire file. Dividing **dwRate** by **dwScale** gives the number of samples per second.

Any stream can define its own time scale to supersede the file time scale.

dwLength

Length of the AVI file. The units are defined by **dwRate** and **dwScale**.

dwEditCount

Number of streams that have been added to or deleted from the AVI file.

szFileType

Null-terminated string containing descriptive information for the file type.

The **AVISTREAMINFO** structure contains information for a single stream.

```
typedef struct {
    DWORD fccType;
    DWORD fccHandler;
    DWORD dwFlags;
    DWORD dwCaps;
    WORD  wPriority;
    WORD  wLanguage;
    DWORD dwScale;
    DWORD dwRate;
    DWORD dwStart;
    DWORD dwLength;
    DWORD dwInitialFrames;
    DWORD dwSuggestedBufferSize;
```

```
    DWORD dwQuality;
    DWORD dwSampleSize;
    RECT  rcFrame;
    DWORD dwEditCount;
    DWORD dwFormatChangeCount;
    char  szName[64];
} AVISTREAMINFO;
```

Members

fccType

Four-character code indicating the stream type. The following constants have been defined for the data commonly found in AVI streams:

streamtypeAUDIO	Indicates an audio stream.
streamtypeMIDI	Indicates a MIDI stream.
streamtypeTEXT	Indicates a text stream.
streamtypeVIDEO	Indicates a video stream.

fccHandler

Four-character code of the compressor handler that will compress this video stream when it is saved (for example, **mmioFOURCC('M','S','V','C')). This member is not used for audio streams.**

dwFlags

Applicable flags for the stream. The bits in the high-order word of these flags are specific to the type of data contained in the stream. The following flags are defined:

AVISTREAMINFO_DISABLED

Indicates this stream should be rendered when explicitly enabled by the user.

AVISTREAMINFO_FORMATCHANGES

Indicates this video stream contains palette changes. This flag warns the playback software that it will need to animate the palette.

dwCaps

Capability flags; currently unused.

wPriority

Priority of the stream.

wLanguage

Language of the stream.

dwScale

Time scale applicable for the stream. Dividing **dwRate** by **dwScale** gives the playback rate in number of samples per second.

For video streams, this rate should be the frame rate. For audio streams, this rate should correspond to the audio block size (the **nBlockAlign** member of the **WAVEFORMAT or PCMWAVEFORMAT structure), which for PCM (Pulse Code Modulation) audio reduces to the sample rate.**

dwRate

See **dwScale**.

dwStart

Sample number of the first frame of the AVI file. The units are defined by **dwRate** and **dwScale**. Normally, this is zero, but it can specify a delay time for a stream that does not start concurrently with the file.

The 1.0 release of the AVI tools does not support a nonzero starting time.

dwLength

Length of this stream. The units are defined by **dwRate** and **dwScale**.

dwInitialFrames

Audio skew. This member specifies how much to skew the audio data ahead of the video frames in interleaved files. Typically, this is about 0.75 seconds.

dwSuggestedBufferSize

Recommended buffer size, in bytes, for the stream. Typically, this member contains a value corresponding to the largest chunk in the stream. Using the correct buffer size makes playback more efficient. Use zero if you do not know the correct buffer size.

dwQuality

Quality indicator of the video data in the stream. Quality is represented as a number between 0 and 10,000. For compressed data, this typically represents the value of the quality parameter passed to the compression software. If set to – 1, drivers use the default quality value.

dwSampleSize

Size, in bytes, of a single data sample. If the value of this member is zero, the samples can vary in size and each data sample (such as a video frame) must be in a separate chunk. A nonzero value indicates that multiple samples of data can be grouped into a single chunk within the file.

For video streams, this number is typically zero, although it can be nonzero if all video frames are the same size. For audio streams, this number should be the same as the **nBlockAlign** member of the **WAVEFORMAT or WAVEFORMATEX structure describing the audio.**

rcFrame

Dimensions of the video destination rectangle. The values represent the coordinates of upper left corner, the height, and the width of the rectangle.

dwEditCount

Number of times the stream has been edited. The stream handler maintains this count.

dwFormatChangeCount

Number of times the stream format has changed. The stream handler maintains this count.

szName

Null-terminated string containing a description of the stream.

4.2.2 BMP File Format

The Microsoft BMP file format is the native bitmap format of the Microsoft Windows operating environment and is used to store virtually any type of bitmap data[17]. BMP

files consist of a file header, bitmap header, optional colour palette, and the bitmap data. All BMP files contain a file header and bitmap header (older bitmap files may only contain a file header, but those older formats are not discussed here). The colour palette exists if the number of bits constituting each pixel is eight or less (<= 8bpp). Since greyscale images are 8bpp, the BMP files examined here all have a colour palette. The structures BITMAPFILEHEADER and BITMAPINFOHEADER hold the file header and bitmap header respectively. The following definitions are taken directly from the Microsoft Developer Studio help files.

The **BITMAPFILEHEADER** structure contains information about the type, size, and layout of a file that contains a device-independent bitmap (DIB).

```
typedef struct tagBITMAPFILEHEADER { // bmfh
        WORD    bfType;
        DWORD   bfSize;
        WORD    bfReserved1;
        WORD    bfReserved2;
        DWORD   bfOffBits;
} BITMAPFILEHEADER;
```

Members

bfType

Specifies the file type. It must be BM.

bfSize

Specifies the size, in bytes, of the bitmap file.

bfReserved1

Reserved; must be zero.

bfReserved2

Reserved; must be zero.

bfOffBits

Specifies the offset, in bytes, from the **BITMAPFILEHEADER** structure to the bitmap bits.

The **BITMAPINFOHEADER** structure contains information about the dimensions and color format of a device-independent bitmap (DIB).

```
typedef struct tagBITMAPINFOHEADER{ // bmih
   DWORD  biSize;
   LONG   biWidth;
   LONG   biHeight;
   WORD   biPlanes;
   WORD   biBitCount
   DWORD  biCompression;
   DWORD  biSizeImage;
   LONG   biXPelsPerMeter;
   LONG   biYPelsPerMeter;
   DWORD  biClrUsed;
   DWORD  biClrImportant;
} BITMAPINFOHEADER;
```

Members

biSize

Specifies the number of bytes required by the structure.

biWidth

Specifies the width of the bitmap, in pixels.

biHeight

Specifies the height of the bitmap, in pixels. If **biHeight** is positive, the bitmap is a bottom-up DIB and its origin is the lower left corner. If **biHeight** is negative, the bitmap is a top-down DIB and its origin is the upper left corner.

biPlanes

Specifies the number of planes for the target device. This value must be set to 1.

biBitCount

Specifies the number of bits per pixel. This value must be 1, 4, 8, 16, 24, or 32.

biCompression

Specifies the type of compression for a compressed bottom-up bitmap (top-down DIBs cannot be compressed). It can be one of the following values:

biXPelsPerMeter

Specifies the horizontal resolution, in pixels per meter, of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that best matches the characteristics of the current device.

biYPelsPerMeter

Specifies the vertical resolution, in pixels per meter, of the target device for the bitmap.

biClrUsed

Specifies the number of color indices in the color table that are actually used by the bitmap. If this value is zero, the bitmap uses the maximum number of colors corresponding to the value of the **biBitCount** member for the compression mode specified by **biCompression**.

If **biClrUsed** is nonzero and the **biBitCount** member is less than 16, the **biClrUsed** member specifies the actual number of colors the graphics engine or device driver accesses. If **biBitCount** is 16 or greater, then **biClrUsed** member specifies the size of the color table used to optimize performance of Windows color palettes. If **biBitCount** equals 16 or 32, the optimal color palette starts immediately following the three doubleword masks.

If the bitmap is a packed bitmap (a bitmap in which the bitmap array immediately follows the **BITMAPINFO header and which is referenced by a single pointer), the biClrUsed** member must be either 0 or the actual size of the color table.

biClrImportant

Specifies the number of color indices that are considered important for displaying the bitmap. If this value is zero, all colors are important.

4.3 Obtain Standard AVI File

The first step of the flowchart, obtaining a standard AVI file, is very simple. There are many AVI files readily available on the Internet. For this thesis, certain considerations are made in the selection of suitable AVI files. First, to simulate the type of video typical of a common application such as video-telephony, AVI files with frame rates of approximately 10-15 frames per second are selected. Another consideration that must be made in the selection of AVI files for the "Proof-of-Concept" in this thesis involves the subject of colour. Only grayscale video files are used in this thesis. The reasoning behind this relates to the fact that filtering pixels with colour is a straight-forward extension of the method used to grayscale pixels. There are two methods for performing this filtering: filtering the three primaries separately, and filtering only the luminance values. Both of the methods are discussed in detail in [1].

4.4 Extract Individual Frames

The next step in the implementation involves extracting each frame of video from the file. Refer to the code supplied at the end of this chapter for the actual code used in this and subsequent AVI-related steps. As mentioned in the introduction, the Microsoft SDK has many functions for manipulating AVI files. To use these functions, the AVIFile library must be initialized using AVIFileInit. The AVI file is then opened using AVIFileOpen. This function can also be used to create new AVI files for writing. The next step is to obtain the video stream using AVIFileGetStream. As mentioned, AVI files can contain multiple streams, where one stream may be video and the others audio. The audio stream

is ignored in this step. With the video stream extracted, the original AVI file can now be closed using AVIFileClose. By using the functions AVIStreamStart and AVIStreamEnd, the original number of frames can be calculated. The use of AVIStreamGetFrameOpen is used to prepare for the extraction of a frame from the stream. Then the function AVIStreamGetFrame is called. This function returns a pointer to a specified frame of video as a Device Independent Bitmap (DIB). The DIB format is also commonly known as the Microsoft Bitmap (BMP) format, which was discussed in the introduction of this chapter.

4.5 Extract Raw Pixel Data from Frames

The BMP file format is a very simple format to extract data from. Since only grayscale (8bpp) images are used, each entry in the bitmap data is an index to the Red-Green-Blue (RGB) value stored in the palette. The fact that the images are grayscale means that R=G=B. If the images were not grayscale, luminance (grayscale) values could still be obtained from the RGB colour values using the following equation[1].

$$Y = 0.3R + 0.59G + 0.11B \tag{5.1}$$

Equation 5.1 is based on the relative sensitivity of the human eye to the different primary colours. By using Equation 5.1, pixel data can be extracted from either a colour or grayscale palette.

4.6 Apply 3-D Filter to Raw Data

This section comprises the central part of the thesis. The previous steps in the implementation are primarily for the purpose of accessing the raw video data. The following algorithm explains the process involved in filtering a sequence of images with a three-dimensional filter. Refer to Figure 4.2 for a graphical representation of the process.

```
Clear wx and wy
For n=0,1,2,...,image_depth-1
{
   Transfer nth frame of video to 1st frame of wx
   wx[0][j][k] = video[n][j][k] for j=0,1,2,...,image_height-1,
                                    k=0,1,2,...,image_width-1

   For m1=0,1,2,...,image_height-1
   {
      For m2=0,1,2,...,image_width-1
         {
```

$$wy[0][m_1][m_2] = \sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} a_{ijk}\, wx[i][m_1 - j][m_2 - k] - \sum_{\substack{i=0 \\ (i+j+k \neq 0)}}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} b_{ijk}\, wy[i][m_1 - j][m_2 - k]$$

```
         }
   }

   Transfer 1st frame of wy into output video
   Output[n][j][k]=wy[0][j][k] for j=0,1,2,...,image_height-1,
                                    k=0,1,2,...,image_width-1

   Shift frames of wx and wy
```

$$frame_{i+1} = frame_i$$

```
}
```

4.7 Reconstruct Frames

With the data filtered, each frame is placed back in BMP format to prepare for insertion into the AVI format. This is a simple task which involves re-attaching the file and bitmap headers to the new raw data. Since the data is entirely 8-bit luminance values, they already act as indexes into a linear colour palette ranging from zero to 255 with each R, G, and B entry equal. If the original image was grayscale, this palette already exists. If the original image had been colour and was converted to grayscale for processing then the old colour palette must be replaced with the linear grayscale palette mentioned above. The original file header remains unchanged, while the only fields of the original bitmap header that differ after the filtering are biWidth, biHeight, and biSizeImage. The width and height fields will each be double the original value, while the image-size field will be four times larger.

4.8 Reconstruct AVI File

The final step of the flowchart of Figure 4.1 is the reconstruction of the AVI file. Like the extraction of frames from the file, this step involves using specific functions in the AVIFile library. Similar to the reconstruction of the BMP frames where the original headers are reused with only slight modifications, much of the stream header information can be reused from the original. In this case, the following fields of the header are changed: dwRate is doubled, dwLength is doubled, dwSuggestedBufferSize is quadrupled, and the length and width of rcFrame are each doubled. Using the modified header, a new stream can be created by using the function AVIFileCreateStream. The format of the stream is

then set using AVIStreamSetFormat. With this done, the stream is now ready to have a filtered frame inserted by using AVIStreamWrite. These steps are required for writing the first frame of filtered video data to the file. Now as each subsequent frame of filtered video is obtained, it can be added to the stream using AVIStreamWrite. The resources from decompressing the frame then need to be released by using AVIStreamGetFrameClose. Both streams, old and new, are closed using AVIStreamClose, and AVIFileClose is used to close the new AVI file. The function AVIFileExit is then used to exit the AVIFile library.

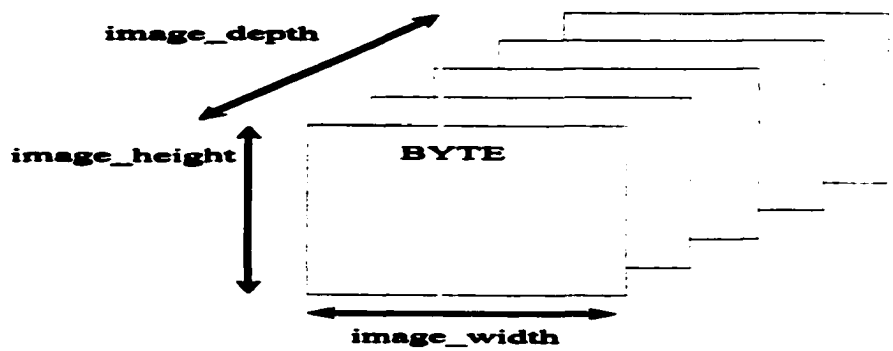


Figure 4.2(a): Original moving image sequence.

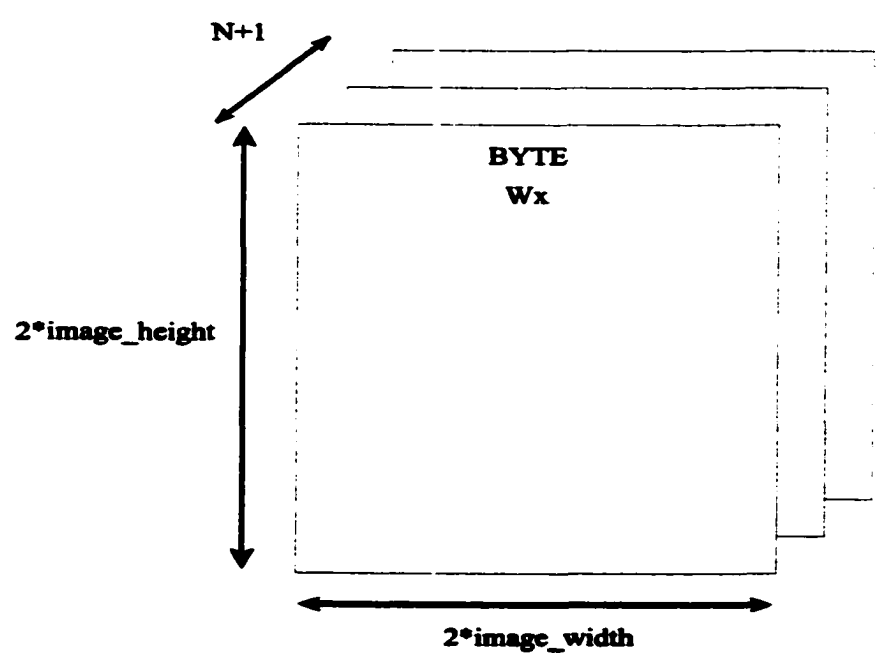


Figure 4.2(b): Wx image buffer.

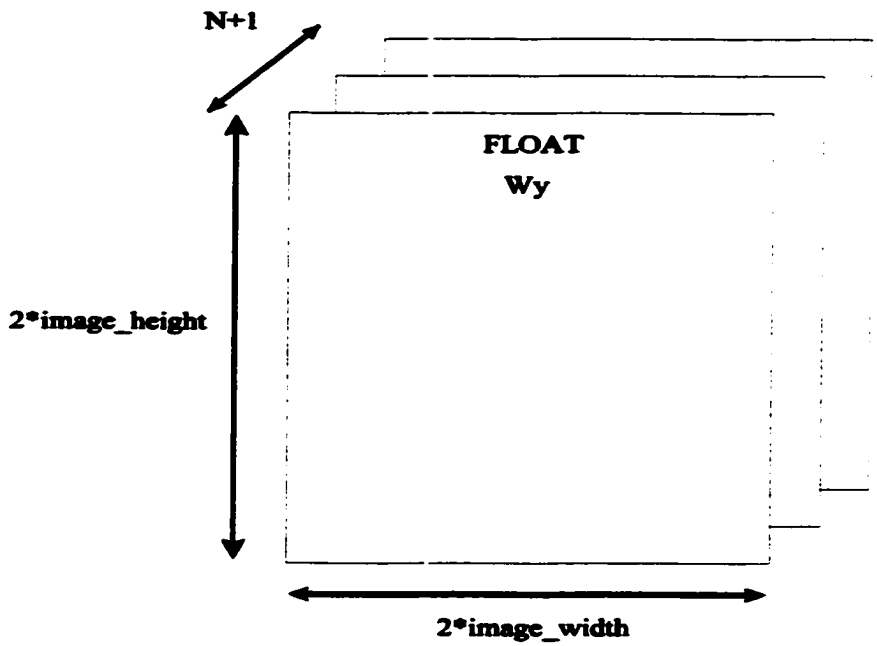


Figure 4.2(c): Wy image buffer.

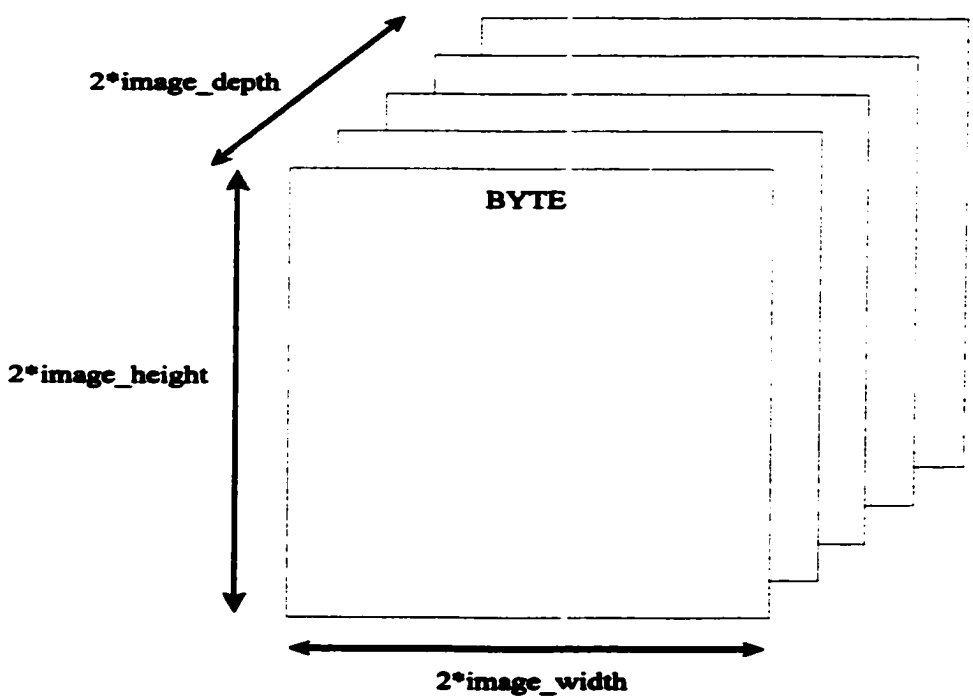


Figure 4.2(d): Filtered moving image sequence.

4.9 Sample AVI Code

```
⋮

// Initialize AVIFile library
AVIFileInit();

// Open AVI file for reading
hr = AVIFileOpen(&pFile,m_lpstrFileName,OF_READ,NULL);
if(hr!=AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Opening the Input File.");
        return FALSE;
}

// Create new AVI file for writing
hr = AVIFileOpen(&pFileNew,m_lpstrNewFileName,OF_WRITE|OF_CREATE,NULL);
if(hr!=AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Creating the Output File.");
        return FALSE;
}

// Open AVI stream for reading
hr = AVIFileGetStream(pFile,&pStream,streamtypeVIDEO,0);
if(hr!=AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Opening the Input Stream.");
        return FALSE;
}

// Close original AVI file
AVIFileClose(pFile);

// Calculate number of frames in stream
numFrames = AVIStreamEnd(pStream)-AVIStreamStart(pStream);

⋮

// Prepare to decompress video frames from stream
getFrameObj = AVIStreamGetFrameOpen(pStream,NULL);

// Obtain address of first decompressed video frame
tempFramePtr = (BYTE *)AVIStreamGetFrame(getFrameObj,0);

⋮

// Get header from old stream
hr = AVIStreamInfo( pStream, &strHdrOld, sizeof(strHdrOld) );
```

```
if(hr != AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Reading Old Stream Header.");
        return FALSE;
}

// Fill in the header for the new video stream
memset(&strHdrNew,0,sizeof(strHdrNew)); // Set strHdrNew to zero
strHdrNew.fccType = streamtypeVIDEO;    // stream type
strHdrNew.fccHandler = 0;           // Compressor Code
strHdrNew.dwScale = strHdrOld.dwScale;  // Time Scale
strHdrNew.dwRate = 2*strHdrOld.dwRate;  // Frames per second
strHdrNew.dwLength = 2*strHdrOld.dwLength;      // Number of frames
strHdrNew.dwSuggestedBufferSize = 4*bmiHeader.biSizeImage;      // buffer size
SetRect(&strHdrNew.rcFrame,0,0,2*bmiHeader.biWidth,2*bmiHeader.biHeight); // rectangle for
stream

// Create the new stream
hr = AVIFileCreateStream(pFileNew,&pStreamNew,&strHdrNew);
if(hr != AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Creating the Output Stream.");
        return FALSE;
}


                                        ⋮

// Set format of new stream
hr = AVIStreamSetFormat(pStreamNew,0,framePtr,
                                bmiHeader.biSize +
                                bmiHeader.biClrUsed*sizeof(RGBQUAD));
if(hr != AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Setting the Output Stream Format.");
        return FALSE;
}

// Write frame to new stream
hr = AVIStreamWrite(pStreamNew,0,1,
                        framePtr + imageOffset,
                        4*bmiHeader.biSizeImage,
                        AVIIF_KEYFRAME, NULL, NULL);
if(hr != AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Writing to the Output Stream.");
        return FALSE;
}

        // Write frame to new stream
        hr = AVIStreamWrite(pStreamNew,frame,1,
```

```
                                        framePtr + imageOffset,
                                        4*bmiHeader.biSizeImage,
                                        AVIIF_KEYFRAME, NULL, NULL);
                if(hr != AVIERR_OK)
                {
                        AfxMessageBox("An Error Occurred Writing to the Output Stream.");
                        return FALSE;
                }


        // Close the files and streams
        AVIStreamGetFrameClose(getFrameObj);
        AVIStreamClose(pStream);
        AVIStreamClose(pStreamNew);
        AVIFileClose(pFileNew);

        AVIFileExit();
        return TRUE;    // function completed successfully
}
```

Chapter 5: Results

5.1 Introduction

All results given in this chapter except plotting, which is done using MATLAB, are generated using a computer program developed using Microsoft Visual C++. This program designs a 3-D IIR filter using the Modified Shank's Method of Chapter 3, and uses it to perform filtering of an AVI file using the process given in Chapter 4. This software implementation of a 3-D filter provides a basis for forming conclusions about the validity of the theory given in the preceding chapters. These conclusions are provided in Chapter 6.

