Using Microsoft DirectX In a DSP Laboratory

Peter E. Goodmann, P.E.
Indiana University – Purdue University Fort Wayne

Abstract

This paper reports on the use of Microsoft DirectX as a laboratory teaching tool in a junior-level digital signal processing (DSP) course for technology students. The DirectX Software Development Kit (available as a no-cost download from Microsoft), along with Microsoft Visual C++ or Visual Studio, can turn any soundcard-equipped desktop or laptop PC into a self-contained DSP laboratory for software development, experimentation, and teaching.

Teaching DSP to ECET students represents a unique challenge, due to the hands-on emphasis compared with the more theoretically-oriented engineering curriculum. Ideally, a technology DSP course would include laboratory exercises which allow the student to experience the results of various digital signal processing functions by seeing or hearing them. Furthermore, some of the lab exercises should require the student to develop code which executes in realtime, to build an awareness of hardware limitations and the need to write efficient code. These objectives suggest the use of DSP hardware, such as the DSP evaluation modules which are available from DSP manufacturers (Analog Devices, Texas Instruments, etc.). That approach means buying multiple copies of hardware which can only be used for the DSP class, and which may represent a significant investment. The approach described here allows the student to develop and execute realtime signal-processing software using C++ and a standard PC. The PC soundcard is used for signal input and signal output, allowing students to hear the results of their DSP software. Hardware limitations imposed by the PC, while not overly restrictive, do require a bit of discipline and ingenuity on the part of the student. The low-cost of this approach makes it easy for students to equip their own home or laptop computers for DSP development, so they are not tethered to a laboratory on campus.

Introduction

The laboratory portion of a DSP course in an ECET curriculum is of particular importance due to the more hands-on, less theoretical nature of the technology curriculum compared with engineering. DSP laboratory experiments should be designed to enhance student understanding of basic DSP concepts which many students find difficult to grasp. Experiments should also expose students to real-world design limitations such as processor throughput in order to teach them the need to write efficient code. The required lab equipment should be affordable. If possible it should be so affordable that individual students can purchase it themselves if they so choose, so they can work on lab experiments outside laboratory sessions.

One common approach to DSP lab experiments is to use DSP hardware, usually in the form of an "evaluation module" (EVM) based on a popular DSP chip. An example is the Analog Devices

ADSP-2189M, which may be purchased for $175 (including the university discount). This is a reasonable price for a university outfitting a DSP lab, but it would be burdensome for a student.

Another approach is to use mathematical analysis software such as Matlab, but this would mean that only experiments in offline (i.e., not realtime) DSP could be performed. The students would not gain experience in making a DSP function execute in realtime using limited processor resources such as throughput and word length. Furthermore, the Student version of Matlab is not that much less expensive than an EVM (though Matlab would probably be useful in other courses).

The approach reported here uses the student's own personal computer as DSP hardware, with the sound card providing input and output channels. This can be done using DirectX®, a set of low-level application programming interfaces (APIs) for multimedia applications, which may be downloaded free-of-charge from Microsoft. All programming is done in Visual C++ using the Visual Studio development environment, which is available at no cost to students at institutions which subscribe to the Microsoft Developer' Network Academic Alliance (MSDNAA). Thus, a student who already has a personal computer (either desktop or laptop) can outfit a personal realtime DSP lab at little or no cost.

A DSP course should be about DSP and implementing DSP functions, not about the nuts and bolts of DirectX® programming. The student is relieved from having to become a DirectX® programmer for the sake of a few lab experiments by providing a custom AppWizard which is used within Visual Studio to create the skeleton (and much of the meat) of a DirectX® project which already contains all the required DirectX® code before the student ever touches it. The student is required to add the DSP code, but does not need to worry about the details of DirectX®. Detailed instructions are also provided for building the project and adding it to the system registry, in order to keep the emphasis on DSP and not on C++, Visual Studio, or DirectX®.