5.2 Filter Design Results

After starting the program Filter3D, selecting 'New' from the toolbar or the File menu presents a dialog box requesting parameters for the design of the 3D filter. Figure 5.1 shows this dialog box. The values given in Figure 5.1 are the default values for the filter. The results in this chapter are generated using a value of 32 as the number of samples. The default values are used for the other design options. By clicking the OK button, the filter is designed using the Modified Shank's Method discussed earlier in the thesis. The resulting filter coefficients are given in Figure 5.2.

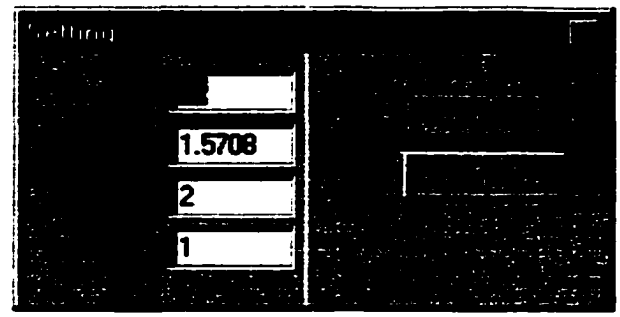
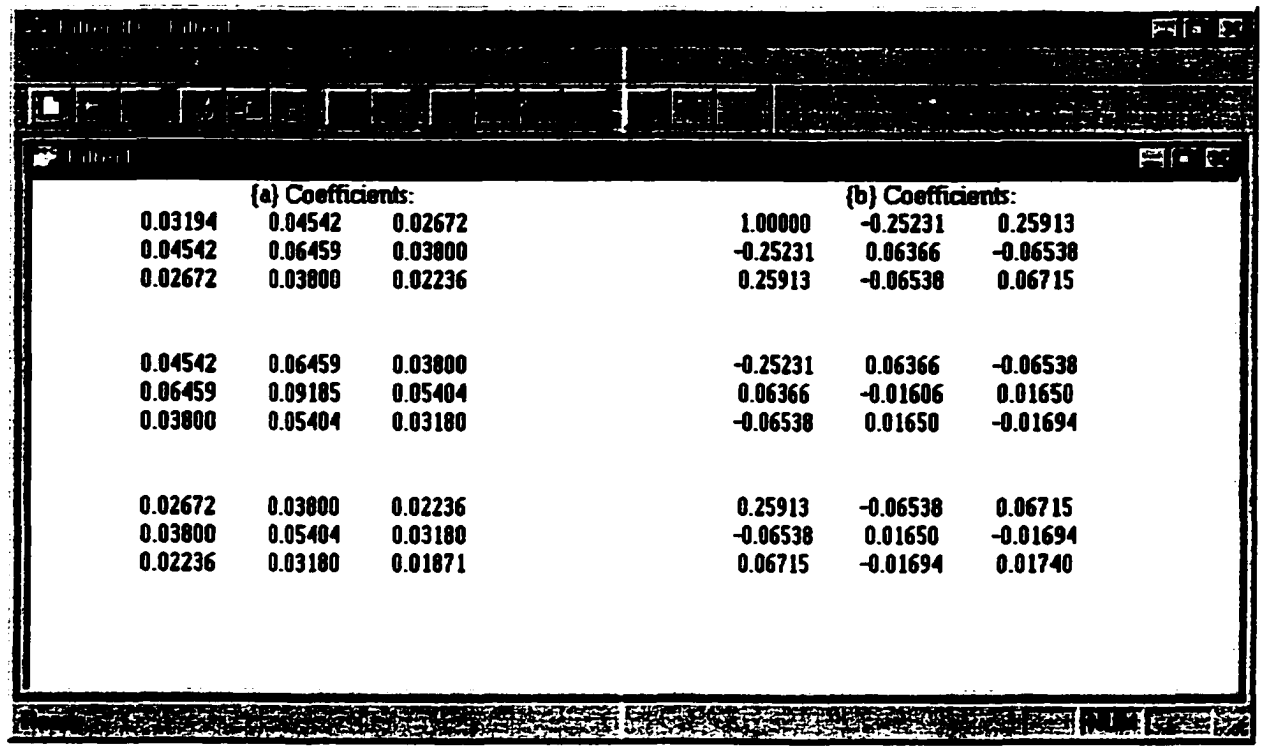


Figure 5.1: Filter Settings Dialog Box



{a} Coefficients:

0.03194	0.04542	0.02672
0.04542	0.06459	0.03800
0.02672	0.03800	0.02236
0.04542	0.06459	0.03800
0.06459	0.09185	0.05404
0.03800	0.05404	0.03180
0.02672	0.03800	0.02236
0.03800	0.05404	0.03180
0.02236	0.03180	0.01871

{b} Coefficients:

1.00000	-0.25231	0.25913
-0.25231	0.06366	-0.06538
0.25913	-0.06538	0.06715
-0.25231	0.06366	-0.06538
0.06366	-0.01606	0.01650
-0.06538	0.01650	-0.01694
0.25913	-0.06538	0.06715
-0.06538	0.01650	-0.01694
0.06715	-0.01694	0.01740

Figure 5.2: Coefficients of designed 3-D filter

Figure 5.3 shows the magnitude response of the designed filter when $\omega 1$ is held constant at zero radians/sec. Note that $\omega 1$ is within the passband of the filter, and the characteristics of the filter are acceptable. Figure 5.4 shows the magnitude response when $\omega 1$ is held constant at 0.98 radians/sec. The value of $\omega 1$ is still within the passband of the filter and the characteristics are again acceptable. Figure 5.5 shows the magnitude response when $\omega 1$ is held constant at 2.16 radians/sec. This value of $\omega 1$ is outside the passband, and therefore the magnitude response is very nearly zero. Figure 5.6 shows the magnitude response when $\omega 1$ is held constant at pi radians/sec. The value of $\omega 1$ is again outside the passband, and the magnitude response is again very nearly zero.

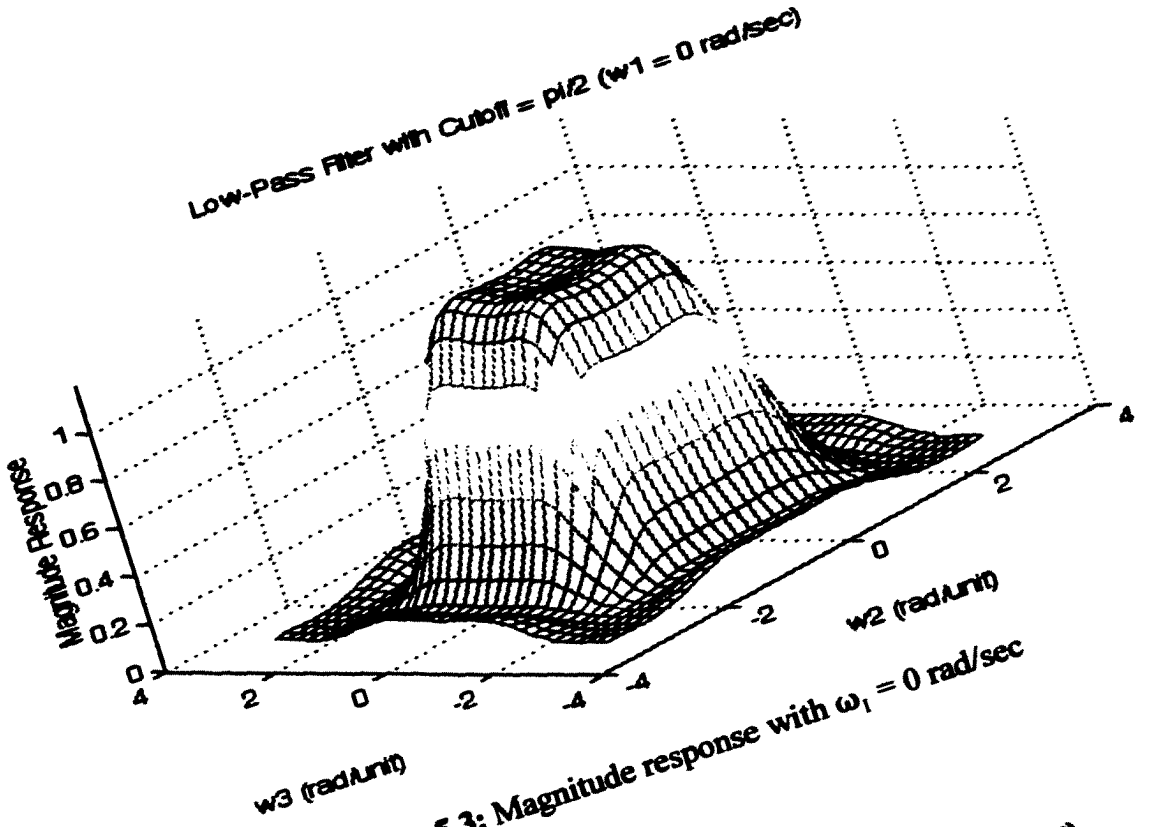


Figure 5.3: Magnitude response with $\omega_1 = 0$ rad/sec

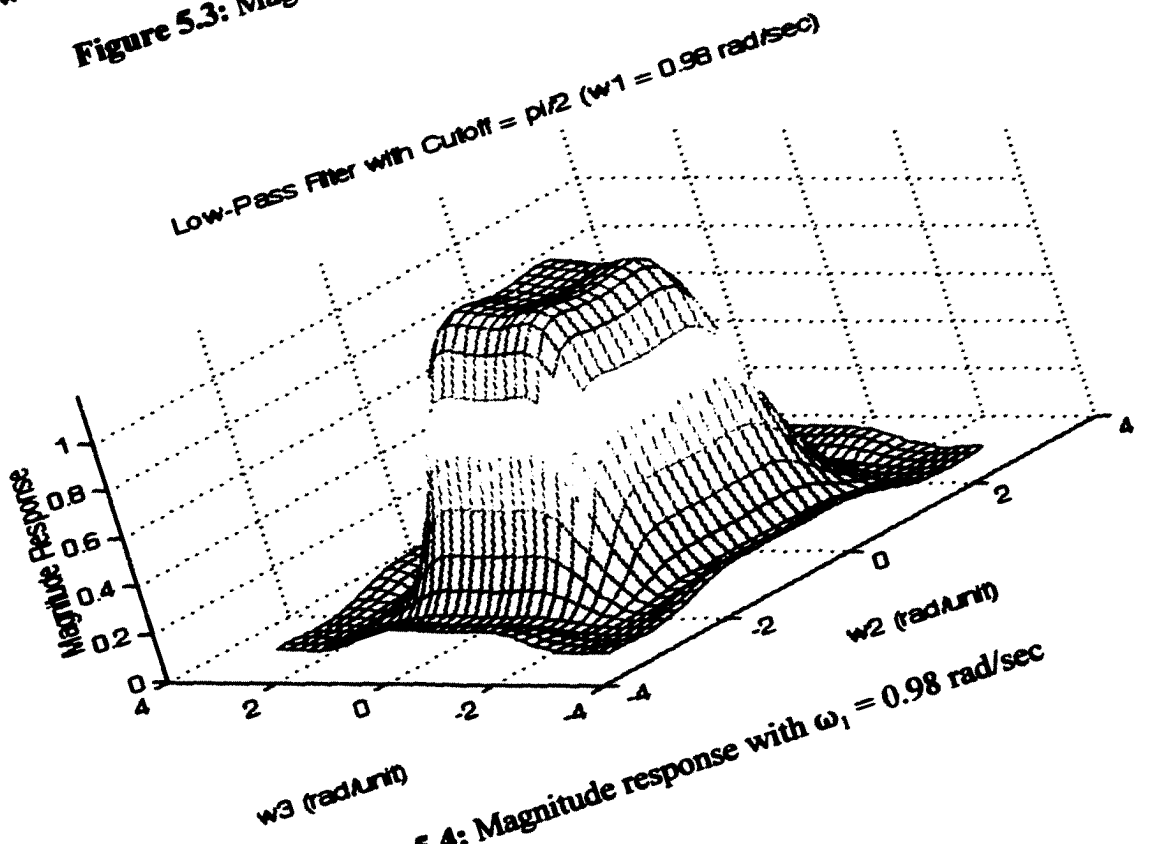


Figure 5.4: Magnitude response with $\omega_1 = 0.98$ rad/sec

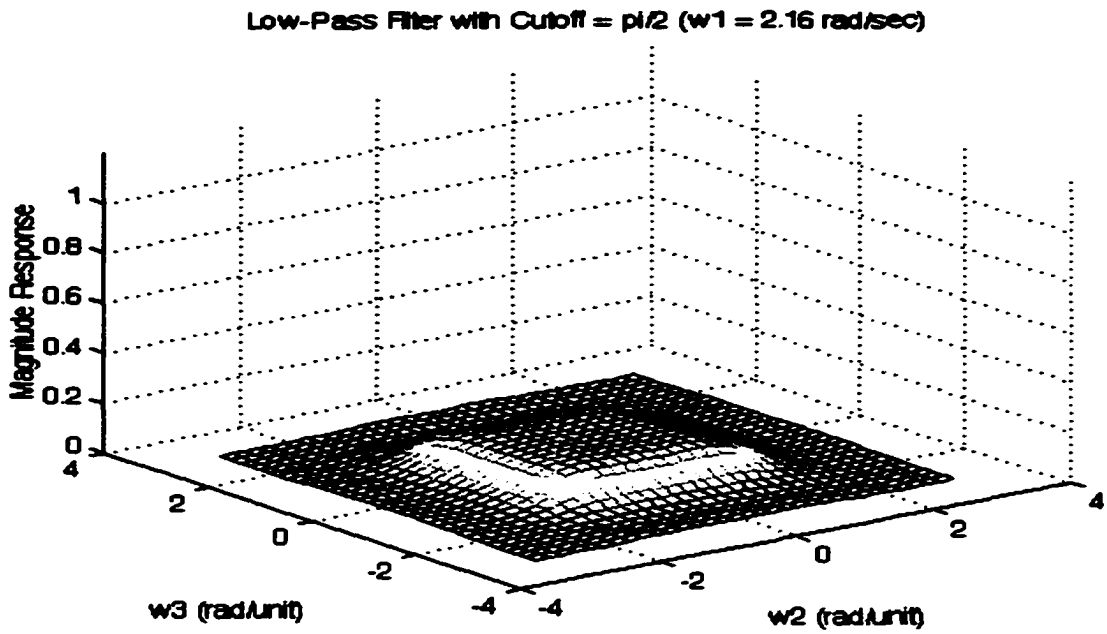


Figure 5.5: Magnitude response with $\omega_1 = 2.16$ rad/sec

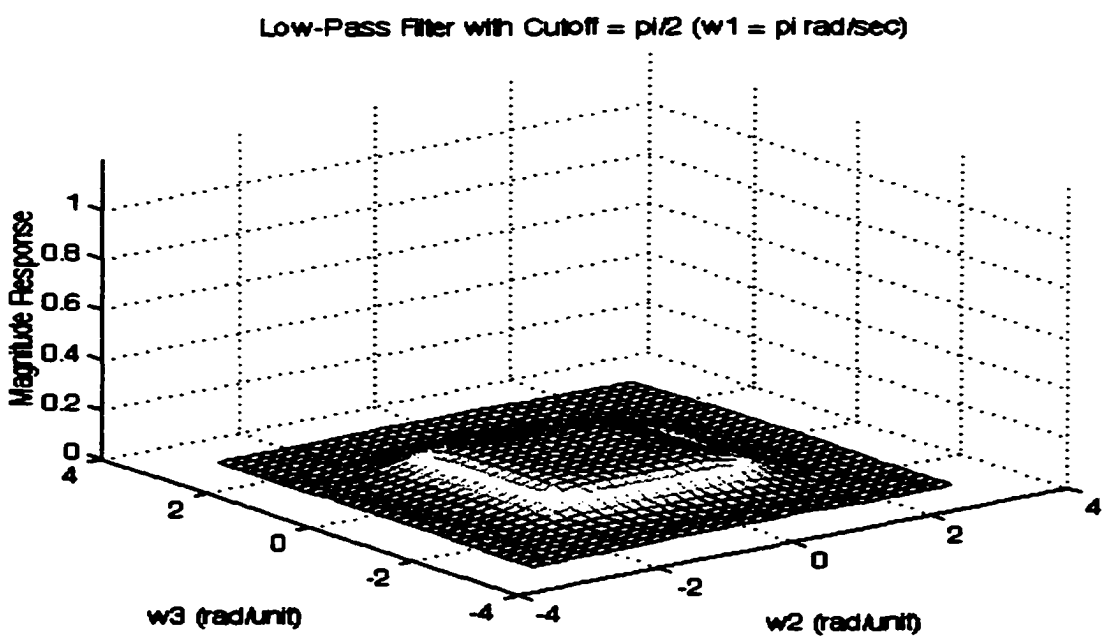


Figure 5.6: Magnitude response with $\omega_1 = \pi$ rad/sec

The plots of Figures 5.3 through 5.6 show that the filter design method used yields an acceptable magnitude response. However, as stated earlier in the thesis, it is also very important to have linear or near-linear phase in the passband of the filter. Figure 5.7 shows the phase response of the filter when $\omega 1=0$ rad/sec. Note that the response appears moderately flat within the passband region. Figure 5.8 shows an approximation to the group delay of the filter with respect to $\omega 3$. It is only an approximation since the resolution between points is finite, but it is sufficient to give an idea of the linearity of the phase response. Note that for values of $\omega 3$ within the passband, there is very little deviation in the phase response. All significant deviation lies outside the passband, so any distortion is attenuated. Figure 5.9 and 5.10 show the same thing except with $\omega 1$ at a value of 0.98 rad/sec. Figure 5.11 shows the phase response of the filter when $\omega 1$ is fixed at 2.16 rad/sec. Note that the phase response at this value of $\omega 1$ is non-linear. However, by examination of Figure 5.12 it can be seen that the non-linearity occurs outside the passband of the filter. Therefore, any resulting phase distortion will be attenuated.

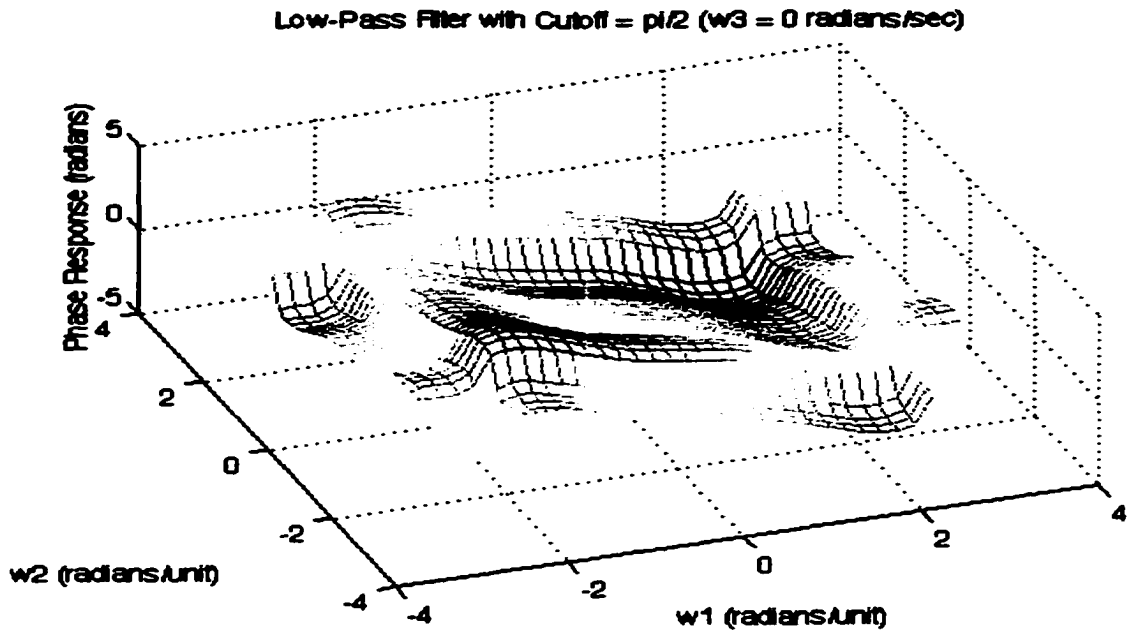


Figure 5.7: Phase response with $\omega_1 = 0$ rad/sec

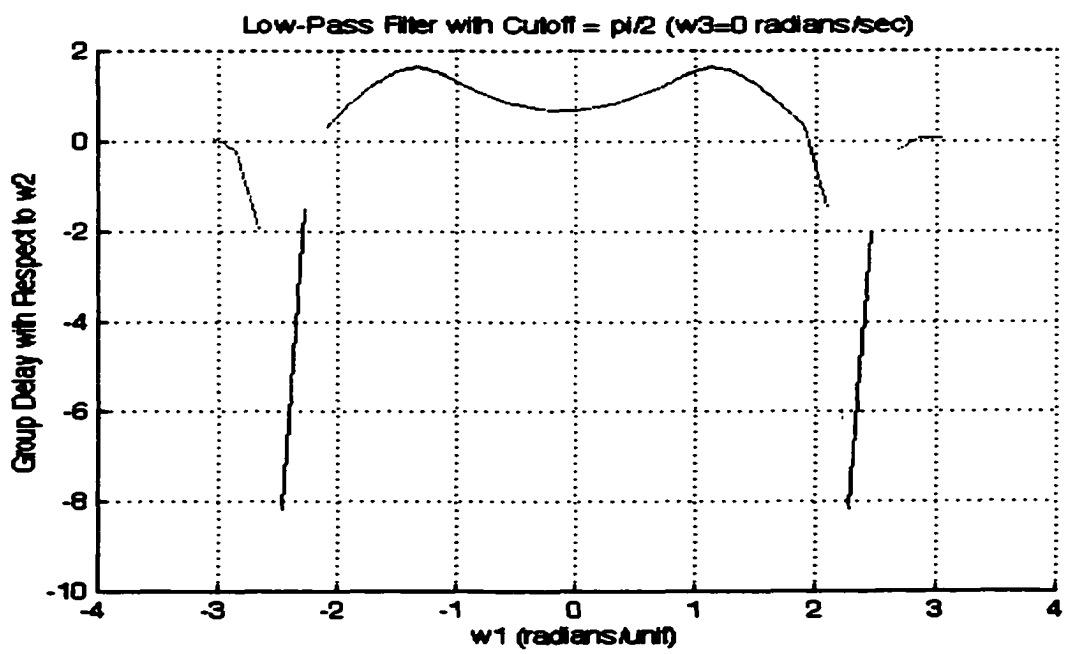


Figure 5.8: Group delay with $\omega_1 = 0$ rad/sec

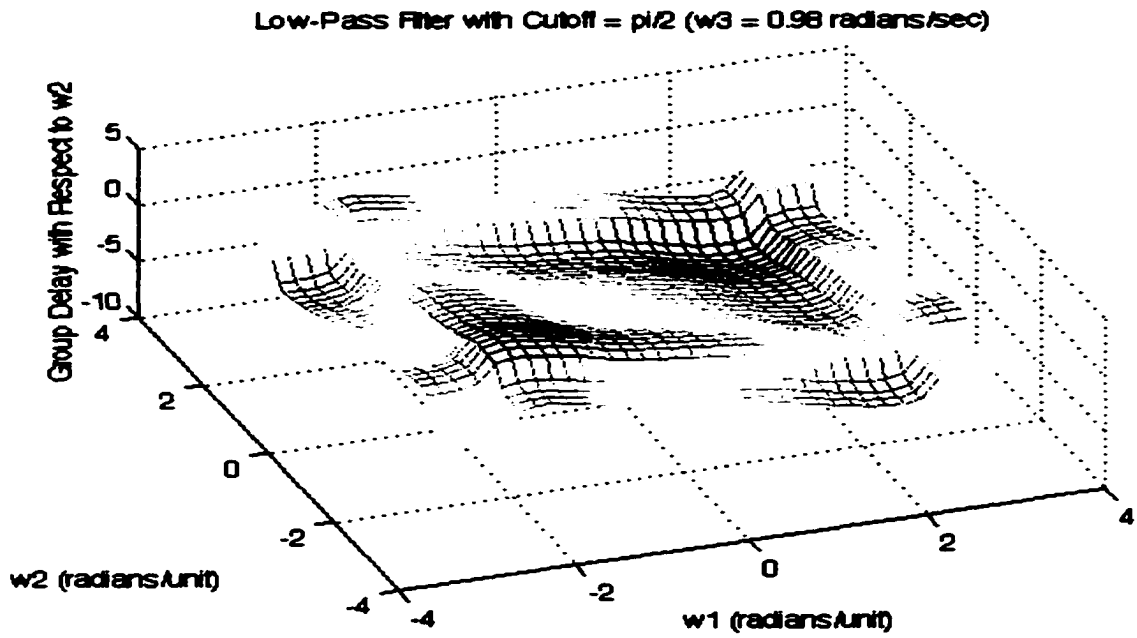


Figure 5.9: Phase response with $\omega_1 = 0.98$ rad/sec

Figure 5.10: Group delay with $\omega_1 = 0.98$ rad/sec

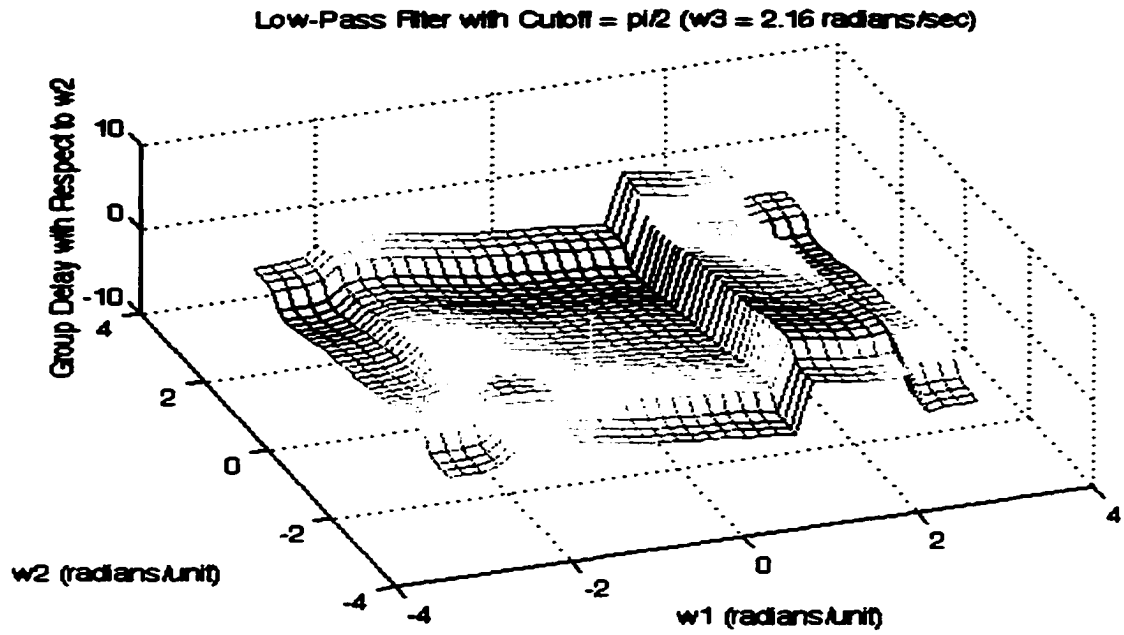


Figure 5.11: Phase response with $\omega_1 = 2.16$ rad/sec

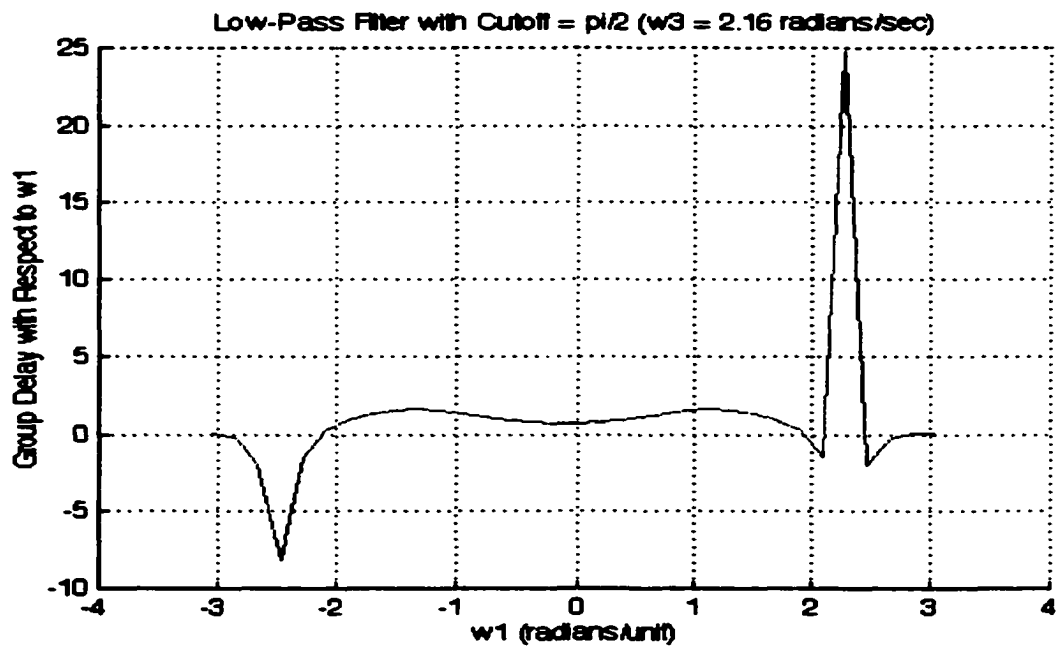


Figure 5.12: Group delay with $\omega_1 = 2.16$ rad/sec

5.3 Video Filtering Results

While it is gratifying to see that the 3D filter design method given in this thesis is effective, it is not the primary goal of this investigation. The main purpose is to verify that these filters can be effectively used to increase the resolution of moving images. By following the implementation algorithm given in the previous chapter, various AVI video files were interpolated using the above 3D filter. The following figures show the results from one of these files.

First let us demonstrate that the number of samples has been increased. Figure 5.13 shows the file properties of the original AVI file compared with the file properties of the filtered AVI file. Note that the width and height are both doubled, and the number of frames is

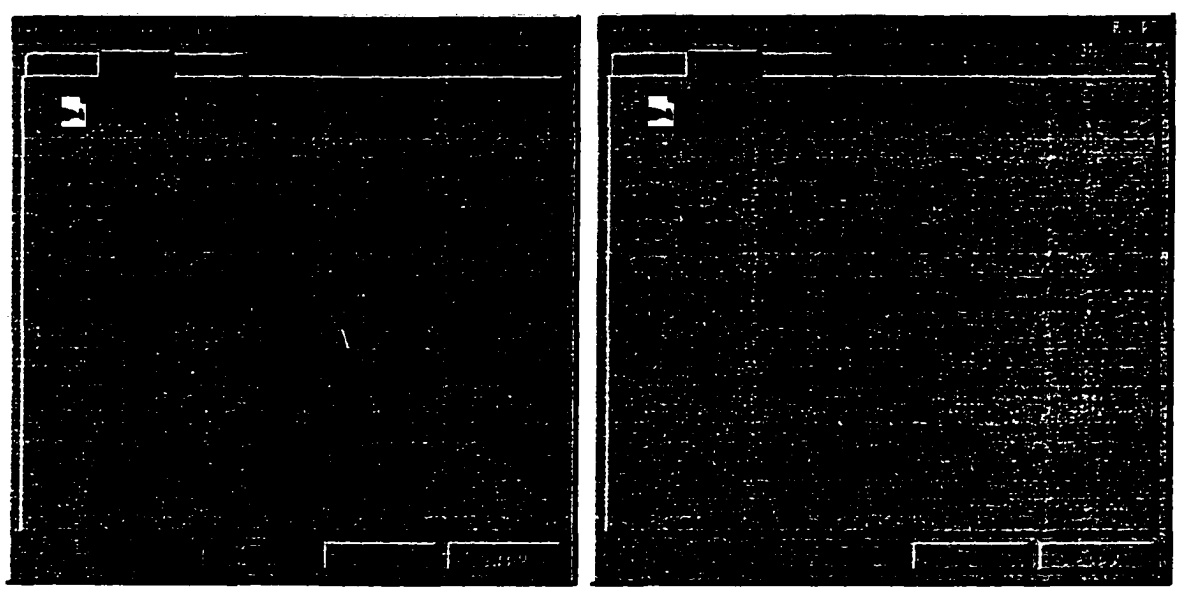


Figure 5.13: Comparison of File Properties dialog boxes.

also doubled. This shows that there is indeed eight times more samples in the filtered video than in the original, but it gives no indication of the quality of this new video.

Figure 5.14 shows a frame of the original video file compared with its equivalent filtered frame. Note that the filtered image is double the width and height of the original.

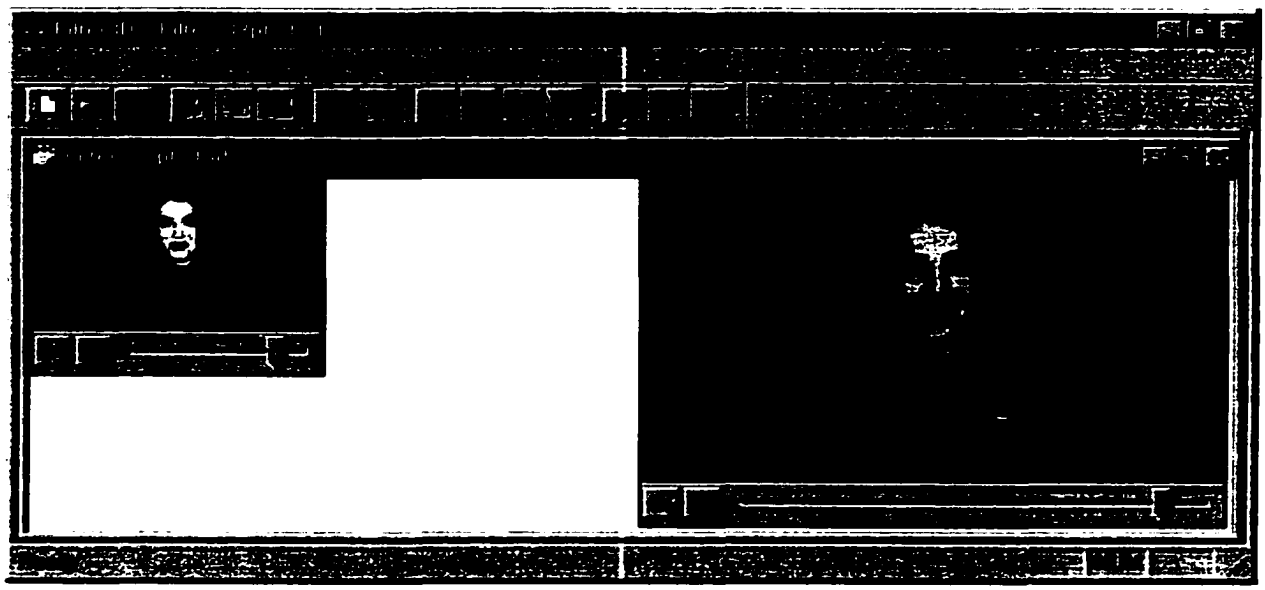


Figure 5.14: Video single frame comparison.

Now examine Figure 5.15. It also shows a frame of the original video sequence compared with a frame from the filtered video sequence, but this time both frames are zoomed in to show the resolution difference.

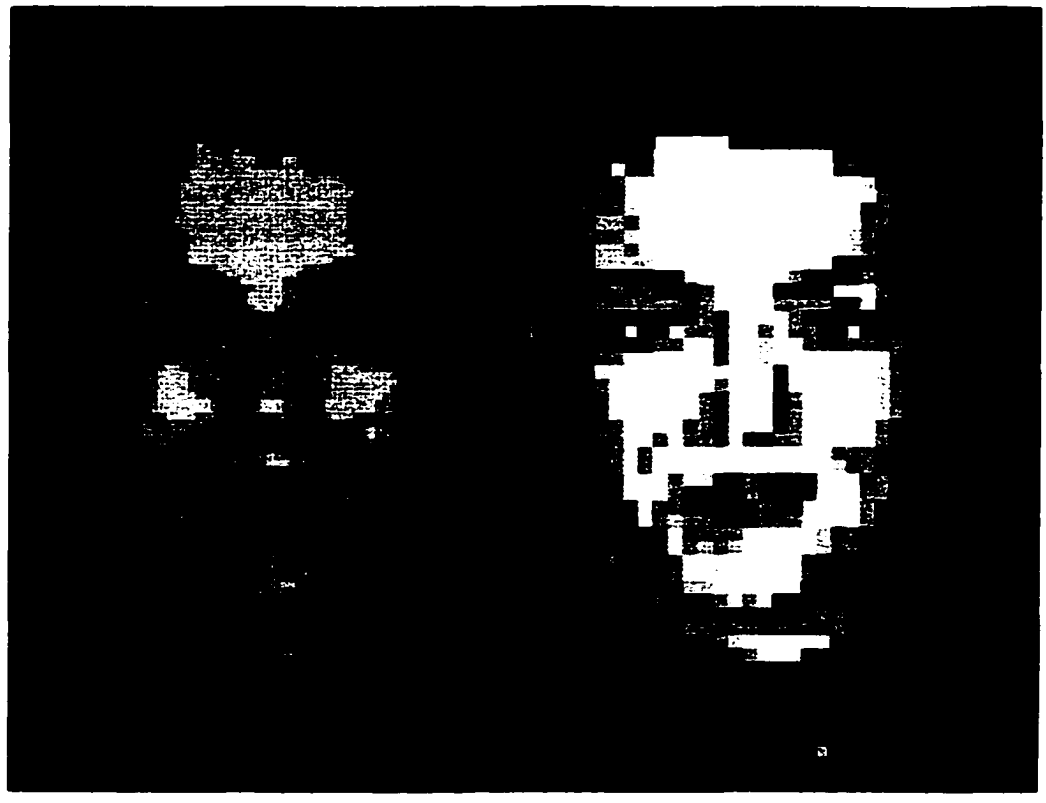


Figure 5.15: Video single frame zoomed comparison.

The filtered frame is on the left, and the original frame is on the right. Note that for every one pixel in the original image, the filtered image has four pixels. Also note the improved definition of features such as the nose, eyes, and ears. The pixelation effect along the edge of the collar is also greatly reduced.

While Figure 5.14 and Figure 5.15 show an impressive increase of resolution in the individual frames of the video sequence, these results could have been obtained by using a 2D filter. The advantage of the 3D filter in this application lies in its ability to also increase the resolution along the time axis by interpolating frames. The result of this can be seen in Figure 5.16. The top two images are equivalent frames from the original and filtered video

sequences respectively (left to right). The bottom left image is the next frame of the original sequence, and the frame to its right is the equivalent frame of the filtered sequence. The frame between the two filtered frames is an interpolated frame that does not exist in the original sequence. Note the mouth is open in the first frame, and is closed in the next frame of the original sequence. Now examine the filtered sequence and note that the mouth is first open (as in the original), then the mouth is partially open, and then the mouth is closed (as in the original). The frame with the mouth partially open did not exist in the original sequence. This frame was successfully interpolated and shows detail that is not visible in the original sequence. These results show that the 3D filter successfully increased the resolution of the video sequence in all three dimensions.

Figure 5.16(a): Original Sequence (time = T) (Mouth is open)

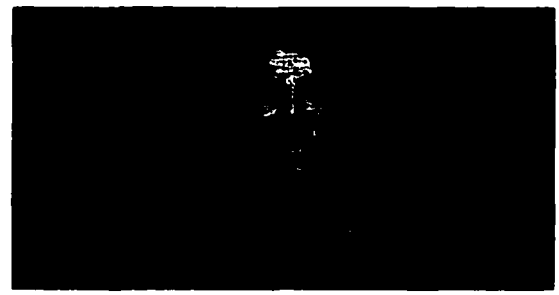


Figure 5.16(c): Interpolated Sequence (time = T) (Mouth is open)

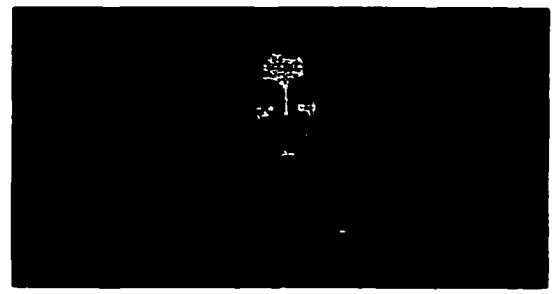


Figure 5.16(d): Interpolated Sequence (time = T+dt/2) (Mouth is partially open)

Figure 5.16(b): Original Sequence (time = T+dt) (Mouth is closed)

Figure 5.16(e): Interpolated Sequence (time = T+dt) (Mouth is closed)

Chapter 6: Summary and Conclusions

6.1 Summary

Chapter 1 of this thesis began by introducing the concept of moving images and giving an overview of various types of digital filters and their design methodologies. It finished by comparing some popular video formats, and giving some examples of the current applications of digital filters.

Chapter 2 began by discussing the Sampling Theorem, which is central to the understanding of digital interpolation. It then explored two methods of one-dimensional interpolation: zero-padding and sample replication. The chapter concluded by extending these concepts to three dimensions for use with three-dimensional digital signals.

The procedure of designing three-dimensional IIR filters using the Modified Shank's Method was presented in detail in Chapter 3. The two-dimensional spatial method was extended to the three-dimensional space-time domain. The chapter concluded by deriving the three-dimensional Fast Fourier Transform (FFT).

Chapter 4 tied all the theory together from the previous chapters to provide an implementation method by which a moving image sequence could be interpolated with a three-dimensional digital filter. It began by giving a description of the AVI and BMP file formats, and then describing how the raw pixel data could be extracted from these

formats. A scheme for applying the 3-D filter was given next, followed by a method to reinsert the raw filtered data back into an AVI file.

Chapter 5 provided results to demonstrate the validity of the theory in Chapter 2 and Chapter 3, and the validity of the implementation method in Chapter 4.

6.2 Conclusions

This thesis is concerned with the use of 3-D digital filters in multimedia applications. Specifically, it is interested in using three-dimensional digital interpolation filters to increase the resolution of moving image sequences in three dimensions. By examination of the results given in Chapter 5, it is clear that both the theory and the proposed implementation given in the thesis are sound. The designed 3-D IIR filter possesses a steep transition band and has near-linear phase response in the passband. After applying the filtering algorithm given in Chapter 4, the video file's resolution is increased by a factor of two in each dimension for a total resolution improvement by a factor of eight. In Chapter 1, the purpose of the thesis was given as: "... a digital 3-D interpolation filter is to be designed which performs inter-pixel and inter-frame interpolation, resulting in increased horizontal resolution, vertical resolution, and temporal resolution (frame rate) of a moving image sequence." The results of Chapter 5 clearly demonstrate that the goal of this thesis has been achieved, and that 3-D filters have application to the field of multimedia.

References

[1] M. A. Sid-Ahmed, "Image Processing: Theory, Algorithms, and Applications," McGraw Hill, New York, 1994.

[2] R. King, M. Ahmadi, R. Gorgui-Naguib, A. Kwabwe, M. Azimi-Sadjadi, "Digital Filtering in One and Two Dimensions: Design and Applications," Plenum Press, New York, 1989.

[3] A. Antoniou, "Digital Filters: Analysis, Design, and Applications," McGraw Hill, Toronto, 1993.

[4] T. S. Huang, "Two Dimensional Windows," *IEEE Trans. Audio and Electroacoustics*, AU-20, no. 1, March 1972, pp 88-90.

[5] T. C. Speake and R. M. Mersereau, "A Note on the Use of Windows for Two Dimensional FIR Filter Design," *IEEE Trans. Acoustics, Speech, and Signal Processing*, ASSP-29, no. 1, Feb. 1981, pp 125-127.

[6] I. S. El-Feghi, "Design of Three Dimensional Digital Filters," MASc Thesis, University of Windsor, Windsor, 1999.

[7] C. Charalambous, "Design of 2-Dimensional Circularly-Symmetric Digital Filters," *IEEE Proc.* 129, Part G, no. 2, pp 47-54, 1982.