Background

The use of DirectX® and/or DirectShow for several DSP applications, including analysis of acoustic signals[1,2], cochlear implant research[3], and musical instrument modeling[4] has been previously reported, but implementation details which could have facilitated the use of DirectShow for similar applications were not described. Some useful implementation details of a software-defined radio application using the DirectSound component of DirectX® have been described[5,6], but this is not directly applicable to DirectShow. One tutorial was found which gives useful, detailed information on implementing audio effects with DirectShow[7], but it is rather limited in scope and does give all the information which is needed for more general DSP functions such as FIR, IIR and adaptive filters. Finally, while the Microsoft documentation is very detailed and correct[8], it has a tendency to tell the reader only that which he already knows[9].

DirectX® and DirectShow

DirectX® is intended for developers of multimedia applications including graphics, music, and video games. DirectShow, one of the components of DirectX®, provides realtime processing capabilities for multichannel audio (e.g., stereophonic) and video streams. This capability may be used for experiments which demonstrate the development and application of DSP techniques such as digital highpass, lowpass, and bandpass filtering using both finite impulse response (FIR) and inifinite impulse response (IIR) structures, adaptive filtering, convolution, correlation, and

modulation. It should also be possible to use DirectShow for spectrum analysis, although a DirectShow spectrum analyzer has not been developed yet.

Laboratory experiments using DirectShow are designed to demonstrate the application of DSP techniques to audio-frequency signals such as music or speech. Signal sources may include sound files such as Wave format files (.wav files) or MP3 files, or realtime inputs from a microphone or other audio signal source via the soundcard's microphone or "Line In" inputs. Processed signals may be listened to in realtime via the soundcard outputs, or may be viewed using a "virtual oscilloscope" which is included with the DirectX® SDK.

DirectShow processes signals (which the Microsoft documentation calls "streams") by connecting a set "building blocks" which are called "filters" to form "filter graph." A stream originates in a "source filter," which represents the source of a stream. This may be a media file (e.g., a wav-format sound file) or a sound card. A "renderer filter" represents the stream's destination, which may also be a media file or sound card. Between the source and destination there are one or more "transform filters" in which the stream (which contains the signal) is processed; these transform filters are where the DSP functions are implemented. Source and renderer filters are included with the DirectX® SDK, as are a number of examples of transform filters.

The DirectX® SDK includes GraphEdit, a utility which enables the student to build a filter graph while simultaneously visualizing the flow of the signal through the signal processing system. An example is shown in Figure 1:

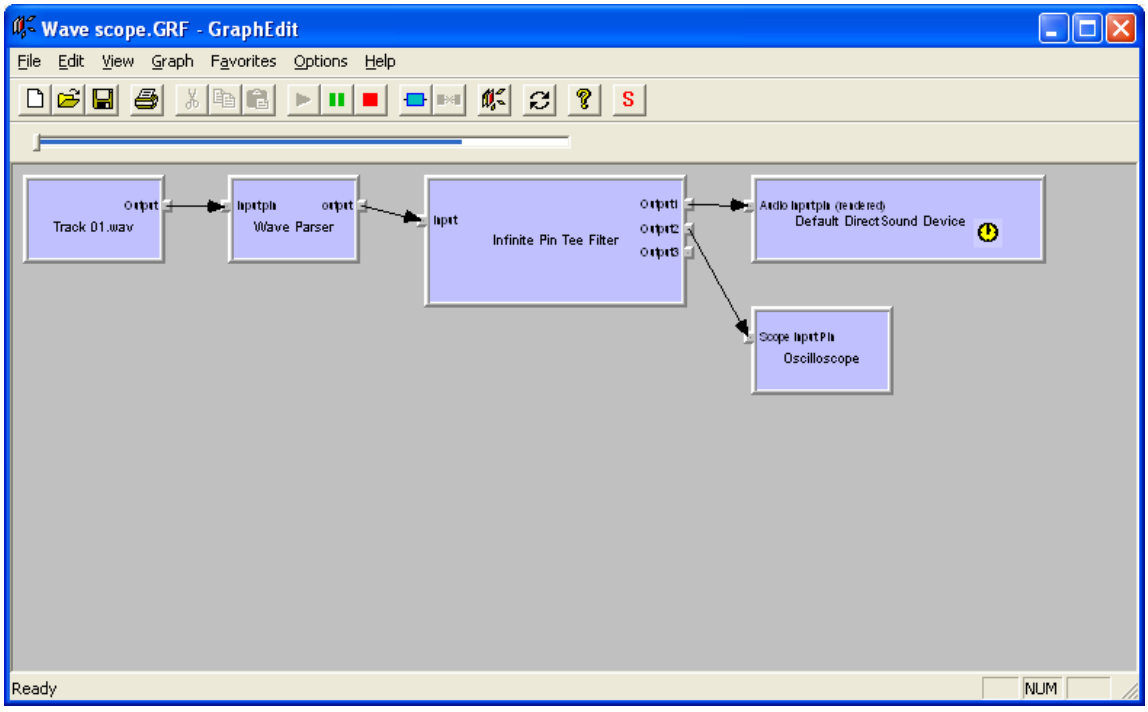

Figure 1
GraphEdit

This filter graph uses a wave-format sound file as the signal source, "parses" it into a stream, then uses a "tee" filter to send the stream simultaneously to the sound card and to a virtual oscilloscope. The virtual oscilloscope display is shown in Figure 2. The filter graph is constructed by connecting an output pin of one filter to the input pin of the next filter in the graph. Filters are selected from a dialog box containing a list of available filters. Many filters are included with the DirectX® SDK, and new filters may be added to the list by the student as they are developed.

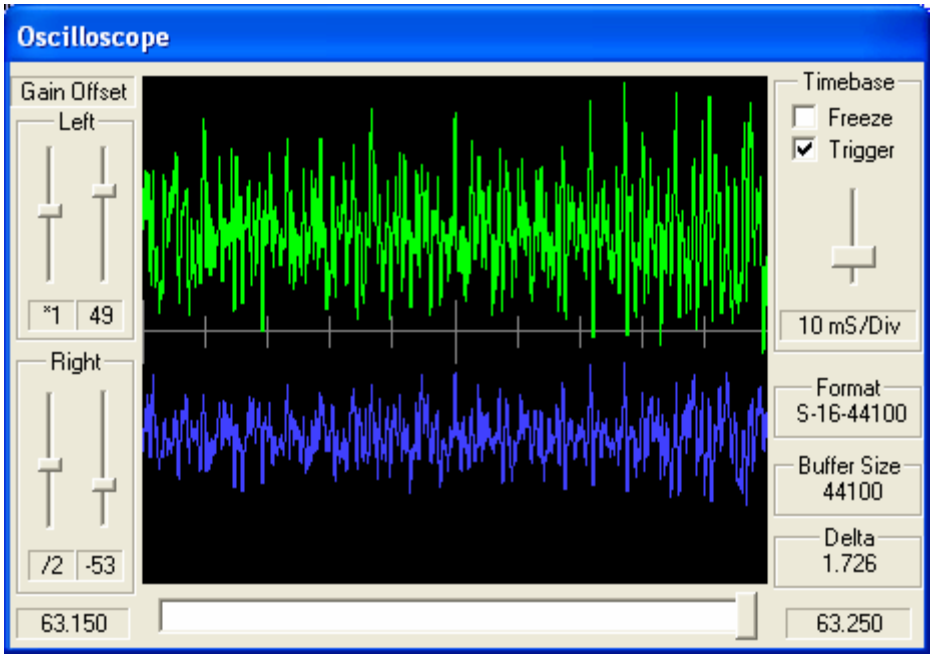

Figure 2
Virtual Oscilloscope

The Stream

In Microsoft terminology, the signal which is processed by the transform filters within the filter graph is called the "stream". The stream may contain one channel in the case of a monaural stream, two channels in the case of a stereo stream, or more than two channels. The stream's format information (word length, sample rate, number of channels) is embedded in the stream, and may be retrieved whenever it is needed.

As the stream is created by the source filter, it is divided into segments called "media samples" in the Microsoft documentation. The source filter passes a media sample to the filter immediately downstream from itself in the filter graph. That filter retrieves the format information, processes the media sample, and passes it along to the next filter in the filter graph. As a filter passes the processed media sample to the filter downstream from it, another media

sample is passed in by the upstream filter. A filter will not accept a new media sample from the upstream filter until it has finished processing the previous media sample and passed it to the downstream filter.

Transform Filters

Transform filters processes the stream, so this is where all signal processing operations are done. A filter graph may contain a single transform filter, or a series of transform filters. Each transform filter has one input pin and one output pin, but the stream which is passed into and out of the filter via the pins may contain two or more parallel data channels, so a transform filter may process multiple signal channels.