[8] K. Rajan and M. N. S. Swamy, "Design of Separable Denominator 2-Dimensional Digital Filters Possessing Real Circularly Symmetric Frequency Responses," *IEEE Proc.* 129, Part G, no. 2, pp 235-240, 1982.

[9] T. Hinmoto and K. Harada, "Design of 3-D Separable Denominator Digital Filters Using Minimal Decomposition and Balanced Realization," *Electronics and Communications in Japan*, Part 3, Vol 77, no. 10, 1994.

[10] D. Goodman, "Some Difficulties with Double Bilinear Transformation in 2-D Digital Filter Design Transfer Function," *IEEE Trans. On Circuits and Systems*, Vol. CAS-25, no. 6, pp 340-343, June 1978.

[11] P. K. Rajan, H. C. Reddy, M. N. S. Swamy, and V. Ramchandran, "Generation of Two Dimensional Digital Function Without Nonessential Singularities of the Second Kind," *IEEE Trans. Acoustics, Speech, and Signal Processing*, Vol. ASSP-28, no. 2, pp 216-223, April 1980.

[12] J. L. Shanks, S. Treitel, and J. H. Justice, "Stability and Sythesis of Two Dimensional Recursive Filters," *IEEE Trans. Audio and Electroacoustics*, AU-20, no. 2, June 1962, pp 115-128.

[13] H. C. Andrews and B. R. Hunt, "Digital Image Restoration," Prentice-Hall, Englewood Cliffs, NJ, 1977.

[14] W. K. Pratt, "Digital Image Processing," John Wiley and Sons, New York, 1978.

[15] J. W. Tukey, "Exploratory Data Analysis," Addison-Wesley, Reading, Massachusetts, 1971.

[16] R. E. Crochiere and L. R. Rabiner, "Multirate Digital Signal Processing," Prentice-Hall, Englewood Cliffs, NJ, 1983.

[17] J. D. Murray and W. vanRyper, "Encyclopedia of Graphics File Formats," O'Reilly & Associates, Sebastopol, 1996.

Appendix A

Source Code for Filter3D Program

```
// ChildFrm.cpp : implementation of the CChildFrame class
//

#include "stdafx.h"
#include "Filter3D.h"

#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CChildFrame

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
	//{{AFX_MSG_MAP(CChildFrame)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//   DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChildFrame construction/destruction

CChildFrame::CChildFrame()
{
	// TODO: add member initialization code here

}

CChildFrame::~CChildFrame()
{
}

BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CMDIChildWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CChildFrame diagnostics

#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
	CMDIChildWnd::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
```

```
{
        CMDIChildWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CChildFrame message handlers
```

```
//////////////////////////////////////////////////////////////////////////////
// Complex.cpp : implementation of the COMPLEX class
//

#include "stdafx.h"
#include "Filter3D.h"

#include "Complex.h"


COMPLEX::COMPLEX()
{
        Real = 0.0;
        Imag = 0.0;
}

COMPLEX::COMPLEX(double real, double imag)
{
        Real = real;
        Imag = imag;
}

double COMPLEX::GetReal(void) const
{
        return Real;
}

double COMPLEX::GetImag(void) const
{
        return Imag;
}

double COMPLEX::Magnitude(void)
{
        return(sqrt(Real*Real + Imag*Imag));
}

double COMPLEX::Phase(void)
{
        return(atan2(Imag,Real));
}

COMPLEX operator+( COMPLEX A, COMPLEX B )
{
        return COMPLEX( A.Real + B.Real,
          A.Imag + B.Imag );
}

COMPLEX operator-( COMPLEX A, COMPLEX B )
{
        return COMPLEX( A.Real - B.Real,
          A.Imag - B.Imag );
}

COMPLEX operator*( COMPLEX A, COMPLEX B )
{
        return COMPLEX( A.Real * B.Real - A.Imag * B.Imag,
```

```
            A.Real * B.Imag + A.Imag * B.Real );
}

COMPLEX operator*( COMPLEX A, double B )
{
        return COMPLEX( A.Real * B , A.Imag * B );
}

COMPLEX operator/( COMPLEX A, double B )
{
        return COMPLEX( A.Real / B , A.Imag / B );
}
```

```
// Filter3D.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Filter3D.h"

#include "MainFrm.h"
#include "ChildFrm.h"
#include "Filter3DDoc.h"
#include "Filter3DView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFilter3DApp

BEGIN_MESSAGE_MAP(CFilter3DApp, CWinApp)
	//{{AFX_MSG_MAP(CFilter3DApp)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//   DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
	// Standard file based document commands
	ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
	ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
	// Standard print setup command
	ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFilter3DApp construction

CFilter3DApp::CFilter3DApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CFilter3DApp object

CFilter3DApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CFilter3DApp initialization

BOOL CFilter3DApp::InitInstance()
{
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.
```

```
#ifdef _AFXDLL
	Enable3dControls();		// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	// Change the registry key under which our settings are stored.
	// You should modify this string to be something appropriate
	// such as the name of your company or organization.
	SetRegistryKey(_T("Steve McFadden - 1998"));

	LoadStdProfileSettings();  // Load standard INI file options (including MRU)

	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views.

	CMultiDocTemplate* pDocTemplate;
	pDocTemplate = new CMultiDocTemplate(
		IDR_FILTERTYPE,
		RUNTIME_CLASS(CFilter3DDoc),
		RUNTIME_CLASS(CChildFrame), // custom MDI child frame
		RUNTIME_CLASS(CFilter3DView));
	AddDocTemplate(pDocTemplate);

	// create main MDI Frame window
	CMainFrame* pMainFrame = new CMainFrame;
	if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
		return FALSE;
	m_pMainWnd = pMainFrame;

	// Parse command line for standard shell commands, DDE, file open
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	// Dispatch commands specified on the command line
//	if (!ProcessShellCommand(cmdInfo))
//		return FALSE;

	// The main window has been initialized, so show and update it.
	pMainFrame->ShowWindow(m_nCmdShow);
	pMainFrame->UpdateWindow();

	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA
```

```
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
		// No message handlers
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CFilter3DApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CFilter3DApp commands
```

```
// Filter3DDoc.cpp : implementation of the CFilter3DDoc class
//

#include "stdafx.h"
#include "Filter3D.h"

#include "Filter3DDoc.h"
#include "Filter3DSettingsDlg.h"
#include "ProcessingDlg.h"
#include <fstream.h>
#include "vfw.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFilter3DDoc

IMPLEMENT_DYNCREATE(CFilter3DDoc, CDocument)

BEGIN_MESSAGE_MAP(CFilter3DDoc, CDocument)
	//{{AFX_MSG_MAP(CFilter3DDoc)
//	ON_COMMAND(ID_VIDEO_PLAY, OnVideoPlay)
//	ON_COMMAND(ID_VIDEO_FILTER, OnVideoFilter)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFilter3DDoc construction/destruction

CFilter3DDoc::CFilter3DDoc()
{
	pi = 4.0*atan(1.0);
	m_dCutoffFreq = 1.5708;
	m_nNumSamples = 16;
	m_nOrder = 2;
	m_nOffset = m_nOrder-1;
}

CFilter3DDoc::~CFilter3DDoc()
{
}

BOOL CFilter3DDoc::OnNewDocument()
{
	if (!CDocument::OnNewDocument())
		return FALSE;

	unsigned i,j;

	// Obtain filter settings
	CFilter3DSettingsDlg dlg;
	dlg.m_dCutoffFreq = m_dCutoffFreq;
	dlg.m_nNumSamples = m_nNumSamples;
```

```
dlg.m_nOrder = m_nOrder;
dlg.m_nOffset = m_nOffset;
if( dlg.DoModal() == IDOK )
{
        m_dCutoffFreq = dlg.m_dCutoffFreq;
        m_nNumSamples = dlg.m_nNumSamples;
        m_nOrder = dlg.m_nOrder;
        m_nOffset = dlg.m_nOffset;

        // Allocate memory for impulse response
        m_pImpulseResponse = new double **[m_nNumSamples/2 + m_nOffset];
        for(i=0;i<(m_nNumSamples/2+m_nOffset);i++)
        {
                m_pImpulseResponse[i] = new double *[m_nNumSamples/2 + m_nOffset];
                for(j=0;j<(m_nNumSamples/2 + m_nOffset);j++)
                        m_pImpulseResponse[i][j] = new double [m_nNumSamples/2 + m_nOffset];
        }

        // Allocate memory for magnitude and phase response
        m_pMagnitudeResponse = new double **[m_nNumSamples+1];
        m_pPhaseResponse = new double **[m_nNumSamples+1];
        for(i=0;i<(m_nNumSamples+1);i++)
        {
                m_pMagnitudeResponse[i] = new double *[m_nNumSamples+1];
                m_pPhaseResponse[i] = new double *[m_nNumSamples+1];
                for(j=0;j<(m_nNumSamples+1);j++)
                {
                        m_pMagnitudeResponse[i][j] = new double [m_nNumSamples+1];
                        m_pPhaseResponse[i][j] = new double [m_nNumSamples+1];
                }
        }

        // Allocate memory for frequency axis
        m_pdFreqAxis = new double [m_nNumSamples+1];

        // Allocate memory for impulse axis
        m_pdImpulseAxis = new double [m_nNumSamples/2+m_nOffset];

        // Allocate memory for {a} and {b} coefficients
        m_pACoeffArray = new double **[m_nOrder+1];
        m_pBCoeffArray = new double **[m_nOrder+1];
        for(i=0;i<(m_nOrder+1);i++)
        {
                m_pACoeffArray[i] = new double *[m_nOrder+1];
                m_pBCoeffArray[i] = new double *[m_nOrder+1];
                for(j=0;j<(m_nOrder+1);j++)
                {
                        m_pACoeffArray[i][j] = new double [m_nOrder+1];
                        m_pBCoeffArray[i][j] = new double [m_nOrder+1];
                }
        }

        ComputeCoefficients();
}
else
{
        return FALSE;
```

```
	}

	return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CFilter3DDoc serialization
void CFilter3DDoc::Serialize(CArchive& ar)
{
	unsigned i,j,k;
	if (ar.IsStoring())
	{
		ar << m_nNumSamples << m_dCutoffFreq << m_nOrder << m_nOffset;

		for(i=0;i<(m_nOrder+1);i++)
			for(j=0;j<(m_nOrder+1);j++)
				for(k=0;k<(m_nOrder+1);k++)
					ar << m_pACoeffArray[i][j][k];

		for(i=0;i<(m_nOrder+1);i++)
			for(j=0;j<(m_nOrder+1);j++)
				for(k=0;k<(m_nOrder+1);k++)
					ar << m_pBCoeffArray[i][j][k];

		for(i=0;i<(m_nNumSamples/2+m_nOffset);i++)
			for(j=0;j<(m_nNumSamples/2+m_nOffset);j++)
				for(k=0;k<(m_nNumSamples/2+m_nOffset);k++)
					ar << m_pImpulseResponse[i][j][k];

		for(i=0;i<(m_nNumSamples+1);i++)
			for(j=0;j<(m_nNumSamples+1);j++)
				for(k=0;k<(m_nNumSamples+1);k++)
					ar << m_pMagnitudeResponse[i][j][k];

		for(i=0;i<(m_nNumSamples+1);i++)
			for(j=0;j<(m_nNumSamples+1);j++)
				for(k=0;k<(m_nNumSamples+1);k++)
					ar << m_pPhaseResponse[i][j][k];

		for(i=0;i<(m_nNumSamples+1);i++)
			ar << m_pdFreqAxis[i];

		for(i=0;i<(m_nNumSamples/2+m_nOffset);i++)
			ar << m_pdImpulseAxis[i];
	}
	else
	{
		ar >> m_nNumSamples >> m_dCutoffFreq >> m_nOrder >> m_nOffset;

		// Allocate memory for impulse response
		m_pImpulseResponse = new double **[m_nNumSamples/2 + m_nOffset];
		for(i=0;i<(m_nNumSamples/2+m_nOffset);i++)
		{
			m_pImpulseResponse[i] = new double *[m_nNumSamples/2 + m_nOffset];
			for(j=0;j<(m_nNumSamples/2 + m_nOffset);j++)
				m_pImpulseResponse[i][j] = new double [m_nNumSamples/2 + m_nOffset];
```

```
}

// Allocate memory for magnitude and phase response
m_pMagnitudeResponse = new double **[m_nNumSamples+1];
m_pPhaseResponse = new double **[m_nNumSamples+1];
for(i=0;i<(m_nNumSamples+1);i++)
{
        m_pMagnitudeResponse[i] = new double *[m_nNumSamples+1];
        m_pPhaseResponse[i] = new double *[m_nNumSamples+1];
        for(j=0;j<(m_nNumSamples+1);j++)
        {
                m_pMagnitudeResponse[i][j] = new double [m_nNumSamples+1];
                m_pPhaseResponse[i][j] = new double [m_nNumSamples+1];
        }
}

// Allocate memory for frequency axis
m_pdFreqAxis = new double [m_nNumSamples+1];

// Allocate memory for impulse axis
m_pdImpulseAxis = new double [m_nNumSamples/2+m_nOffset];

// Allocate memory for {a} and {b} coefficients
m_pACoeffArray = new double **[m_nOrder+1];
m_pBCoeffArray = new double **[m_nOrder+1];
for(i=0;i<(m_nOrder+1);i++)
{
        m_pACoeffArray[i] = new double *[m_nOrder+1];
        m_pBCoeffArray[i] = new double *[m_nOrder+1];
        for(j=0;j<(m_nOrder+1);j++)
        {
                m_pACoeffArray[i][j] = new double [m_nOrder+1];
                m_pBCoeffArray[i][j] = new double [m_nOrder+1];
        }
}

for(i=0;i<(m_nOrder+1);i++)
        for(j=0;j<(m_nOrder+1);j++)
                for(k=0;k<(m_nOrder+1);k++)
                        ar >> m_pACoeffArray[i][j][k];

for(i=0;i<(m_nOrder+1);i++)
        for(j=0;j<(m_nOrder+1);j++)
                for(k=0;k<(m_nOrder+1);k++)
                        ar >> m_pBCoeffArray[i][j][k];

for(i=0;i<(m_nNumSamples/2+m_nOffset);i++)
        for(j=0;j<(m_nNumSamples/2+m_nOffset);j++)
                for(k=0;k<(m_nNumSamples/2+m_nOffset);k++)
                        ar >> m_pImpulseResponse[i][j][k];

for(i=0;i<(m_nNumSamples+1);i++)
        for(j=0;j<(m_nNumSamples+1);j++)
                for(k=0;k<(m_nNumSamples+1);k++)
                        ar >> m_pMagnitudeResponse[i][j][k];

for(i=0;i<(m_nNumSamples+1);i++)
```

```
                        for(j=0;j<(m_nNumSamples+1);j++)
                                for(k=0;k<(m_nNumSamples+1);k++)
                                        ar >> m_pPhaseResponse[i][j][k];

                for(i=0;i<(m_nNumSamples+1);i++)
                        ar >> m_pdFreqAxis[i];

                for(i=0;i<(m_nNumSamples/2+m_nOffset);i++)
                        ar >> m_pdImpulseAxis[i];
        }
}


/////////////////////////////////////////////////////////////////////////////
// CFilter3DDoc diagnostics
#ifdef _DEBUG
void CFilter3DDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CFilter3DDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG


/////////////////////////////////////////////////////////////////////////////
// CFilter3DDoc commands
BOOL CFilter3DDoc::Simq(double **matrix, unsigned nEquations)
{
        unsigned i,j,k,l;
        double Big,temp;
        for(j=0;j<nEquations;j++)     // pass #
        {
                // Find Big
                Big = fabs(matrix[j][j]);
                l = j;
                for(i=j+1;i<nEquations;i++)
                {
                        if(Big<fabs(matrix[i][j]))
                        {
                                Big = fabs(matrix[i][j]);
                                l = i;
                        }
                }

                // Check that Big not equal to zero
                if(Big < 1.0e-7)
                {
                        AfxMessageBox("Unable to Solve Set of Equations");
                        return FALSE;
                }

                // Switch Rows
                if(l != j)
```

```
		{
			for(k=0;k<nEquations+1;k++)
			{
				temp = matrix[j][k];
				matrix[j][k] = matrix[l][k];
				matrix[l][k] = temp;
			}
		}

		// Normalization
		for(k=j+1;k<nEquations+1;k++)
			matrix[j][k] /= matrix[j][j];
		matrix[j][j] = 1.0;

		// Elimination
		for(i=0;i<nEquations;i++)
		{
			if(i == j) continue;
			for(k=j+1;k<nEquations+1;k++)
				matrix[i][k] -= matrix[j][k] * matrix[i][j];
			matrix[i][j] = 0.0;
		}
	} // End of Pass
	return TRUE;
}


void CFilter3DDoc::BitReversal(unsigned *L, unsigned N)
{
	// Sub-program developed by M.A. Sid-Ahmed
	// Routine for generating LUT for bit reversal.
	// Note: N=(2 to the power of m).
	// LUT will reside in L[]

	unsigned MASK,C,A,j,k,i,m;

	m = (int)(log10((double)N)/log10(2.0));

	for(k=0;k<N;k++)
	{
		MASK = 1;
		C = 0;
		for(i=0,j=m-1;i<m;i++,j--)
		{
			A=(k&MASK)>>i;
			A<<=j;
			C|=A;
			MASK=MASK<<1;
		}
		L[k] = C;
	}
}


void CFilter3DDoc::FFT3D(COMPLEX ***X, unsigned N, unsigned fft)
{
	unsigned i,j,k;
```

```
COMPLEX temp;

for(i=0;i<N;i++)
{
        for(j=0;j<N;j++)
        {
                FFT1D(X[i][j],N,fft);          // FFT of each row (of each frame)
        }
}

// Take transpose of each frame of X array
for(i=0;i<N;i++)
{
        for(j=0;j<N;j++)
        {
                for(k=0;k<N;k++)
                {
                        if(j==k) break;
                        temp = X[i][j][k];
                        X[i][j][k] = X[i][k][j];
                        X[i][k][j] = temp;
                }
        }
}

for(i=0;i<N;i++)
{
        for(j=0;j<N;j++)
        {
                FFT1D(X[i][j],N,fft);          // FFT of each row (of each frame) after transpose
        }
}

// Take transpose of each 'row' of X matrix
for(j=0;j<N;j++)    // for each row
{
        for(i=0;i<N;i++)
        {
                for(k=0;k<N;k++)
                {
                        if(i==k) break;
                        temp = X[i][j][k];
                        X[i][j][k] = X[k][j][i];
                        X[k][j][i] = temp;
                }
        }
}

for(i=0;i<N;i++)
{
        for(j=0;j<N;j++)
        {
                FFT1D(X[i][j],N,fft);          // FFT of each row (of each frame) after 2nd transpose
        }
}

// Take transpose of each 'row' of X matrix
```

```
		for(j=0;j<N;j++)	// for each row
		{
			for(i=0;i<N;i++)
			{
				for(k=0;k<N;k++)
				{
					if(i==k) break;
					temp = X[i][j][k];
					X[i][j][k] = X[k][j][i];
					X[k][j][i] = temp;
				}
			}
		}

		// Take transpose of each frame of X matrix
		// Take transpose of each frame of X array
		for(i=0;i<N;i++)
		{
			for(j=0;j<N;j++)
			{
				for(k=0;k<N;k++)
				{
					if(j==k) break;
					temp = X[i][j][k];
					X[i][j][k] = X[i][k][j];
					X[i][k][j] = temp;
				}
			}
		}
}

// computes the one-dimensional fft of an array of values
// X[] holds the values of the array
// N is number of values
// fft = 1 --> fft
// fft = 2 --> ifft
// W[] holds the twiddle factors
void CFilter3DDoc::FFT1D(COMPLEX *X, unsigned N, unsigned fft)
{
		unsigned i,j,k;
		unsigned incr,n,ip,group,stage,m;
		unsigned int *L;
		COMPLEX T,*W,*Temp;

		m = (int)(log10((double)N)/log10(2.0));
		incr = 2;  // distance between groups
		n = (int)pow(2,(m-1));
		ip = incr/2;		// distance between butterfly inputs

		// Allocate memory for twiddle factors
		W = new COMPLEX [N/2];

		// Allocate memory for bit-reversed LUT
		L = new unsigned int [N];

		// Allocate memory for temporary array
		Temp = new COMPLEX [N];
```

```
// Generate bit-reversed LUT
BitReversal(L,N);

// Rearrange order in FFT input array
for(i=0;i<N;i++)
        Temp[i] = X[i];
for(i=0;i<N;i++)
        X[L[i]] = Temp[i];
delete Temp;

// Generate twiddle factor LUT
for(i=0;i<N/2;i++)
{
        if(fft == 1)
                W[i] = COMPLEX( cos((2.0*pi/(float)N)*double(i)),
                                -sin((2.0*pi/(float)N)*double(i)) );
        else
                W[i] = COMPLEX( cos((2.0*pi/(float)N)*double(i)),
                                sin((2.0*pi/(float)N)*double(i)) );
}

// Algorithm for first stage with all weights equal to 1
// ---------------------------------------------
for(group=0;group<N;group += incr)
{
        j = group + ip;
        T = X[j];
        X[j] = X[group] - T;
        X[group] = X[group] + T;
}

incr = incr *2;
n = n/2;
ip = incr/2;

// Algorithm for remaining stages with weights not always equal to 1
// ---------------------------------------------------
for(stage=1;stage<m;stage++)            // N = 2 to the power (m)
{
        for(group=0;group<N;group+=incr)
        {
                for(k=0;k<(N/(2*n));k++)
                {
                        j = k+ip;
                        T = X[group + j] * W[n*k];
                        X[group + j] = X[group + k] - T;
                        X[group + k] = X[group + k] + T;
                }
        }
        incr = incr * 2;
        n = n/2;
        ip = incr/2;
}

if( fft == 2 )
        for(i=0;i<N;i++)
```

```
		{
			X[i] = X[i] / double(N);
		}
}


void CFilter3DDoc::ComputeCoefficients()
{
	unsigned i,j,k;
	unsigned k1,k2,k3,N1;
	COMPLEX ***H;

	N1 = m_nNumSamples/2;

	// Convert cutoff frequency to samples number
	m_dCutoffFreq *= double(m_nNumSamples)/(2.0*pi);

	// Allocate memory for Desired Magnitude Response
	H = new COMPLEX **[m_nNumSamples];
	for(i=0;i<m_nNumSamples;i++)
	{
		H[i] = new COMPLEX *[m_nNumSamples];
		for(j=0;j<m_nNumSamples;j++)
			H[i][j] = new COMPLEX [m_nNumSamples];
	}

	// Form Desired Magnitude Response
	for(k1=0;k1<m_nNumSamples;k1++)
	{
		for(k2=0;k2<m_nNumSamples;k2++)
		{
			for(k3=0;k3<m_nNumSamples;k3++)
			{

if((abs(k1-N1)<m_dCutoffFreq)&&(abs(k2-N1)<m_dCutoffFreq)&&(abs(k3-N1)<m_dCutoffFreq))
						H[k1][k2][k3] = COMPLEX(1.0,0.0);
					else
						H[k1][k2][k3] = COMPLEX(0.0,0.0);
			}
		}
	}

	// Apply shift in Frequency Domain
	for(k1=0;k1<m_nNumSamples;k1++)
	{
		for(k2=0;k2<m_nNumSamples;k2++)
		{
			for(k3=0;k3<m_nNumSamples;k3++)
				H[k1][k2][k3] = H[k1][k2][k3] * pow(-1,(k1+k2+k3));
		}
	}

	FFT3D(H,m_nNumSamples,2);		// 3-D IFFT of Desired Magnitude Response

	// Apply shift in Time Domain
	for(k1=0;k1<m_nNumSamples;k1++)
	{
```

```
		for(k2=0;k2<m_nNumSamples;k2++)
		{
			for(k3=0;k3<m_nNumSamples;k3++)
				H[k1][k2][k3] = H[k1][k2][k3] * pow(-1,(k1+k2+k3));
		}
	}

	// ---------------------------------------------------
	// Shank's method begins here
	//
	unsigned x,y,z,n1,n2,n3,M,M1;
	double **A;

	M = m_nNumSamples/2 + m_nOffset;
	M1 = m_nNumSamples/2 - m_nOffset;

	// Trim impulse response
	for(i=M1;i<m_nNumSamples;i++)
	{
		for(j=M1;j<m_nNumSamples;j++)
		{
			for(k=M1;k<m_nNumSamples;k++)
				m_pImpulseResponse[i-M1][j-M1][k-M1] = H[i][j][k].GetReal();
		}
	}

	// Set values of Impulse Axis
	for(i=0;i<(m_nNumSamples/2+m_nOffset);i++)
		m_pdImpulseAxis[i] = double(i);

#ifdef _DEBUG
	// ********************************************
	// Write impulse response to file for debugging
	fstream impulse("Impulse.dbg",ios::out);
	for(n1=0;n1<M;n1++)
	{
		for(n2=0;n2<M;n2++)
		{
			for(n3=0;n3<M;n3++)
				impulse << m_pImpulseResponse[n1][n2][n3] << "\t";
			impulse << endl;
		}
		impulse << endl << endl;
	}
	impulse.close();
	// ********************************************
#endif

	// De-allocate memory for Desired Magnitude Response
	for(i=0;i<m_nNumSamples;i++)
	{
		for(j=0;j<m_nNumSamples;j++)
			delete [] H[i][j];
		delete [] H[i];
	}
	delete [] H;
```

```
// Allocate memory for A matrix
A = new double *[(m_nOrder+1)*(m_nOrder+1)*(m_nOrder+1)-1];
for(i=0;i<((m_nOrder+1)*(m_nOrder+1)*(m_nOrder+1)-1);i++)
        A[i] = new double [(m_nOrder+1)*(m_nOrder+1)*(m_nOrder+1)];

// Forming the A matrix
unsigned row,col;
row = 0;
for(x=0;x<=m_nOrder;x++)
{
        for(y=0;y<=m_nOrder;y++)
        {
                for(z=0;z<=m_nOrder;z++)
                {
                        if((x+y+z)==0) continue;
                        col = 0;
                        for(i=0;i<=m_nOrder;i++)
                        {
                                for(j=0;j<=m_nOrder;j++)
                                {
                                        for(k=0;k<=m_nOrder;k++)
                                        {
                                                if((i+j+k)==0) continue;
                                                A[row][col] = 0.0;
                                                for(n1=(m_nOrder+1);n1<M;n1++)
                                                {
                                                        for(n2=(m_nOrder+1);n2<M;n2++)
                                                        {
                                                                for(n3=(m_nOrder+1);n3<M;n3++)
                                                                        A[row][col] +=
m_pImpulseResponse[n1-i][n2-j][n3-k] *

        m_pImpulseResponse[n1-x][n2-y][n3-z];
                                                        }
                                                }
                                                col++;
                                        }
                                }
                        }
                        row++;
                }
        }
}

row=0;
for(x=0;x<=m_nOrder;x++)
{
        for(y=0;y<=m_nOrder;y++)
        {
                for(z=0;z<=m_nOrder;z++)
                {
                        if((x+y+z)==0) continue;
                        A[row][((m_nOrder+1)*(m_nOrder+1)*(m_nOrder+1)-1)] = 0.0;
                        for(n1=(m_nOrder+1);n1<M;n1++)
                        {
                                for(n2=(m_nOrder+1);n2<M;n2++)
                                {
```

```
                                                for(n3=(m_nOrder+1);n3<M;n3++)
                                                        A[row][((m_nOrder+1)*(m_nOrder+1)*(m_nOrder+1)-1)]
-=
                                                                m_pImpulseResponse[n1][n2][n3] *
m_pImpulseResponse[n1-x][n2-y][n3-z];
                                        }
                                }
                                row++;
                        }
                }
        }

        Simq(A,(m_nOrder+1)*(m_nOrder+1)*(m_nOrder+1)-1);

        row=0:
        m_pBCoeffArray[0][0][0] = 1.0;
        for(i=0;i<=m_nOrder;i++)
        {
                for(j=0;j<=m_nOrder;j++)
                {
                        for(k=0;k<=m_nOrder;k++)
                        {
                                if((i+j+k)==0) continue;
                                m_pBCoeffArray[i][j][k] = A[row][(m_nOrder+1)*(m_nOrder+1)*(m_nOrder+1)-1];
                                row++;
                        }
                }
        }

        for(n1=0;n1<=m_nOrder;n1++)
        {
                for(n2=0;n2<=m_nOrder;n2++)
                {
                        for(n3=0;n3<=m_nOrder;n3++)
                        {
                                m_pACoeffArray[n1][n2][n3] = 0.0;
                                for(i=0;i<=m_nOrder;i++)
                                {
                                        if(int(n1-i)<0) continue;
                                        for(j=0;j<=m_nOrder;j++)
                                        {
                                                if(int(n2-j)<0) continue;
                                                for(k=0;k<=m_nOrder;k++)
                                                {
                                                        if(int(n3-k)<0) continue;
                                                        m_pACoeffArray[n1][n2][n3] += m_pBCoeffArray[i][j][k]
*
                                                                m_pImpulseResponse[n1-i][n2-j][n3-k];
                                                }
                                        }
                                }
                        }
                }
        }

        // De-allocate memory for A matrix
        for(i=0;i<((m_nOrder+1)*(m_nOrder+1)*(m_nOrder+1)-1);i++)
```

```
                delete [] A[i];
        delete [] A;

#ifdef _DEBUG
        // ********************************************
        // Write coefficients to file for debugging
        fstream coeff("Coefficients.dbg",ios::out);
        for(i=0;i<=m_nOrder;i++)
        {
                for(j=0;j<=m_nOrder;j++)
                {
                        for(k=0;k<=m_nOrder;k++)
                                coeff << m_pBCoeffArray[i][j][k] << "\t";
                        coeff << endl;
                }
                coeff << endl << endl;
        }

        coeff << endl << endl;
        for(i=0;i<=m_nOrder;i++)
        {
                for(j=0;j<=m_nOrder;j++)
                {
                        for(k=0;k<=m_nOrder;k++)
                                coeff << m_pACoeffArray[i][j][k] << "\t";
                        coeff << endl;
                }
                coeff << endl << endl;
        }
        coeff.close();
        // ********************************************
#endif

        // Computing Magnitude and Phase Response
        double dW;
        COMPLEX num,den;

        // Compute frequency arrays
        m_pdFreqAxis[0] = -pi;
        dW = 2.0*pi/double(m_nNumSamples);
        for(i=1;i<(m_nNumSamples+1);i++)
                m_pdFreqAxis[i] = m_pdFreqAxis[i-1] + dW;

        for(n1=0;n1<(m_nNumSamples+1);n1++)
        {
                for(n2=0;n2<(m_nNumSamples+1);n2++)
                {
                        for(n3=0;n3<(m_nNumSamples+1);n3++)
                        {
                                num = COMPLEX(0.0,0.0);
                                den = COMPLEX(0.0,0.0);
                                for(i=0;i<=m_nOrder;i++)
                                {
                                        for(j=0;j<=m_nOrder;j++)
                                        {
                                                for(k=0;k<=m_nOrder;k++)
                                                {
```

```
                                                    num = num + COMPLEX(m_pACoeffArray[i][j][k],0.0) *
COMPLEX(cos(i*m_pdFreqAxis[n1]+j*m_pdFreqAxis[n2]+k*m_pdFreqAxis[n3]),-sin(i*m_pdFreqAxis[n1]+j*m_pdFreqAxi
s[n2]+k*m_pdFreqAxis[n3]));
                                                    den = den + COMPLEX(m_pBCoeffArray[i][j][k],0.0) *
COMPLEX(cos(i*m_pdFreqAxis[n1]+j*m_pdFreqAxis[n2]+k*m_pdFreqAxis[n3]),-sin(i*m_pdFreqAxis[n1]+j*m_pdFreqAxi
s[n2]+k*m_pdFreqAxis[n3]));
                                                }
                                            }
                                        }
                                        m_pMagnitudeResponse[n1][n2][n3] = num.Magnitude()/den.Magnitude();
                                        m_pPhaseResponse[n1][n2][n3] = num.Phase() - den.Phase();
                                    }
                                }
                            }

#ifdef _DEBUG
        // ********************************************
        // Write magnitude and phase response to file for debugging
        fstream magnitude("Magnitude.dbg",ios::out);
        fstream phase("Phase.dbg",ios::out);

        for(n1=0;n1<(m_nNumSamples+1);n1++)
        {
                magnitude << m_pdFreqAxis[n1] << "\nH=[";
                phase << m_pdFreqAxis[n1] << "\nP=[";
                for(n2=0;n2<(m_nNumSamples+1);n2++)
                {
                        for(n3=0;n3<(m_nNumSamples+1);n3++)
                        {
                                magnitude << m_pMagnitudeResponse[n1][n2][n3] << "\t";
                                phase << m_pPhaseResponse[n1][n2][n3] << "\t";
                        }
                        magnitude << ';' << endl;
                        phase << ';' << endl;
                }
                magnitude << endl << endl;
                phase << endl << endl;
        }
        magnitude.close();
        phase.close();
        // ********************************************
#endif
}


double*** CFilter3DDoc::GetACoefficients()
{       return m_pACoeffArray;      }


double*** CFilter3DDoc::GetBCoefficients()
{       return m_pBCoeffArray;      }


unsigned CFilter3DDoc::GetNumCoefficients()
{       return (m_nOrder+1);        }
```

```
double*** CFilter3DDoc::GetImpulseResponse()
{           return m_pImpulseResponse;          }

unsigned CFilter3DDoc::GetImpulseResponseSize()
{           return (m_nNumSamples/2 + m_nOffset);          }

double*** CFilter3DDoc::GetMagnitudeResponse()
{           return m_pMagnitudeResponse;          }

unsigned CFilter3DDoc::GetMagnitudeResponseSize()
{           return (m_nNumSamples + 1);          }

double*** CFilter3DDoc::GetPhaseResponse()
{           return m_pPhaseResponse;   }

unsigned CFilter3DDoc::GetPhaseResponseSize()
{           return (m_nNumSamples + 1);          }

double* CFilter3DDoc::GetFreqAxis()
{           return m_pdFreqAxis;          }

double* CFilter3DDoc::GetImpulseAxis()
{           return m_pdImpulseAxis;     }

BOOL CFilter3DDoc::FilterMovie()
{
        unsigned i,j,k,n2,n3,frame;
        AVISTREAMINFO strHdrOld, strHdrNew;
        PAVIFILE pFile=NULL,pFileNew=NULL;
        PAVISTREAM pStream=NULL,pStreamNew=NULL;
        HRESULT hr;       // handle for error checking
        BITMAPINFOHEADER bmiHeader;
        unsigned numFrames;
        IGetFrame* getFrameObj = NULL;
        BYTE *tempFramePtr = NULL;
        BYTE *framePtr = NULL;
        DWORD imageOffset,paletteOffset;
        BYTE *palette = NULL, pixelValue;
        double maxVal,minVal;
        double **aviFrame;

        // Check to ensure version of Video for Windows is up-to-date
        WORD wVer = HIWORD(VideoForWindowsVersion());
        if(wVer < 0x010a)
        {
                AfxMessageBox("Video for Windows version is too old.");
                return FALSE;
        }
```

```
// Initialize AVIFile library
AVIFileInit();

// Open AVI file for reading
hr = AVIFileOpen(&pFile,m_lpstrFileName,OF_READ,NULL);
if(hr!=AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Opening the Input File.");
        return FALSE;
}

// Create new AVI file for writing
hr = AVIFileOpen(&pFileNew,m_lpstrNewFileName,OF_WRITE|OF_CREATE,NULL);
if(hr!=AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Creating the Output File.");
        return FALSE;
}

// Open AVI stream for reading
hr = AVIFileGetStream(pFile,&pStream,streamtypeVIDEO,0);
if(hr!=AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Opening the Input Stream.");
        return FALSE;
}

// Close original AVI file
AVIFileClose(pFile);

// Calculate number of frames in stream
numFrames = AVIStreamEnd(pStream)-AVIStreamStart(pStream);

// Create modeless Processing Dialog Box and display to user
CProcessingDlg dlg;
dlg.m_progressCtrl.SetRange(0,int(2*numFrames));
dlg.m_progressCtrl.SetStep(1);

// Prepare to decompress video frames from stream
getFrameObj = AVIStreamGetFrameOpen(pStream,NULL);

// Obtain address of first decompressed video frame
tempFramePtr = (BYTE *)AVIStreamGetFrame(getFrameObj,0);

// Extract BITMAPINFOHEADER from first decompressed video frame
ExtractBMPHeader(bmiHeader,tempFramePtr);

// Calculate palette offset and image offset
paletteOffset = bmiHeader.biSize;
imageOffset = bmiHeader.biSize+bmiHeader.biClrUsed*sizeof(RGBQUAD);

// Allocate memory for input buffer
inputBuffer = new BYTE **[m_nOrder+1];
for(i=0;i<(m_nOrder+1);i++)
{
        inputBuffer[i] = new BYTE *[(2*bmiHeader.biHeight)];
        for(j=0;j<(unsigned)(2*bmiHeader.biHeight);j++)
```

```
                inputBuffer[i][j] = new BYTE [(2*bmiHeader.biWidth)];
}

// Allocate memory for output buffer
outputBuffer = new double **[m_nOrder+1];
for(i=0;i<(m_nOrder+1);i++)
{
        outputBuffer[i] = new double *[(2*bmiHeader.biHeight)];
        for(j=0;j<(unsigned)(2*bmiHeader.biHeight);j++)
                outputBuffer[i][j] = new double [(2*bmiHeader.biWidth)];
}

// Allocate memory for aviFrame
aviFrame = new double *[(2*bmiHeader.biHeight)];
for(i=0;i<(unsigned)(2*bmiHeader.biHeight);i++)
        aviFrame[i] = new double [(2*bmiHeader.biWidth)];

// Allocate memory for framePtr
framePtr = new BYTE [bmiHeader.biSize+bmiHeader.biClrUsed*sizeof(RGBQUAD)+4*bmiHeader.biSizeImage];

// Clear input and output buffers
for(i=0;i<=m_nOrder;i++)
{
        for(j=0;j<(unsigned)(2*bmiHeader.biHeight);j++)
        {
                for(k=0;k<(unsigned)(2*bmiHeader.biWidth);k++)
                {
                        inputBuffer[i][j][k] = 0;
                        outputBuffer[i][j][k] = 0.0;
                }
        }
}

// Allocate memory for palette and fill it in
palette = new BYTE [bmiHeader.biClrUsed*sizeof(RGBQUAD)];
for(i=0;i<(bmiHeader.biClrUsed*sizeof(RGBQUAD));i++)
        palette[i] = tempFramePtr[paletteOffset+i];

// Get header from old stream
hr = AVIStreamInfo( pStream, &strHdrOld, sizeof(strHdrOld) );
if(hr != AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Reading Old Stream Header.");
        return FALSE;
}

// Fill in the header for the new video stream
memset(&strHdrNew,0,sizeof(strHdrNew));     // Set strHdrNew to zero
strHdrNew.fccType = streamtypeVIDEO;        // stream type
strHdrNew.fccHandler = 0;   // Compressor Code
strHdrNew.dwScale = strHdrOld.dwScale;      // Time Scale
strHdrNew.dwRate = 2*strHdrOld.dwRate;      // Frames per second
strHdrNew.dwLength = 2*strHdrOld.dwLength;  // Number of frames
strHdrNew.dwSuggestedBufferSize = 4*bmiHeader.biSizeImage;    // buffer size
SetRect(&strHdrNew.rcFrame,0,0,2*bmiHeader.biWidth,2*bmiHeader.biHeight); // rectangle for stream

// Create the new stream
```

```
hr = AVIFileCreateStream(pFileNew,&pStreamNew,&strHdrNew);
if(hr != AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Creating the Output Stream.");
        return FALSE;
}

bool zero_pad;
if((bmiHeader.biWidth%2)!=0)zero_pad=true;
else zero_pad=false;

int x=0,y=1,z=2;

// transfer first decompressed frame (BMP) to input buffer (RAW)
for(j=0;j<(unsigned)(2*bmiHeader.biHeight);j++)
{
        for(k=0;k<(unsigned)(2*bmiHeader.biWidth);k++)
        {
                if((j%2)!=0)        // Odd row => Copy pixel from previous row
                        inputBuffer[0][j][k] = inputBuffer[0][j-1][k];

                else if((k%2)!=0)   // Odd pixel => Copy pixel from previous column
                        inputBuffer[0][j][k] = inputBuffer[0][j][k-1];
                else
                {
                        // Even row, Even pixel => Transfer new pixel
                        pixelValue = tempFramePtr[(bmiHeader.biWidth+zero_pad)*j/2 +
                                                                        imageOffset + k/2];
                        inputBuffer[0][j][k] = (unsigned
char)((0.3*(double)palette[pixelValue*sizeof(RGBQUAD)+x]
                                +0.59*(double)palette[pixelValue*sizeof(RGBQUAD)+y]
                                +0.11*(double)palette[pixelValue*sizeof(RGBQUAD)+z])+0.5);
                }
        }
}

        // initialize max and min values for scaling
        maxVal = -10000.0;
        minVal = 10000.0;

        // filter input buffer and store result in output buffer (RAW)
        for(n2=0;n2<(unsigned)(2*bmiHeader.biHeight);n2++)
        {
                for(n3=0;n3<(unsigned)(2*bmiHeader.biWidth);n3++)
                {
                        outputBuffer[0][n2][n3] = 0.0;
                        for(i=0;i<=m_nOrder;i++)
                        {
                                for(j=0;j<=m_nOrder;j++)
                                {
                                        for(k=0;k<=m_nOrder;k++)
                                        {
                                                if((int(n2-j)<0)||(int(n3-k)<0)) continue;
                                                outputBuffer[0][n2][n3] +=
m_pACoeffArray[i][j][k]*inputBuffer[i][n2-j][n3-k];
                                                if((i+j+k)==0) continue;
                                                outputBuffer[0][n2][n3] -=
```

```
m_pBCoeffArray[i][j][k]*outputBuffer[i][n2-j][n3-k];
                                                }
                                        }
                                }
                                // Store max and min values for scaling
                                // Exclude edges from consideration
                                if((n2>=(m_nOrder+1))&&(n2<=2*bmiHeader.biHeight-(m_nOrder+1)))
                                {
                                        if((n3>=(m_nOrder+1))&&(n3<=2*bmiHeader.biWidth-(m_nOrder+1)))
                                        {
                                                maxVal = max(maxVal,outputBuffer[0][n2][n3]);
                                                minVal = min(minVal,outputBuffer[0][n2][n3]);
                                        }
                                }
                                // Fill frame that will be converted back to BMP format
                                aviFrame[n2][n3] = outputBuffer[0][n2][n3];
                        }
                }

        // Scale Frame so values lie between 0 and 255
                for(i=0;i<(unsigned)(2*bmiHeader.biHeight);i++)
                {
                        for(j=0;j<(unsigned)(2*bmiHeader.biWidth);j++)
                        {
                                aviFrame[i][j] = (aviFrame[i][j]-minVal)*255.0 / (maxVal-minVal)+0.5;
                                if(aviFrame[i][j]>255.0) aviFrame[i][j] = 255.0;
                                if(aviFrame[i][j]<0.0) aviFrame[i][j] = 0.0;
                        }
                }

                // Insert header into framePtr
                InsertHeader(bmiHeader,framePtr,tempFramePtr);


                // Create new palette
                InsertPalette(bmiHeader,framePtr,paletteOffset);

                // Insert image data
                for(i=0;i<(unsigned)(2*bmiHeader.biHeight);i++)
                {
                        for(j=0;j<(unsigned)(2*bmiHeader.biWidth);j++)
                                framePtr[bmiHeader.biSize+bmiHeader.biClrUsed*sizeof(RGBQUAD)+
                                        i*(2*bmiHeader.biWidth+zero_pad)+j] = (BYTE)aviFrame[i][j];
                }

                // Set format of new stream
                hr = AVIStreamSetFormat(pStreamNew,0,framePtr,
                                                                bmiHeader.biSize +
                                                                bmiHeader.biClrUsed*sizeof(RGBQUAD));
        if(hr != AVIERR_OK)
        {
                AfxMessageBox("An Error Occurred Setting the Output Stream Format.");
                return FALSE;
        }

        // Write frame to new stream
        hr = AVIStreamWrite(pStreamNew,0,1,
```

```
                                        framePtr + imageOffset,
                                        4*bmiHeader.biSizeImage,
                                        AVIIF_KEYFRAME, NULL, NULL);
if(hr != AVIERR_OK)
{
        AfxMessageBox("An Error Occurred Writing to the Output Stream.");
        return FALSE;
}

// Shift Frames in Input and Output Buffer
for(i=0;i<m_nOrder;i++)
{
        for(j=0;j<(unsigned)(2*bmiHeader.biHeight);j++)
        {
                for(k=0;k<(unsigned)(2*bmiHeader.biWidth);k++)
                {
                        inputBuffer[m_nOrder-i][j][k] = inputBuffer[m_nOrder-1-i][j][k];
                        outputBuffer[m_nOrder-i][j][k] = outputBuffer[m_nOrder-1-i][j][k];
                }
        }
}

// Advance the current position of the progress bar
dlg.m_progressCtrl.StepIt();


// Main Frame filtering loop ------------------------------
for(frame=1;frame<(2*numFrames);frame++)
{
        if((frame%2)!=0)   // Obtain address of first decompressed video frame
                tempFramePtr = (BYTE *)AVIStreamGetFrame(getFrameObj,frame/2);

        // transfer decompressed frame (BMP) to input buffer (RAW)
        for(j=0;j<(unsigned)(2*bmiHeader.biHeight);j++)
        {
                for(k=0;k<(unsigned)(2*bmiHeader.biWidth);k++)
                {
                        if((frame%2)!=0)   // Odd frame => Copy pixel from previous frame
                                inputBuffer[0][j][k] = inputBuffer[1][j][k];

                        else if((j%2)!=0)   // Odd line => Copy pixel from previous line
                                inputBuffer[0][j][k] = inputBuffer[0][j-1][k];

                        else if((k%2)!=0)   // Odd pixel => Copy pixel from previous column
                                inputBuffer[0][j][k] = inputBuffer[0][j][k-1];
                        else
                        {
                                // Even frame, Even line, Even pixel => Transfer new pixel
                                pixelValue = tempFramePtr[bmiHeader.biWidth*j/2 + imageOffset + k/2];
                                inputBuffer[0][j][k] = (unsigned
char)((0.3*(double)palette[pixelValue*sizeof(RGBQUAD)+x]
                                        +0.59*(double)palette[pixelValue*sizeof(RGBQUAD)+y]
                                        +0.11*(double)palette[pixelValue*sizeof(RGBQUAD)+z])+0.5);
                        }
                }
        }
```

```
// filter input buffer and store result in output buffer (RAW)
for(n2=0;n2<(unsigned)(2*bmiHeader.biHeight);n2++)
{
	for(n3=0;n3<(unsigned)(2*bmiHeader.biWidth);n3++)
	{
		outputBuffer[0][n2][n3] = 0.0;
		for(i=0;i<=m_nOrder;i++)
		{
			for(j=0;j<=m_nOrder;j++)
			{
				for(k=0;k<=m_nOrder;k++)
				{
					if((int(n2-j)<0)||(int(n3-k)<0)) continue;
					outputBuffer[0][n2][n3] +=
m_pACoeffArray[i][j][k]*inputBuffer[i][n2-j][n3-k];
					if((i+j+k)==0) continue;
					outputBuffer[0][n2][n3] -=
m_pBCoeffArray[i][j][k]*outputBuffer[i][n2-j][n3-k];
				}
			}
		}
		// Store max and min values for scaling
		// Exclude edges from consideration
		if((n2>=2*(m_nOrder+1))&&(n2<=2*bmiHeader.biHeight-2*(m_nOrder+1)))
		{
			if((n3>=2*(m_nOrder+1))&&(n3<=2*bmiHeader.biWidth-2*(m_nOrder+1)))
			{
				maxVal = max(maxVal,outputBuffer[0][n2][n3]);
				minVal = min(minVal,outputBuffer[0][n2][n3]);
			}
		}
		// Fill frame that will be converted back to BMP format
		aviFrame[n2][n3] = outputBuffer[0][n2][n3];
	}
}

// Scale Frame so values lie between 0 and 255
	for(i=0;i<(unsigned)(2*bmiHeader.biHeight);i++)
	{
		for(j=0;j<(unsigned)(2*bmiHeader.biWidth);j++)
		{
			aviFrame[i][j] = (aviFrame[i][j]-minVal)*255.0 / (maxVal-minVal) + 0.5;
			if(aviFrame[i][j]>255.0) aviFrame[i][j] = 255.0;
			if(aviFrame[i][j]<0.0) aviFrame[i][j] = 0.0;
		}
	}

// Insert header into framePtr
	InsertHeader(bmiHeader,framePtr,tempFramePtr);
	// Create new palette
	InsertPalette(bmiHeader,framePtr,paletteOffset);

	// Insert image data
	for(i=0;i<(unsigned)(2*bmiHeader.biHeight);i++)
	{
		for(j=0;j<(unsigned)(2*bmiHeader.biWidth);j++)
			framePtr[bmiHeader.biSize+bmiHeader.biClrUsed*sizeof(RGBQUAD)+
```

```
                                                        i*(2*bmiHeader.biWidth+zero_pad)+j] =
(BYTE)aviFrame[i][j];
                }

                // Write frame to new stream
                hr = AVIStreamWrite(pStreamNew,frame,1,
                                                        framePtr + imageOffset,
                                                        4*bmiHeader.biSizeImage,
                                                        AVIIF_KEYFRAME, NULL, NULL);
                if(hr != AVIERR_OK)
                {
                        AfxMessageBox("An Error Occurred Writing to the Output Stream.");
                        return FALSE;
                }

                // Shift Frames in Input and Output Buffer
                for(i=0;i<m_nOrder;i++)
                {
                        for(j=0;j<(unsigned)(2*bmiHeader.biHeight);j++)
                        {
                                for(k=0;k<(unsigned)(2*bmiHeader.biWidth);k++)
                                {
                                        inputBuffer[m_nOrder-i][j][k] = inputBuffer[m_nOrder-1-i][j][k];
                                        outputBuffer[m_nOrder-i][j][k] = outputBuffer[m_nOrder-1-i][j][k];
                                }
                        }
                }

                // Advance the current position of the progress bar
                dlg.m_progressCtrl.StepIt();
        }       // End of main Frame filtering loop


        // Close dialog box
        dlg.DestroyWindow();

        // Close the files and streams
        AVIStreamGetFrameClose(getFrameObj);
        AVIStreamClose(pStream);
        AVIStreamClose(pStreamNew);
        AVIFileClose(pFileNew);

        AVIFileExit();
        return TRUE;    // function completed successfully
}

void CFilter3DDoc::ExtractBMPHeader(BITMAPINFOHEADER &bmpHdr, BYTE *tempFramePtr)
{
        // Store BITMAPINFOHEADER information ------------------
        bmpHdr.biSize = tempFramePtr[0x00]+(tempFramePtr[0x01]<<8)+
                                (tempFramePtr[0x02]<<16)+(tempFramePtr[0x03]<<24);
        bmpHdr.biWidth = tempFramePtr[0x04]+(tempFramePtr[0x05]<<8)+
                                (tempFramePtr[0x06]<<16)+(tempFramePtr[0x07]<<24);
        bmpHdr.biHeight = tempFramePtr[0x08]+(tempFramePtr[0x09]<<8)+
                                (tempFramePtr[0x0A]<<16)+(tempFramePtr[0x0B]<<24);
        bmpHdr.biPlanes = 1;
        bmpHdr.biBitCount = tempFramePtr[0x0E]+(tempFramePtr[0x0F]<<8);
```

```
        bmpHdr.biCompression = tempFramePtr[0x10]+(tempFramePtr[0x11]<<8)+
                                        (tempFramePtr[0x12]<<16)+(tempFramePtr[0x13]<<24);
        bmpHdr.biSizeImage = tempFramePtr[0x14]+(tempFramePtr[0x15]<<8)+
                                        (tempFramePtr[0x16]<<16)+(tempFramePtr[0x17]<<24);
        bmpHdr.biXPelsPerMeter = tempFramePtr[0x18]+(tempFramePtr[0x19]<<8)+
                                        (tempFramePtr[0x1A]<<16)+(tempFramePtr[0x1B]<<24);
        bmpHdr.biYPelsPerMeter = tempFramePtr[0x1C]+(tempFramePtr[0x1D]<<8)+
                                        (tempFramePtr[0x1E]<<16)+(tempFramePtr[0x1F]<<24);
        bmpHdr.biClrUsed = tempFramePtr[0x20]+(tempFramePtr[0x21]<<8)+
                                        (tempFramePtr[0x22]<<16)+(tempFramePtr[0x23]<<24);
        bmpHdr.biClrImportant = tempFramePtr[0x24]+(tempFramePtr[0x25]<<8)+
                                        (tempFramePtr[0x26]<<16)+(tempFramePtr[0x27]<<24);
        // Done Storing BITMAPINFOHEADER info ----------------
}


BOOL CFilter3DDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
        if (!CDocument::OnOpenDocument(lpszPathName))
                return FALSE;

        return TRUE;
}


void CFilter3DDoc::InsertHeader(BITMAPINFOHEADER bmiHeader,
                                                        BYTE *framePtr, BYTE *tempFramePtr)
{
        unsigned i;
        for(i=0;i<bmiHeader.biSize;i++)
        {
                framePtr[i] = tempFramePtr[i];          // Copy Header
                framePtr[0x04] = (BYTE)(2*bmiHeader.biWidth);           // double Width
                framePtr[0x05] = (BYTE)((2*bmiHeader.biWidth)>>8);
                framePtr[0x06] = (BYTE)((2*bmiHeader.biWidth)>>16);
                framePtr[0x07] = (BYTE)((2*bmiHeader.biWidth)>>24);

                framePtr[0x08] = (BYTE)(2*bmiHeader.biHeight);          // double Height
                framePtr[0x09] = (BYTE)((2*bmiHeader.biHeight)>>8);
                framePtr[0x0A] = (BYTE)((2*bmiHeader.biHeight)>>16);
                framePtr[0x0B] = (BYTE)((2*bmiHeader.biHeight)>>24);

                framePtr[0x14] = (BYTE)(4*bmiHeader.biSizeImage);       // change size
                framePtr[0x15] = (BYTE)((4*bmiHeader.biSizeImage)>>8);
                framePtr[0x16] = (BYTE)((4*bmiHeader.biSizeImage)>>16);
                framePtr[0x17] = (BYTE)((4*bmiHeader.biSizeImage)>>24);
        }
}


void CFilter3DDoc::InsertPalette(BITMAPINFOHEADER bmiHeader,
                                                        BYTE *framePtr,DWORD paletteOffset )
{
        unsigned i;
        for(i=0;i<bmiHeader.biClrUsed;i++)
        {
                framePtr[paletteOffset+4*i] = (BYTE)i;
```

```
                    framePtr[paletteOffset+4*i+1] = (BYTE)i;
                    framePtr[paletteOffset+4*i+2] = (BYTE)i;
                    framePtr[paletteOffset+4*i+3] = (BYTE)0;
          }
}


void CFilter3DDoc::SetOpenFileName(LPTSTR lpstrFile)
{
          m_lpstrFileName = lpstrFile;
}


void CFilter3DDoc::SetSaveFileName(LPTSTR lpstrFile)
{
          m_lpstrNewFileName = lpstrFile;
}
```

```
// Filter3dSettingsDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Filter3D.h"
#include "Filter3dSettingsDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFilter3DSettingsDlg dialog


CFilter3DSettingsDlg::CFilter3DSettingsDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CFilter3DSettingsDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CFilter3DSettingsDlg)
	m_dCutoffFreq = 0.0;
	m_nOffset = 0;
	m_nOrder = 0;
	m_nNumSamples = 0;
	//}}AFX_DATA_INIT
}


void CFilter3DSettingsDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CFilter3DSettingsDlg)
	DDX_Text(pDX, IDC_CUTOFF, m_dCutoffFreq);
	DDX_Text(pDX, IDC_OFFSET, m_nOffset);
	DDX_Text(pDX, IDC_ORDER, m_nOrder);
	DDX_Text(pDX, IDC_SAMPLES, m_nNumSamples);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CFilter3DSettingsDlg, CDialog)
	//{{AFX_MSG_MAP(CFilter3DSettingsDlg)
		// NOTE: the ClassWizard will add message map macros here
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFilter3DSettingsDlg message handlers
```

```
// Filter3DView.cpp : implementation of the CFilter3DView class
//

#include "stdafx.h"
#include "Filter3D.h"

#include "Filter3DDoc.h"
#include "Filter3DView.h"
#include "vfw.h"
#include <fstream.h>          // For debugging purposes
//#include <commdlg.h>        // For open and save dialog boxes

#define   IMPULSE                    0
#define   MAGNITUDE          1
#define   PHASE              2
#define COEFFICIENTS    3
#define VIDEO                4

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFilter3DView

IMPLEMENT_DYNCREATE(CFilter3DView, CView)

BEGIN_MESSAGE_MAP(CFilter3DView, CView)
        //{{AFX_MSG_MAP(CFilter3DView)
        ON_COMMAND(ID_DISPLAY_COEFFICIENTS, OnDisplayCoefficients)
        ON_COMMAND(ID_DISPLAY_IMPULSE, OnDisplayImpulse)
        ON_COMMAND(ID_DISPLAY_MAGNITUDE, OnDisplayMagnitude)
        ON_COMMAND(ID_DISPLAY_PHASE, OnDisplayPhase)
        ON_COMMAND(ID_VIDEO_OPEN, OnVideoOpen)
        ON_COMMAND(ID_VIDEO_PLAY, OnVideoPlay)
        ON_COMMAND(ID_VIDEO_FILTER, OnVideoFilter)
        ON_COMMAND(ID_VIDEO_OPENOUTPUT, OnVideoOpenOutput)
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFilter3DView construction/destruction

CFilter3DView::CFilter3DView()
{
        pi = 4.0*atan(1.0);
        m_nNumDataPoints = 0;
        m_nDisplay = COEFFICIENTS;

        m_dRotationX = 0.0;
        m_dRotationY = -38.0*(pi/180);
```

```
	m_dRotationZ = 0.0;

	m_hwndOriginalAVI = NULL;
	m_hwndFilteredAVI = NULL;
}

CFilter3DView::~CFilter3DView()
{
}

BOOL CFilter3DView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CFilter3DView drawing

void CFilter3DView::OnDraw(CDC* pDC)
{
	CFilter3DDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	switch(m_nDisplay)
	{
	case COEFFICIENTS:
		DrawCoefficients(pDC);
		break;

	case IMPULSE:
	case MAGNITUDE:
	case PHASE:
		PlotPoints(pDC);
		break;

	case VIDEO:

		break;
	}
}

/////////////////////////////////////////////////////////////////////////////
// CFilter3DView printing

BOOL CFilter3DView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}


void CFilter3DView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
```

```
}

void CFilter3DView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CFilter3DView diagnostics

#ifdef _DEBUG
void CFilter3DView::AssertValid() const
{
	CView::AssertValid();
}


void CFilter3DView::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}


CFilter3DDoc* CFilter3DView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CFilter3DDoc)));
	return (CFilter3DDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CFilter3DView message handlers

void CFilter3DView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
	CFilter3DDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	unsigned i,j,k;
	switch(m_nDisplay)
	{
	case VIDEO:
		break;

	case IMPULSE:
		// If MCI windows exist, remove them
		if(m_hwndOriginalAVI!=NULL)
		{
			MCIWndDestroy(m_hwndOriginalAVI);
			m_hwndOriginalAVI = NULL;

			if(m_hwndFilteredAVI!=NULL)
			{
				MCIWndDestroy(m_hwndFilteredAVI);
				m_hwndFilteredAVI = NULL;
			}
```

```
		}

		if(m_nNumDataPoints)		// if array already exists, de-allocate
		{
			for(i=0;i<m_nNumDataPoints;i++)
				delete [] m_pDisplayPointArray[i];
			delete [] m_pDisplayPointArray;
		}
		m_pDataPointArray = pDoc->GetImpulseResponse();
		m_nNumDataPoints = pDoc->GetImpulseResponseSize();

		// Allocate display point memory
		m_pDisplayPointArray = new CPoint *[m_nNumDataPoints];
		for(i=0;i<m_nNumDataPoints;i++)
			m_pDisplayPointArray[i] = new CPoint [m_nNumDataPoints];


		m_pdAxisX = pDoc->GetImpulseAxis();
		m_pdAxisZ = pDoc->GetImpulseAxis();
		for(j=0;j<m_nNumDataPoints;j++)
		{
			for(k=0;k<m_nNumDataPoints;k++)
				m_pDisplayPointArray[j][k] = ConvertToScreen(m_pdAxisX[k],m_pDataPointArray[5
/*(unsigned)m_dConstAxisFreq */][j][k],m_pdAxisZ[j]);
		}
		MakePlotFitWindow();
		break;

	case MAGNITUDE:
		// If MCI windows exist, remove them
		if(m_hwndOriginalAVI!=NULL)
		{
			MCIWndDestroy(m_hwndOriginalAVI);
			m_hwndOriginalAVI = NULL;

			if(m_hwndFilteredAVI!=NULL)
			{
				MCIWndDestroy(m_hwndFilteredAVI);
				m_hwndFilteredAVI = NULL;
			}
		}

		if(m_nNumDataPoints)		// if array already exists, de-allocate
		{
			for(i=0;i<m_nNumDataPoints;i++)
				delete [] m_pDisplayPointArray[i];
			delete [] m_pDisplayPointArray;
		}
		m_pDataPointArray = pDoc->GetMagnitudeResponse();
		m_nNumDataPoints = pDoc->GetMagnitudeResponseSize();
		m_pDisplayPointArray = new CPoint *[m_nNumDataPoints];
		for(i=0;i<m_nNumDataPoints;i++)
			m_pDisplayPointArray[i] = new CPoint [m_nNumDataPoints];

		m_pdAxisX = pDoc->GetFreqAxis();
		m_pdAxisZ = pDoc->GetFreqAxis();
		for(j=0;j<m_nNumDataPoints;j++)
```

```
		{
			for(k=0;k<m_nNumDataPoints;k++)
				m_pDisplayPointArray[j][k] = ConvertToScreen(m_pdAxisX[k],m_pDataPointArray[8
/*(unsigned)m_dConstAxisFreq */][j][k],m_pdAxisZ[j]);
		}
		MakePlotFitWindow();
		break;

	case PHASE:
		// If MCI windows exist, remove them
		if(m_hwndOriginalAVI!=NULL)
		{
			MCIWndDestroy(m_hwndOriginalAVI);
			m_hwndOriginalAVI = NULL;

			if(m_hwndFilteredAVI!=NULL)
			{
				MCIWndDestroy(m_hwndFilteredAVI);
				m_hwndFilteredAVI = NULL;
			}
		}

		if(m_nNumDataPoints)	// if array already exists, de-allocate
		{
			for(i=0;i<m_nNumDataPoints;i++)
				delete [] m_pDisplayPointArray[i];
			delete [] m_pDisplayPointArray;
		}
		m_pDataPointArray = pDoc->GetPhaseResponse();
		m_nNumDataPoints = pDoc->GetPhaseResponseSize();
		m_pDisplayPointArray = new CPoint *[m_nNumDataPoints];
		for(i=0;i<m_nNumDataPoints;i++)
			m_pDisplayPointArray[i] = new CPoint [m_nNumDataPoints];

		m_pdAxisX = pDoc->GetFreqAxis();
		m_pdAxisZ = pDoc->GetFreqAxis();
		for(j=0;j<m_nNumDataPoints;j++)
		{
			for(k=0;k<m_nNumDataPoints;k++)
				m_pDisplayPointArray[j][k] = ConvertToScreen(m_pdAxisX[k],m_pDataPointArray[8
/*(unsigned)m_dConstAxisFreq */][j][k],m_pdAxisZ[j]);
		}
		MakePlotFitWindow();
		break;

	case COEFFICIENTS:
		// If MCI windows exist, remove them
		if(m_hwndOriginalAVI!=NULL)
		{
			MCIWndDestroy(m_hwndOriginalAVI);
			m_hwndOriginalAVI = NULL;

			if(m_hwndFilteredAVI!=NULL)
			{
				MCIWndDestroy(m_hwndFilteredAVI);
				m_hwndFilteredAVI = NULL;
			}
```

```
                }

                m_pACoeffArray = pDoc->GetACoefficients();
                m_pBCoeffArray = pDoc->GetBCoefficients();
                m_nNumCoefficients = pDoc->GetNumCoefficients();
                break;
        }
        RedrawWindow();
}


void CFilter3DView::DrawCoefficients(CDC *pDC)
{
        CFont fontCur;
        if(fontCur.CreatePointFont(100, "Roman", pDC))
        {
                CFont* pOldFont = pDC->SelectObject(&fontCur);

                char ch_buffer[10];
                unsigned i,j,k;
                CRect lRect;
                CString A_coeff = "{a} Coefficients:\n";
                CString B_coeff = "{b} Coefficients:\n";

                GetClientRect(lRect);
                lRect.right /= 2;

                for(i=0;i<m_nNumCoefficients;i++)
                {
                        for(j=0;j<m_nNumCoefficients;j++)
                        {
                                for(k=0;k<m_nNumCoefficients;k++)
                                {
                                        sprintf(ch_buffer,"%.5f",m_pACoeffArray[i][j][k]);
                                        A_coeff += ch_buffer;
                                        A_coeff += "        ";
                                }
                                A_coeff += "\n";
                        }
                        A_coeff += "\n\n";
                }
                pDC->DrawText(A_coeff, lRect, DT_CENTER);

                for(i=0;i<m_nNumCoefficients;i++)
                {
                        for(j=0;j<m_nNumCoefficients;j++)
                        {
                                for(k=0;k<m_nNumCoefficients;k++)
                                {
                                        sprintf(ch_buffer,"%.5f",m_pBCoeffArray[i][j][k]);
                                        B_coeff += ch_buffer;
                                        B_coeff += "        ";
                                }
                                B_coeff += "\n";
                        }
                        B_coeff += "\n\n";
                }
```

```
                lRect.left = lRect.right;
                lRect.right *= 2;

                pDC->DrawText(B_coeff, lRect, DT_CENTER);

                pDC->SelectObject(pOldFont);
        }
        fontCur.DeleteObject();
}


void CFilter3DView::PlotPoints(CDC *pDC)
{
        int i,j;
        CPoint points[4];
        for(i=m_nNumDataPoints-1;i>0;i--)
        {
                for(j=0;(unsigned)j<m_nNumDataPoints-1;j++)
                {
                        points[0] = CPoint(long(m_pDisplayPointArray[i][j].x),long(m_pDisplayPointArray[i][j].y));
                        points[1] = CPoint(long(m_pDisplayPointArray[i][j+1].x),long(m_pDisplayPointArray[i][j+1].y));
                        points[2] =
CPoint(long(m_pDisplayPointArray[i-1][j+1].x),long(m_pDisplayPointArray[i-1][j+1].y));
                        points[3] = CPoint(long(m_pDisplayPointArray[i-1][j].x),long(m_pDisplayPointArray[i-1][j].y));

                        pDC->Polygon(points,4);
                }
        }
}


void CFilter3DView::OnDisplayCoefficients()
{       m_nDisplay = COEFFICIENTS;
        OnUpdate(NULL, 0L, NULL);       }


void CFilter3DView::OnDisplayImpulse()
{       m_nDisplay = IMPULSE;
        OnUpdate(NULL, 0L, NULL);               }


void CFilter3DView::OnDisplayMagnitude()
{       m_nDisplay = MAGNITUDE;
        OnUpdate(NULL, 0L, NULL);       }


void CFilter3DView::OnDisplayPhase()
{       m_nDisplay = PHASE;
        OnUpdate(NULL, 0L, NULL);       }


POINT CFilter3DView::ConvertToScreen(double x, double y, double z)
{
        POINT point;
        float xp = 0, yp = 0.5, zp = 1;         // Perspective vector

        TransformPoints(x,y,z);
```

```
	x *= 1000;
	z *= 1000;
	switch(m_nDisplay)
	{
	case IMPULSE:
		y *= 7000000;
		break;
	case MAGNITUDE:
		y *= 300000;
		break;
	case PHASE:
		y *= 1000;
		break;
	}

	point.x = long((4.0/3.0)*(x + z*(xp/zp)));
	point.y = long(-(y+z*(yp/zp)));
	return point;
}


void CFilter3DView::MakePlotFitWindow()
{
	// Scale and center the plot so it fits the window
	// with a margin on all sides
	unsigned i,j;
	int max_x = m_pDisplayPointArray[0][0].x;
	int min_x = m_pDisplayPointArray[0][0].x;
	int max_y = m_pDisplayPointArray[0][0].y;
	int min_y = m_pDisplayPointArray[0][0].y;
	int margin = 50;
	CRect lpRect;

	// Get max and min values of points
	for(i=0;i<m_nNumDataPoints;i++)
	{
		for(j=0;j<m_nNumDataPoints;j++)
		{
			max_x = max(max_x,m_pDisplayPointArray[i][j].x);
			max_y = max(max_y,m_pDisplayPointArray[i][j].y);
			min_x = min(min_x,m_pDisplayPointArray[i][j].x);
			min_y = min(min_y,m_pDisplayPointArray[i][j].y);
		}
	}

	GetClientRect( lpRect );
	if( (lpRect.right < (2*margin+50)) || (lpRect.bottom < (2*margin+50)) )
		lpRect = CRect(0,0,(2*margin+50),(2*margin+50));

	lpRect.DeflateRect(margin,margin);

	double xScale = double(lpRect.right-lpRect.left)/double(max_x-min_x);
	double yScale = double(lpRect.bottom-lpRect.top)/double(max_y-min_y);

	for(i=0;i<m_nNumDataPoints;i++)
	{
		for(j=0;j<m_nNumDataPoints;j++)
```

```
		{
			m_pDisplayPointArray[i][j].x = long((double)m_pDisplayPointArray[i][j].x * xScale);
			m_pDisplayPointArray[i][j].y = long((double)m_pDisplayPointArray[i][j].y * yScale);
		}
	}

	// adjust minimums and maximums to reflect scaling effects
	min_x = long(min_x*xScale);
	max_x = long(max_x*xScale);
	min_y = long(min_y*yScale);
	max_y = long(max_y*yScale);

	// Center plot in window
	int dx = lpRect.left - min_x;
	int dy = lpRect.bottom - max_y;

	for(i=0;i<m_nNumDataPoints;i++)
	{
		for(j=0;j<m_nNumDataPoints;j++)
		{
			m_pDisplayPointArray[i][j].Offset(dx.dy);
		}
	}
}


void CFilter3DView::TransformPoints(double &x, double &y, double &z)
{
	double xtemp = x;
	double ytemp = y;
	double ztemp = z;

	// Rotation about x-axis
	y = float(ytemp*cos(m_dRotationX) - ztemp*sin(m_dRotationX));
	z = float(ytemp*sin(m_dRotationX) + ztemp*cos(m_dRotationX));

	xtemp = x;
	ytemp = y;
	ztemp = z;

	// Rotation about y-axis
	x = float(xtemp*cos(m_dRotationY) + ztemp*sin(m_dRotationY));
	z = float(-xtemp*sin(m_dRotationY) + ztemp*cos(m_dRotationY));

	xtemp = x;
	ytemp = y;
	ztemp = z;

	// Rotation about z-axis
	x = float(xtemp*cos(m_dRotationZ) - ytemp*sin(m_dRotationZ));
	y = float(xtemp*sin(m_dRotationZ) + ytemp*cos(m_dRotationZ));
}


void CFilter3DView::OnVideoOpen()
{
	m_nDisplay = VIDEO;
```

```
	OnUpdate(NULL, 0L, NULL);

	CFilter3DDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	char buffer[256];
	buffer[0] = NULL;

	OPENFILENAME opfn;
	opfn.lStructSize = sizeof(OPENFILENAME);
	opfn.hwndOwner = m_hWnd;
	opfn.hInstance = 0;
	opfn.lpstrFilter = NULL;
	opfn.lpstrCustomFilter = NULL;
	opfn.nMaxCustFilter = 0;
	opfn.nFilterIndex = 0;
	opfn.lpstrFile = buffer;	// File name to open
	opfn.nMaxFile = 256;
	opfn.lpstrFileTitle = NULL;
	opfn.nMaxFileTitle = 0;
	opfn.lpstrInitialDir = NULL;
	opfn.lpstrTitle = NULL;
	opfn.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY |
				OFN_NONETWORKBUTTON | OFN_PATHMUSTEXIST;
	opfn.nFileOffset = 0;
	opfn.nFileExtension = 0;
	opfn.lpstrDefExt = NULL;
	opfn.lCustData = 0;
	opfn.lpfnHook = 0;
	opfn.lpTemplateName = 0;

	BOOL error = GetOpenFileName(&opfn);
	if(*opfn.lpstrFile==NULL)	// // If the user did not specify a file name
		return;

	if(m_hwndOriginalAVI!=NULL)
		{
			MCIWndDestroy(m_hwndOriginalAVI);
			m_hwndOriginalAVI = NULL;
		}

	pDoc->SetOpenFileName(opfn.lpstrFile);		// Set file to open

	if(*opfn.lpstrFile!=NULL)	// If the user specified a file name
	{
		if((m_hwndOriginalAVI==NULL))
			// If no MCI window exists create it and open file
			m_hwndOriginalAVI = MCIWndCreate(m_hWnd,AfxGetInstanceHandle(),
				MCIWNDF_SHOWNAME |
				MCIWNDF_SHOWMODE |
				WS_VISIBLE		|
				WS_BORDER		|
				WS_CHILD,
				opfn.lpstrFile);
	}
}
```

```
void CFilter3DView::OnVideoPlay()
{
	MCIWndPlay(m_hwndOriginalAVI);
	MCIWndPlay(m_hwndFilteredAVI);
}


void CFilter3DView::OnVideoFilter()
{
	CFilter3DDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	char buffer[256], dlgTitle[] = "Save Filtered Movie As";
	buffer[0] = NULL;

	OPENFILENAME opfn;
	opfn.lStructSize = sizeof(OPENFILENAME);
	opfn.hwndOwner = m_hWnd;
	opfn.hInstance = 0;
	opfn.lpstrFilter = NULL;
	opfn.lpstrCustomFilter = NULL;
	opfn.nMaxCustFilter = 0;
	opfn.nFilterIndex = 0;
	opfn.lpstrFile = buffer;	// File name to open
	opfn.nMaxFile = 256;
	opfn.lpstrFileTitle = NULL;
	opfn.nMaxFileTitle = 0;
	opfn.lpstrInitialDir = NULL;
	opfn.lpstrTitle = dlgTitle;
	opfn.Flags =  OFN_HIDEREADONLY | OFN_NONETWORKBUTTON;
	opfn.nFileOffset = 0;
	opfn.nFileExtension = 0;
	opfn.lpstrDefExt = NULL;
	opfn.lCustData = 0;
	opfn.lpfnHook = 0;
	opfn.lpTemplateName = 0;

	BOOL error = GetSaveFileName(&opfn);

	pDoc->SetSaveFileName(opfn.lpstrFile);		// Set filename to save as

	if(*opfn.lpstrFile==NULL)	// If the user did not specify a file name
	{
		RedrawWindow();	// Redraw MCIWindow toolbar
		return;
	}

	pDoc->FilterMovie();		// Filter AVI file

	if(m_hwndFilteredAVI!=NULL)
	{
		MCIWndDestroy(m_hwndOriginalAVI);
		m_hwndFilteredAVI = NULL;
	}

	m_hwndFilteredAVI = MCIWndCreate(m_hWnd,AfxGetInstanceHandle(),
		MCIWNDF_SHOWNAME |
```

```
                MCIWNDF_SHOWMODE |
                WS_VISIBLE                      |
                WS_BORDER                       |
                WS_CHILD,
                opfn.lpstrFile);

        RECT rCurrent;
        ::GetWindowRect(m_hwndFilteredAVI,&rCurrent);

        ::MoveWindow(m_hwndFilteredAVI,rCurrent.right,0/*rCurrent.top*/,
                                (rCurrent.right-rCurrent.left), (rCurrent.bottom-rCurrent.top), TRUE);

}


void CFilter3DView::OnVideoOpenOutput()
{
        char buffer[256],dlgTitle[] = "Open Filtered Movie";
        buffer[0] = NULL;

        OPENFILENAME opfn;
        opfn.lStructSize = sizeof(OPENFILENAME);
        opfn.hwndOwner = m_hWnd;
        opfn.hInstance = 0;
        opfn.lpstrFilter = NULL;
        opfn.lpstrCustomFilter = NULL;
        opfn.nMaxCustFilter = 0;
        opfn.nFilterIndex = 0;
        opfn.lpstrFile = buffer;        // File name to open
        opfn.nMaxFile = 256;
        opfn.lpstrFileTitle = NULL;
        opfn.nMaxFileTitle = 0;
        opfn.lpstrInitialDir = NULL;
        opfn.lpstrTitle = dlgTitle;
        opfn.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY |
                                OFN_NONETWORKBUTTON | OFN_PATHMUSTEXIST;
        opfn.nFileOffset = 0;
        opfn.nFileExtension = 0;
        opfn.lpstrDefExt = NULL;
        opfn.lCustData = 0;
        opfn.lpfnHook = 0;
        opfn.lpTemplateName = 0;

        BOOL error = GetOpenFileName(&opfn);
        if(*opfn.lpstrFile==NULL)   // // If the user did not specify a file name
                return;

        if(m_hwndFilteredAVI!=NULL)
                {
                        MCIWndDestroy(m_hwndFilteredAVI);
                        m_hwndFilteredAVI = NULL;
                }

        if(*opfn.lpstrFile!=NULL)   // If the user specified a file name
        {
                if((m_hwndFilteredAVI==NULL))
                        // If no MCI window exists create it and open file
```

```
                m_hwndFilteredAVI = MCIWndCreate(m_hWnd,AfxGetInstanceHandle(),
                        MCIWNDF_SHOWNAME |
                        MCIWNDF_SHOWMODE |
                        WS_VISIBLE       |
                        WS_BORDER        |
                        WS_CHILD,
                        opfn.lpstrFile);
        }

        RECT rCurrent;
        ::GetWindowRect(m_hwndFilteredAVI,&rCurrent);

        ::MoveWindow(m_hwndFilteredAVI,rCurrent.right,0/*rCurrent.top*/,
                        (rCurrent.right-rCurrent.left), (rCurrent.bottom-rCurrent.top), TRUE);
}
```

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Filter3D.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
	//{{AFX_MSG_MAP(CMainFrame)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//   DO NOT EDIT what you see in these blocks of generated code !
	ON_WM_CREATE()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
	ID_SEPARATOR,		// status line indicator
	ID_INDICATOR_CAPS,
	ID_INDICATOR_NUM,
	ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
	// TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
		return -1;

	if (!m_wndToolBar.Create(this) ||
		!m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
	{
		TRACE0("Failed to create toolbar\n");
```

```
		return -1;	// fail to create
	}

	if (!m_wndStatusBar.Create(this) ||
		!m_wndStatusBar.SetIndicators(indicators,
		  sizeof(indicators)/sizeof(UINT)))
	{
		TRACE0("Failed to create status bar\n");
		return -1;	// fail to create
	}

	// TODO: Remove this if you don't want tool tips or a resizeable toolbar
	m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
		CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

	// TODO: Delete these three lines if you don't want the toolbar to
	//  be dockable
	m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
	EnableDocking(CBRS_ALIGN_ANY);
	DockControlBar(&m_wndToolBar);

	return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	cs.x = 0;
	cs.y = 0;
	cs.cy = ::GetSystemMetrics(SM_CYSCREEN)-30;
  cs.cx = ::GetSystemMetrics(SM_CXSCREEN);

	return CMDIFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
	CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
	CMDIFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

```
// ProcessingDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Filter3D.h"
#include "ProcessingDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CProcessingDlg dialog


CProcessingDlg::CProcessingDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CProcessingDlg::IDD, pParent)
{
	// Create a modeless dialog box
	Create(IDD_PROCESSING,NULL);
	//{{AFX_DATA_INIT(CProcessingDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}


void CProcessingDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CProcessingDlg)
	DDX_Control(pDX, IDC_PROGRESS, m_progressCtrl);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CProcessingDlg, CDialog)
	//{{AFX_MSG_MAP(CProcessingDlg)
		// NOTE: the ClassWizard will add message map macros here
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CProcessingDlg message handlers
```

```
// ChildFrm.h : interface of the CChildFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_CHILDFRM_H__C8CEC46B_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
#define AFX_CHILDFRM_H__C8CEC46B_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CChildFrame : public CMDIChildWnd
{
	DECLARE_DYNCREATE(CChildFrame)
public:
	CChildFrame();

// Attributes
public:

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CChildFrame)
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CChildFrame();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
	//{{AFX_MSG(CChildFrame)
		// NOTE - the ClassWizard will add and remove member functions here.
		//   DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_CHILDFRM_H__C8CEC46B_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
```

```
/////////////////////////////////////////////////////////////////////////////
// Complex.h : interface of the Complex class
//

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include<math.h>

class COMPLEX
{
protected:
        double Real,Imag;

public:
        COMPLEX();
        COMPLEX(double,double);
        double GetReal(void) const;
        double GetImag(void) const;
        double Magnitude(void);
        double Phase(void);

        friend COMPLEX operator+( COMPLEX, COMPLEX );
        friend COMPLEX operator-( COMPLEX, COMPLEX );
        friend COMPLEX operator*( COMPLEX, COMPLEX );
        friend COMPLEX operator*( COMPLEX, double  );
        friend COMPLEX operator/( COMPLEX, double  );
};
```

```
// Filter3D.h : main header file for the FILTER3D application
//

#if !defined(AFX_FILTER3D_H__C8CEC465_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
#define AFX_FILTER3D_H__C8CEC465_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"	// main symbols

/////////////////////////////////////////////////////////////////////////////
// CFilter3DApp:
// See Filter3D.cpp for the implementation of this class
//

class CFilter3DApp : public CWinApp
{
public:
	CFilter3DApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CFilter3DApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CFilter3DApp)
	afx_msg void OnAppAbout();
		// NOTE - the ClassWizard will add and remove member functions here.
		//   DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FILTER3D_H__C8CEC465_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
```

```
// Filter3DDoc.h : interface of the CFilter3DDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_FILTER3DDOC_H__C8CEC46D_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
#define AFX_FILTER3DDOC_H__C8CEC46D_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "Complex.h"

class CFilter3DDoc : public CDocument
{
protected: // create from serialization only
	CFilter3DDoc();
	DECLARE_DYNCREATE(CFilter3DDoc)

// Attributes
public:

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CFilter3DDoc)
	public:
	virtual BOOL OnNewDocument();
	virtual void Serialize(CArchive& ar);
	virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
	//}}AFX_VIRTUAL

// Implementation
public:
	void SetSaveFileName(LPTSTR);
	void SetOpenFileName(LPTSTR);
	double* GetImpulseAxis(void);
	double* GetFreqAxis(void);
	unsigned GetPhaseResponseSize( void );
	double*** GetPhaseResponse( void );
	unsigned GetMagnitudeResponseSize( void );
	double*** GetMagnitudeResponse( void );
	unsigned GetImpulseResponseSize( void );
	double*** GetImpulseResponse( void );
	unsigned GetNumCoefficients( void );
	double*** GetBCoefficients( void );
	double *** GetACoefficients( void );
	BOOL FilterMovie(void);
	virtual ~CFilter3DDoc();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:
```

```
// Generated message map functions
protected:
        CString m_lpstrNewFileName;
        CString m_lpstrFileName;
        void InsertPalette(BITMAPINFOHEADER,BYTE *,DWORD);
        void InsertHeader(BITMAPINFOHEADER,BYTE *,BYTE *);
        double *** outputBuffer;
        BYTE *** inputBuffer;
        void ExtractBMPHeader(BITMAPINFOHEADER &, BYTE *);
        double* m_pdImpulseAxis;
        double* m_pdFreqAxis;
        double pi;
        void FFT1D(COMPLEX *,unsigned,unsigned);
        void FFT3D(COMPLEX ***,unsigned,unsigned);
        void BitReversal( unsigned*,unsigned);
        BOOL Simq(double **, unsigned);
        void ComputeCoefficients(void);
        double*** m_pBCoeffArray;
        double*** m_pACoeffArray;
        double*** m_pPhaseResponse;
        double*** m_pMagnitudeResponse;
        double*** m_pImpulseResponse;
        unsigned m_nOffset;
        unsigned m_nOrder;
        unsigned m_nNumSamples;
        double m_dCutoffFreq;
        //{{AFX_MSG(CFilter3DDoc)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FILTER3DDOC_H__C8CEC46D_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
```

```
#if !defined(AFX_FILTER3DSETTINGSDLG_H__C6B545E1_8E8C_11D2_9E39_0020AFDA97B0__INCLUDED_)
#define AFX_FILTER3DSETTINGSDLG_H__C6B545E1_8E8C_11D2_9E39_0020AFDA97B0__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Filter3DSettingsDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CFilter3DSettingsDlg dialog

class CFilter3DSettingsDlg : public CDialog
{
// Construction
public:
	CFilter3DSettingsDlg(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	//{{AFX_DATA(CFilter3DSettingsDlg)
	enum { IDD = IDD_SETTINGS };
	double	m_dCutoffFreq;
	UINT	m_nOffset;
	UINT	m_nOrder;
	UINT	m_nNumSamples;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CFilter3DSettingsDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CFilter3DSettingsDlg)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FILTER3DSETTINGSDLG_H__C6B545E1_8E8C_11D2_9E39_0020AFDA97B0__INCLUDED_)
```

```
// Filter3DView.h : interface of the CFilter3DView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_FILTER3DVIEW_H__C8CEC46F_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
#define AFX_FILTER3DVIEW_H__C8CEC46F_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CFilter3DView : public CView
{
protected: // create from serialization only
        CFilter3DView();
        DECLARE_DYNCREATE(CFilter3DView)

// Attributes
public:
        CFilter3DDoc* GetDocument();

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CFilter3DView)
        public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CFilter3DView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        HWND m_hwndFilteredAVI;
        HWND m_hwndOriginalAVI;
        double m_dRotationZ;
        double m_dRotationY;
        double m_dRotationX;
        void TransformPoints( double &,double &,double & );
        void MakePlotFitWindow();
        POINT ConvertToScreen( double,double,double );
```

```
        double pi;
        double m_dConstAxisFreq;
        double* m_pdAxisZ;
        double* m_pdAxisX;
        unsigned m_nNumDataPoints;
        CPoint** m_pDisplayPointArray;
        double*** m_pDataPointArray;
        unsigned m_nNumCoefficients;
        double*** m_pBCoeffArray;
        double*** m_pACoeffArray;
        unsigned m_nDisplay;
        void PlotPoints( CDC * );
        void DrawCoefficients( CDC * );
        //{{AFX_MSG(CFilter3DView)
        afx_msg void OnDisplayCoefficients();
        afx_msg void OnDisplayImpulse();
        afx_msg void OnDisplayMagnitude();
        afx_msg void OnDisplayPhase();
        afx_msg void OnVideoOpen();
        afx_msg void OnVideoPlay();
        afx_msg void OnVideoFilter();
        afx_msg void OnVideoOpenOutput();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in Filter3DView.cpp
inline CFilter3DDoc* CFilter3DView::GetDocument()
  { return (CFilter3DDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FILTER3DVIEW_H__C8CEC46F_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
```

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__C8CEC469_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
#define AFX_MAINFRM_H__C8CEC469_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CMDIFrameWnd
{
	DECLARE_DYNAMIC(CMainFrame)
public:
	CMainFrame();

// Attributes
public:

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMainFrame)
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CMainFrame();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
	CStatusBar  m_wndStatusBar;
	CToolBar    m_wndToolBar;

// Generated message map functions
protected:
	//{{AFX_MSG(CMainFrame)
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
		// NOTE - the ClassWizard will add and remove member functions here.
		//   DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MAINFRM_H__C8CEC469_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
```

```
#if !defined(AFX_PROCESSINGDLG_H__C49A1C01_94F8_11D2_9E39_0020AFDA97B0__INCLUDED_)
#define AFX_PROCESSINGDLG_H__C49A1C01_94F8_11D2_9E39_0020AFDA97B0__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ProcessingDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CProcessingDlg dialog

class CProcessingDlg : public CDialog
{
// Construction
public:
	CProcessingDlg(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	//{{AFX_DATA(CProcessingDlg)
	enum { IDD = IDD_PROCESSING };
	CProgressCtrl	m_progressCtrl;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CProcessingDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CProcessingDlg)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_PROCESSINGDLG_H__C49A1C01_94F8_11D2_9E39_0020AFDA97B0__INCLUDED_)
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Filter3D.rc
//
#define IDD_ABOUTBOX            100
#define IDR_MAINFRAME           128
#define IDR_FILTERTYPE          129
#define IDD_SETTINGS            131
#define IDD_PROCESSING          133
#define IDC_SAMPLES             1000
#define IDC_CUTOFF              1001
#define IDC_PROGRESS            1001
#define IDC_ORDER               1002
#define IDC_OFFSET              1003
#define ID_DISPLAY_IMPULSE          32771
#define ID_DISPLAY_MAGNITUDE        32772
#define ID_DISPLAY_PHASE            32773
#define ID_DISPLAY_COEFFICIENTS     32774
#define ID_VIDEO_OPEN               32776
#define ID_VIDEO_FILTER             32777
#define ID_VIDEO_PLAY               32779
#define ID_VIDEO_OPENOUTPUTDEMO     32783
#define ID_VIDEO_OPENOUTPUT         32784

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS            1
#define _APS_NEXT_RESOURCE_VALUE    134
#define _APS_NEXT_COMMAND_VALUE     32786
#define _APS_NEXT_CONTROL_VALUE     1002
#define _APS_NEXT_SYMED_VALUE       101
#endif
#endif
```

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

/////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

/////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
  "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
  "#include ""afxres.h""\r\n"
  "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
  "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
  "#define _AFX_NO_OLE_RESOURCES\r\n"
  "#define _AFX_NO_TRACKER_RESOURCES\r\n"
  "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
  "\r\n"
  "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
  "#ifdef _WIN32\r\n"
  "LANGUAGE 9, 1\r\n"
  "#pragma code_page(1252)\r\n"
  "#endif\r\n"
  "#include ""res\\Filter3D.rc2""  // non-Microsoft Visual C++ edited resources\r\n"
  "#include ""afxres.rc""        // Standard components\r\n"
  "#include ""afxprint.rc""       // printing/print preview resources\r\n"
  "#endif\0"
END
```

```
#endif   // APSTUDIO_INVOKED


/////////////////////////////////////////////////////////////////////////////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME        ICON   DISCARDABLE    "res\\Filter3D.ico"
IDR_FILTERTYPE       ICON   DISCARDABLE    "res\\Filter3DDoc.ico"

/////////////////////////////////////////////////////////////////////////////
//
// Bitmap
//

IDR_MAINFRAME        BITMAP  MOVEABLE PURE   "res\\mainfram.bmp"

/////////////////////////////////////////////////////////////////////////////
//
// Toolbar
//

IDR_MAINFRAME TOOLBAR DISCARDABLE  16, 15
BEGIN
  BUTTON    ID_FILE_NEW
  BUTTON    ID_FILE_OPEN
  BUTTON    ID_FILE_SAVE
  SEPARATOR
  BUTTON    ID_EDIT_CUT
  BUTTON    ID_EDIT_COPY
  BUTTON    ID_EDIT_PASTE
  SEPARATOR
  BUTTON    ID_FILE_PRINT
  BUTTON    ID_APP_ABOUT
  SEPARATOR
  BUTTON    ID_DISPLAY_IMPULSE
  BUTTON    ID_DISPLAY_MAGNITUDE
  BUTTON    ID_DISPLAY_PHASE
  BUTTON    ID_DISPLAY_COEFFICIENTS
  SEPARATOR
  BUTTON    ID_VIDEO_OPEN
  BUTTON    ID_VIDEO_FILTER
  BUTTON    ID_VIDEO_PLAY
END


/////////////////////////////////////////////////////////////////////////////
//
// Menu
//

IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
  POPUP "&File"
```

```
  BEGIN
    MENUITEM "&New\tCtrl+N",          ID_FILE_NEW
    MENUITEM "&Open...\tCtrl+O",      ID_FILE_OPEN
    MENUITEM SEPARATOR
    MENUITEM "P&rint Setup...",       ID_FILE_PRINT_SETUP
    MENUITEM SEPARATOR
    MENUITEM "Recent File",           ID_FILE_MRU_FILE1. GRAYED
    MENUITEM SEPARATOR
    MENUITEM "E&xit",                 ID_APP_EXIT
  END
  POPUP "&View"
  BEGIN
    MENUITEM "&Toolbar",              ID_VIEW_TOOLBAR
    MENUITEM "&Status Bar",           ID_VIEW_STATUS_BAR
  END
  POPUP "&Help"
  BEGIN
    MENUITEM "&About Filter3D...",    ID_APP_ABOUT
  END
END

IDR_FILTERTYPE MENU PRELOAD DISCARDABLE
BEGIN
  POPUP "&File"
  BEGIN
    MENUITEM "&New\tCtrl+N",          ID_FILE_NEW
    MENUITEM "&Open...\tCtrl+O",      ID_FILE_OPEN
    MENUITEM "&Close",                ID_FILE_CLOSE
    MENUITEM "&Save\tCtrl+S",         ID_FILE_SAVE
    MENUITEM "Save &As...",           ID_FILE_SAVE_AS
    MENUITEM SEPARATOR
    MENUITEM "&Print...\tCtrl+P",     ID_FILE_PRINT
    MENUITEM "Print Pre&view",        ID_FILE_PRINT_PREVIEW
    MENUITEM "P&rint Setup...",       ID_FILE_PRINT_SETUP
    MENUITEM SEPARATOR
    MENUITEM "Recent File",           ID_FILE_MRU_FILE1. GRAYED
    MENUITEM SEPARATOR
    MENUITEM "E&xit",                 ID_APP_EXIT
  END
  POPUP "&Edit"
  BEGIN
    MENUITEM "&Undo\tCtrl+Z",         ID_EDIT_UNDO
    MENUITEM SEPARATOR
    MENUITEM "Cu&t\tCtrl+X",          ID_EDIT_CUT
    MENUITEM "&Copy\tCtrl+C",         ID_EDIT_COPY
    MENUITEM "&Paste\tCtrl+V",        ID_EDIT_PASTE
  END
  POPUP "&View"
  BEGIN
    MENUITEM "&Toolbar",              ID_VIEW_TOOLBAR
    MENUITEM "&Status Bar",           ID_VIEW_STATUS_BAR
  END
  POPUP "&Window"
  BEGIN
    MENUITEM "&New Window",           ID_WINDOW_NEW
    MENUITEM "&Cascade",              ID_WINDOW_CASCADE
    MENUITEM "&Tile",                 ID_WINDOW_TILE_HORZ
```

```
    MENUITEM "&Arrange Icons",          ID_WINDOW_ARRANGE
  END
  POPUP "&Help"
  BEGIN
    MENUITEM "&About Filter3D...",      ID_APP_ABOUT
  END
  POPUP "&Display"
  BEGIN
    MENUITEM "&Impulse",                ID_DISPLAY_IMPULSE
    MENUITEM "&Magnitude",              ID_DISPLAY_MAGNITUDE
    MENUITEM "&Phase",                  ID_DISPLAY_PHASE
    MENUITEM "&Coefficients",           ID_DISPLAY_COEFFICIENTS
  END
  POPUP "&Video"
  BEGIN
    MENUITEM "&Open",                   ID_VIDEO_OPEN
    MENUITEM "&Filter",                 ID_VIDEO_FILTER
    MENUITEM "&Play Both",              ID_VIDEO_PLAY
    MENUITEM SEPARATOR
    MENUITEM "O&pen Output (Demo)",     ID_VIDEO_OPENOUTPUT
  END
END


/////////////////////////////////////////////////////////////////////////////
//
// Accelerator
//

IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
  "N",        ID_FILE_NEW,        VIRTKEY, CONTROL
  "O",        ID_FILE_OPEN,       VIRTKEY, CONTROL
  "S",        ID_FILE_SAVE,       VIRTKEY, CONTROL
  "P",        ID_FILE_PRINT,      VIRTKEY, CONTROL
  "Z",        ID_EDIT_UNDO,       VIRTKEY, CONTROL
  "X",        ID_EDIT_CUT,        VIRTKEY, CONTROL
  "C",        ID_EDIT_COPY,       VIRTKEY, CONTROL
  "V",        ID_EDIT_PASTE,      VIRTKEY, CONTROL
  VK_BACK,    ID_EDIT_UNDO,       VIRTKEY, ALT
  VK_DELETE,  ID_EDIT_CUT,        VIRTKEY, SHIFT
  VK_INSERT,  ID_EDIT_COPY,       VIRTKEY, CONTROL
  VK_INSERT,  ID_EDIT_PASTE,      VIRTKEY, SHIFT
  VK_F6,      ID_NEXT_PANE,       VIRTKEY
  VK_F6,      ID_PREV_PANE,       VIRTKEY, SHIFT
END


/////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 217, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About Filter3D"
FONT 8, "MS Sans Serif"
```

```
BEGIN
  ICON          IDR_MAINFRAME,IDC_STATIC,11,17,20,20
  LTEXT         "Filter3D Version 1.0",IDC_STATIC,40,10,119,8,
            SS_NOPREFIX
  LTEXT         "Copyright (C) 1998",IDC_STATIC,40,25,119,8
  DEFPUSHBUTTON   "OK",IDOK,178,7,32,14,WS_GROUP
END

IDD_SETTINGS DIALOG DISCARDABLE  0, 0, 174, 72
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Settings"
FONT 8, "MS Sans Serif"
BEGIN
  EDITTEXT      IDC_SAMPLES,45,5,35,12,ES_AUTOHSCROLL
  EDITTEXT      IDC_CUTOFF,45,20,35,12,ES_AUTOHSCROLL
  EDITTEXT      IDC_ORDER,45,35,35,12,ES_AUTOHSCROLL
  EDITTEXT      IDC_OFFSET,45,50,35,12,ES_AUTOHSCROLL
  DEFPUSHBUTTON   "OK",IDOK,112,11,50,14
  PUSHBUTTON      "Cancel",IDCANCEL,112,28,50,14
  LTEXT         "Samples:",IDC_STATIC,9,7,35,10
  LTEXT         "Cutoff:",IDC_STATIC,9,22,35,10
  LTEXT         "Order:",IDC_STATIC,9,37,35,10
  LTEXT         "Offset:",IDC_STATIC,9,52,35,10
END

IDD_PROCESSING DIALOG DISCARDABLE  0, 0, 137, 46
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CENTER | WS_POPUP | WS_VISIBLE
FONT 8, "MS Sans Serif"
BEGIN
  CONTROL       "Progress1",IDC_PROGRESS,"msctls_progress32",WS_BORDER,
            12,25,113,14
  CTEXT         "Processing...",IDC_STATIC,45,7,48,10
END


#ifndef _MAC
/////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
  BLOCK "StringFileInfo"
  BEGIN
    BLOCK "040904B0"
```

```
      BEGIN
        VALUE "CompanyName", "\0"
        VALUE "FileDescription", "Filter3D MFC Application\0"
        VALUE "FileVersion", "1, 0, 0, 1\0"
        VALUE "InternalName", "Filter3D\0"
        VALUE "LegalCopyright", "Copyright (C) 1998\0"
        VALUE "LegalTrademarks", "\0"
        VALUE "OriginalFilename", "Filter3D.EXE\0"
        VALUE "ProductName", "Filter3D Application\0"
        VALUE "ProductVersion", "1, 0, 0, 1\0"
      END
    END
    BLOCK "VarFileInfo"
    BEGIN
      VALUE "Translation", 0x409, 1200
    END
  END

#endif   // !_MAC


/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
  IDD_ABOUTBOX, DIALOG
  BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 210
    TOPMARGIN, 7
    BOTTOMMARGIN, 48
  END

  IDD_SETTINGS, DIALOG
  BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 167
    TOPMARGIN, 7
    BOTTOMMARGIN, 65
  END

  IDD_PROCESSING, DIALOG
  BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 130
    TOPMARGIN, 7
    BOTTOMMARGIN, 39
  END
END
#endif   // APSTUDIO_INVOKED


/////////////////////////////////////////////////////////////////////////////
```

```
//
// String Table
//

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
  IDR_MAINFRAME        "Filter3D"
  IDR_FILTERTYPE       "\nFilter\nFilter\nFilter Files (*.f3d)\n.f3d\nFilter3D.Document\nFilter Document"
END

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
  AFX_IDS_APP_TITLE     "Filter3D"
  AFX_IDS_IDLEMESSAGE   "Ready"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_INDICATOR_EXT      "EXT"
  ID_INDICATOR_CAPS     "CAP"
  ID_INDICATOR_NUM      "NUM"
  ID_INDICATOR_SCRL     "SCRL"
  ID_INDICATOR_OVR      "OVR"
  ID_INDICATOR_REC      "REC"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_FILE_NEW           "Create a new document\nNew"
  ID_FILE_OPEN          "Open an existing document\nOpen"
  ID_FILE_CLOSE         "Close the active document\nClose"
  ID_FILE_SAVE          "Save the active document\nSave"
  ID_FILE_SAVE_AS       "Save the active document with a new name\nSave As"
  ID_FILE_PAGE_SETUP    "Change the printing options\nPage Setup"
  ID_FILE_PRINT_SETUP   "Change the printer and printing options\nPrint Setup"
  ID_FILE_PRINT         "Print the active document\nPrint"
  ID_FILE_PRINT_PREVIEW "Display full pages\nPrint Preview"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_APP_ABOUT          "Display program information, version number and copyright\nAbout"
  ID_APP_EXIT           "Quit the application; prompts to save documents\nExit"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_FILE_MRU_FILE1     "Open this document"
  ID_FILE_MRU_FILE2     "Open this document"
  ID_FILE_MRU_FILE3     "Open this document"
  ID_FILE_MRU_FILE4     "Open this document"
  ID_FILE_MRU_FILE5     "Open this document"
  ID_FILE_MRU_FILE6     "Open this document"
  ID_FILE_MRU_FILE7     "Open this document"
  ID_FILE_MRU_FILE8     "Open this document"
  ID_FILE_MRU_FILE9     "Open this document"
  ID_FILE_MRU_FILE10    "Open this document"
```

```
    ID_FILE_MRU_FILE11     "Open this document"
    ID_FILE_MRU_FILE12     "Open this document"
    ID_FILE_MRU_FILE13     "Open this document"
    ID_FILE_MRU_FILE14     "Open this document"
    ID_FILE_MRU_FILE15     "Open this document"
    ID_FILE_MRU_FILE16     "Open this document"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_NEXT_PANE          "Switch to the next window pane\nNext Pane"
    ID_PREV_PANE          "Switch back to the previous window pane\nPrevious Pane"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_WINDOW_NEW         "Open another window for the active document\nNew Window"
    ID_WINDOW_ARRANGE     "Arrange icons at the bottom of the window\nArrange Icons"
    ID_WINDOW_CASCADE     "Arrange windows so they overlap\nCascade Windows"
    ID_WINDOW_TILE_HORZ   "Arrange windows as non-overlapping tiles\nTile Windows"
    ID_WINDOW_TILE_VERT   "Arrange windows as non-overlapping tiles\nTile Windows"
    ID_WINDOW_SPLIT       "Split the active window into panes\nSplit"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_EDIT_CLEAR         "Erase the selection\nErase"
    ID_EDIT_CLEAR_ALL     "Erase everything\nErase All"
    ID_EDIT_COPY          "Copy the selection and put it on the Clipboard\nCopy"
    ID_EDIT_CUT           "Cut the selection and put it on the Clipboard\nCut"
    ID_EDIT_FIND          "Find the specified text\nFind"
    ID_EDIT_PASTE         "Insert Clipboard contents\nPaste"
    ID_EDIT_REPEAT        "Repeat the last action\nRepeat"
    ID_EDIT_REPLACE       "Replace specific text with different text\nReplace"
    ID_EDIT_SELECT_ALL    "Select the entire document\nSelect All"
    ID_EDIT_UNDO          "Undo the last action\nUndo"
    ID_EDIT_REDO          "Redo the previously undone action\nRedo"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_VIEW_TOOLBAR       "Show or hide the toolbar\nToggle ToolBar"
    ID_VIEW_STATUS_BAR    "Show or hide the status bar\nToggle StatusBar"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCSIZE        "Change the window size"
    AFX_IDS_SCMOVE        "Change the window position"
    AFX_IDS_SCMINIMIZE    "Reduce the window to an icon"
    AFX_IDS_SCMAXIMIZE    "Enlarge the window to full size"
    AFX_IDS_SCNEXTWINDOW  "Switch to the next document window"
    AFX_IDS_SCPREVWINDOW  "Switch to the previous document window"
    AFX_IDS_SCCLOSE       "Close the active window and prompts to save the documents"
END

STRINGTABLE DISCARDABLE
```

```
BEGIN
  AFX_IDS_SCRESTORE      "Restore the window to normal size"
  AFX_IDS_SCTASKLIST     "Activate Task List"
  AFX_IDS_MDICHILD       "Activate this window"
END

STRINGTABLE DISCARDABLE
BEGIN
  AFX_IDS_PREVIEW_CLOSE  "Close print preview mode\nCancel Preview"
END

STRINGTABLE DISCARDABLE
BEGIN
  ID_DISPLAY_IMPULSE     "Plot Impulse Response\nImpulseResponse"
  ID_DISPLAY_MAGNITUDE   "Plot Magnitude Response\nMagnitude Response"
  ID_DISPLAY_PHASE       "Plot Phase Response\nPhase Response"
  ID_DISPLAY_COEFFICIENTS "Display Coefficients\nCoefficients"
  ID_VIDEO_OPEN          "Open a Video File for Filtering\nOpen Movie"
  ID_VIDEO_FILTER        "Filter Video Clip\nFilter Movie"
  ID_VIDEO_PLAY          "Play Both Video Clips at the Same Time\nPlay Both Clips"
  ID_VIDEO_OPENOUTPUTDEMO "Open a Second Video Clip for Comparison\nOpen Output"
END

#endif   // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////



#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif
#include "res\Filter3D.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#include "afxprint.rc"       // printing/print preview resources
#endif
/////////////////////////////////////////////////////////////////////////////
#endif   // not APSTUDIO_INVOKED
```

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__C8CEC467_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
#define AFX_STDAFX_H__C8CEC467_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN            // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                         // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__C8CEC467_8DFA_11D2_9E39_0020AFDA97B0__INCLUDED_)
```

Appendix B

Matlab Source Code for Generating Magnitude Response, Phase Response, and Group Delay Plots

```
% thesis_magnitude.m - Generates plots of 3-D filter magnitude response
%                       Responses were generated by Filter3D.exe and are
%                       not included due to excessive length

% w3 = 0.0
H=[ ... ];

[w1,w2] = freqspace(length(H));
w1 = w1.*pi;
w2 = w1;

figure
mesh(w1,w2,H),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w1 = 0 rad/sec)')
xlabel('w2 (rad/unit)'), ylabel('w3 (rad/unit)'), zlabel('Magnitude Response')
axis([-4 4 -4 4 0 1.2])


% w3 = 0.98
H=[ ... ];

figure
mesh(w1,w2,H),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w1 = 0.98 rad/sec)')
xlabel('w2 (rad/unit)'), ylabel('w3 (rad/unit)'), zlabel('Magnitude Response')
axis([-4 4 -4 4 0 1.2])


% w3 = 2.16
H=[ ... ];

figure
mesh(w1,w2,H),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w1 = 2.16 rad/sec)')
xlabel('w2 (rad/unit)'), ylabel('w3 (rad/unit)'), zlabel('Magnitude Response')
axis([-4 4 -4 4 0 1.2])


% w3 = pi
H=[ ... ];


figure,
mesh(w1,w2,H),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w1 = pi rad/sec)')
xlabel('w2 (rad/unit)'), ylabel('w3 (rad/unit)'), zlabel('Magnitude Response')
axis([-4 4 -4 4 0 1.2])
```

```
% thesis_phase.m - Generates plots of 3-D filter phase response
%                   and group delay
%                   Responses were generated by Filter3D.exe and are
%                   not included due to excessive length

% w3 = 0.0
P=[ ... ];

[w1,w2] = freqspace(length(P));
w1 = w1.*pi;
w2 = w1;
winc = w2(2)-w2(1);

P = unwrap(P);
figure
mesh(w1,w2,P),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w3 = 0 radians/sec)')
xlabel('w1 (radians/unit)'), ylabel('w2 (radians/unit)'), zlabel('Phase
Response (radians)')
view(-20,60)

for l=1:length(P)
   for k=1:length(P)-1
      if abs(P(l,k+1)-P(l,k)) < 5
         Gd(l,k) = -(P(l,k+1)-P(l,k))/winc;
      else
         if P(l,k+1)-P(l,k) < 0
            Gd(l,k) = -(P(l,k+1)-(P(l,k)-2*pi))/winc;
         else
            Gd(l,k) = -(P(l,k+1)-(P(l,k)+2*pi))/winc;
         end
      end
   end
   Gd(l,length(P)) = -(P(l,length(P))-P(l,length(P)-1))/winc;
end

figure,
mesh(w1,w2,Gd),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w3=0 radians/sec)')
xlabel('w1 (radians/unit)'), ylabel('w2 (radians/unit)')
zlabel('Group Delay with Respect to w2')
view(0,0)



% w3 = 0.98
P=[ ... ];

P = unwrap(P);

figure
mesh(w1,w2,P),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w3 = 0.98 radians/sec)')
xlabel('w1 (radians/unit)'), ylabel('w2 (radians/unit)')
zlabel('Group Delay with Respect to w2')
view(-20,60)
```

```
for l=1:length(P)
   for k=1:length(P)-1
      if abs(P(l,k+1)-P(l,k)) < 5
         Gd(l,k) = -(P(l,k+1)-P(l,k))/winc;
      else
         if P(l,k+1)-P(l,k) < 0
            Gd(l,k) = -(P(l,k+1)-(P(l,k)-2*pi))/winc;
         else
            Gd(l,k) = -(P(l,k+1)-(P(l,k)+2*pi))/winc;
         end
      end
   end
   Gd(l,length(P)) = -(P(l,length(P))-P(l,length(P)-1))/winc;
end

figure
mesh(w1,w2,Gd),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w3=0.98 radians/sec)')
xlabel('w1 (radians/unit)'), ylabel('w2 (radians/unit)')
zlabel('Group Delay with Respect to w1')
view(0,0)


% w3 = 2.16
P=[ ... ];

P = unwrap(P);
figure
mesh(w1,w2,P),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w3 = 2.16 radians/sec)')
xlabel('w1 (radians/unit)'), ylabel('w2 (radians/unit)')
zlabel('Group Delay with Respect to w2')
view(-20,60)

for l=1:length(P)
   for k=1:length(P)-1
      if abs(P(l,k+1)-P(l,k)) < 5
         Gd(l,k) = -(P(l,k+1)-P(l,k))/winc;
      else
         if P(l,k+1)-P(l,k) < 0
            Gd(l,k) = -(P(l,k+1)-(P(l,k)-2*pi))/winc;
         else
            Gd(l,k) = -(P(l,k+1)-(P(l,k)+2*pi))/winc;
         end
      end
   end
   if abs(P(l,length(P))-P(l,length(P)-1)) < 5
      Gd(l,length(P)) = -(P(l,length(P))-P(l,length(P)-1))/winc;
   end
end

figure
mesh(w1,w2,Gd),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w3 = 2.16 radians/sec)')
xlabel('w1 (radians/unit)'), ylabel('w2 (radians/unit)')
zlabel('Group Delay with Respect to w1')
view(0,0)
```

```
% w3 = pi
P=[ ... ];

P = unwrap(P);
figure
mesh(w1,w2,P),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w3 = pi radians/sec)')
xlabel('w1 (radians/unit)'), ylabel('w2 (radians/unit)')
zlabel('Group Delay with Respect to w2')
view(-20,60)

for l=1:length(P)
   for k=1:length(P)-1
      if abs(P(l,k+1)-P(l,k)) < 5
         Gd(l,k) = -(P(l,k+1)-P(l,k))/winc;
      else
         if P(l,k+1)-P(l,k) < 0
            Gd(l,k) = -(P(l,k+1)-(P(l,k)-2*pi))/winc;
         else
            Gd(l,k) = -(P(l,k+1)-(P(l,k)+2*pi))/winc;
         end
      end
   end
   if abs(P(l,length(P))-P(l,length(P)-1)) < 5
      Gd(l,length(P)) = -(P(l,length(P))-P(l,length(P)-1))/winc;
   end
end

figure
mesh(w1,w2,Gd),rotate3d on
title('Low-Pass Filter with Cutoff = pi/2 (w3 = pi radians/sec)')
xlabel('w1 (radians/unit)'), ylabel('w2 (radians/unit)')
zlabel('Group Delay with Respect to w1')
view(0,0)
```

Vita Auctoris

Steven Bruce McFadden

1974 Born October 19th in Lindsay, Ontario, Canada

1993 High School Diploma from I. E. Weldon Secondary School, Lindsay, Ontario, Canada

1997 Bachelor of Applied Science in Electrical Engineering from University of Windsor, Windsor, Ontario, Canada

2000 Candidate for Master of Applied Science in Electrical Engineering from University of Windsor, Windsor, Ontario, Canada