Writing a transform filter from the ground up would be a daunting task, much too challenging for an experiment which is supposed to be completed within one or two weeks. Fortunately, any transform filter is about 90% identical to any other transform filter. This identical 90%, which is concerned with meeting DirectX® requirements and not with signal processing, is provided to the student in the form of a "bare bones" transform filter project. The student creates the project using the transform filter AppWizard, which is provided by the instructor, then develops code within the project to implement the assigned DSP function. This any need for students to become DirectX® experts and helps to keep the emphasis of the lab exercise on DSP implementation instead of DirectX.

The "Karaoke" Experiment

The first DSP experiment using DirectX® has the students create a filter which cancels the lead vocalist in most stereo recordings. This is done by subtracting each sample of the Left channel from the corresponding Right channel sample, or vice-versa. No past samples need to be used, as would be necessary in an IIR of FIR filter, which makes the process very simple to implement. The student downloads an "Audio Effect Filter" custom AppWizard, which actually creates all of the code for the experiment, because the purpose of this experiment is to familiarize the student with the process of creating, building, and registering DirectX® projects. Following detailed instructions[10] which may be viewed on the course website, the student installs the AppWizard[11] and uses it to create the project. The student then builds the project and demonstrates it by constructing a filter graph with GraphEdit, as shown in Figure 3:

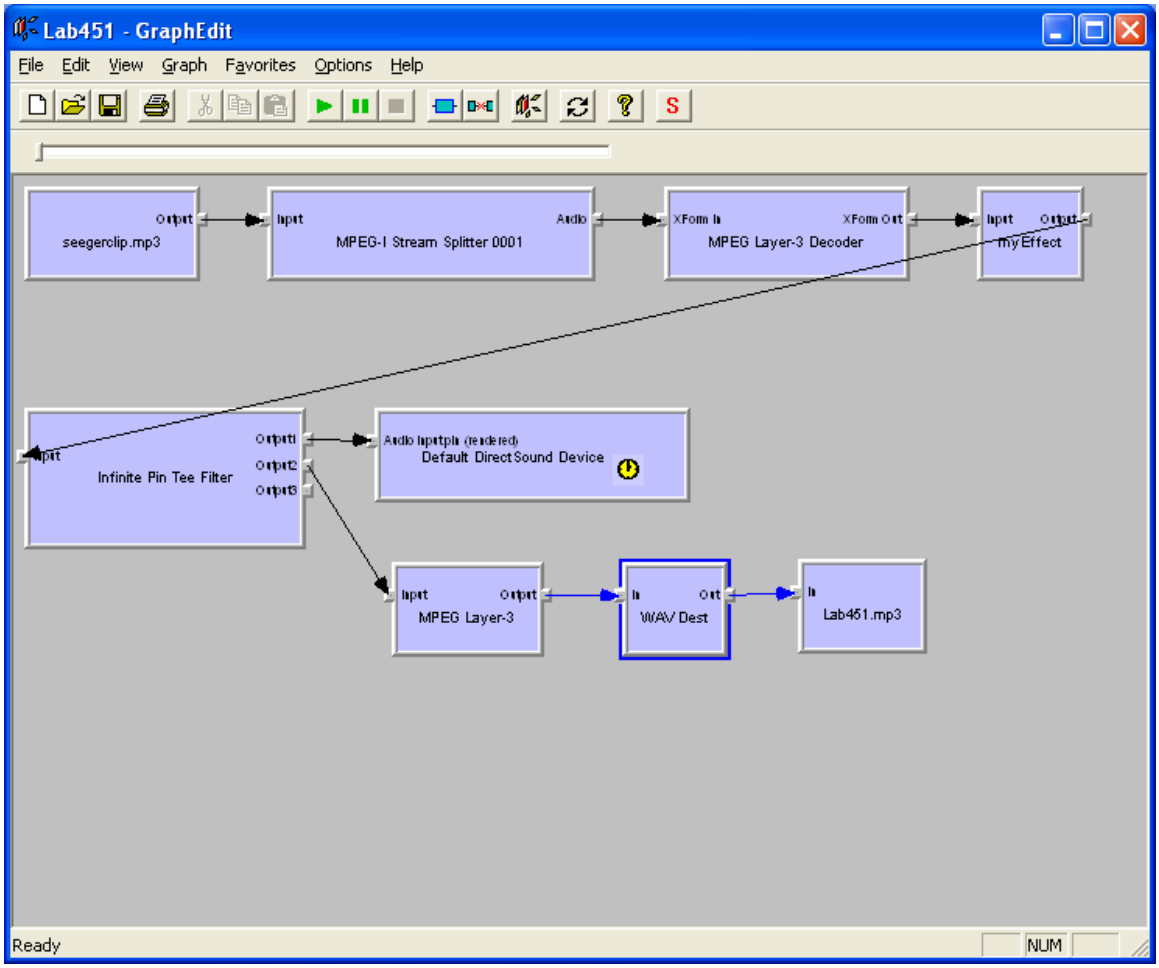

Figure 3
"Karaoke" Filter Graph

The signal source is a stereo MP3 file, which is decompressed and passed to the vocal cancellation filter ("myEffect", in Figure 3). The output of that filter is passed to the soundcard so the student may hear the result, and is also compressed and written to a new MP3 file. The student is required to turn in his Visual studio project (all the files comprising the project – source, header, executable, etc.), the file the filter graph is saved in, and the input and output MP3 files.

The FIR filter Experiment

The second DirectX® experiment[12] is a bit more challenging. The Transform Filter AppWizard, which creates a transform filter project, is used. The student is guided through the process of developing and coding a lowpass filter using DSP techniques. The student completes the filter development, and then constructs a filter graph which uses the filter in order to demonstrate it. As before, the student is required to turn in all project files, the input and output MP3 files, and the filter graph file.

Other Experiments

At the present time only the two DirectX experiments described here have been developed, but others will be developed in the future. The other experiments in the course require the use of Matlab, and cover signal generation, convolution, and FIR and IIR filter design.

Conclusion

This approach was successfully used in the "Realtime DSP" course for third-year ECET students during the Spring, 2004 term at IPFW. Many of the students are "nontraditional" and cannot always make it to class, and many of the traditional students have other courses which conflict with the lab portion of the DSP course. This has made it necessary to teach the lab portion of the course as if it were an online course. The key to doing this successfully has been to provide tools which the students can use on their own computers, and to provide clear and detailed lab instructions on the course website. The students were able to complete both experiments on their own with very few problems. Many of the students reported that they enjoyed "experiencing" DSP by using it to alter sound files of their own choosing, and one student reported developing an interest in multimedia programming as a result.

References

1. Keith Bromley, Bob Dukelow, and Jerry Symanski, Signal Processing Using Microsoft Windows, SPAWAR Systems Center, San Diego, CA.

2. Keith Bromley, Bob Dukelow, and Jerry Symanski, "The Acoustic Analysis Workbench," Department of Defense High Performance Computing Modernization Program User's Group Conference 2001, Biloxi, MS, June 18-21, 2001, http://www.hpcmo.hpc.mil/Htdocs/UGC/UGC01/paper/keith_bromley_paper.pdf

3. Kaiser, A. R., & Svirsky, M. A. (2000). "Using a personal computer to perform real-time signal processing in cochlear implant research." Proceedings of the IXth IEEE-DSP Workshop. October 15-18, 2000. Hunt, TX. Also in http://spib.ece.rice.edu/SPTM/DSP2000/submission/DSP/papers/paper123/paper123.pdf

4. Paul Kendrick, Real time Synthesis of a Plucked Guitar String Using Physical Modeling Techniques, (MSc thesis) University of Salford, Salford, Greater Manchester, U.K. http://www.kenders.net/projects/Physical/MSc_thesis.pdf

5. Gerald Youngblood, AC5OG, "A Software-Defined Radio for the Masses, Part 1," QEX, July/Aug 2002, pp. 13-21., http://www.flex-radio.com/articles_files/SDRFMP1.pdf

6. Gerald Youngblood, AC5OG, "A Software-Defined Radio for the Masses, Part 2," QEX, Sept/Oct 2002, pp. 10-18., http://www.flex-radio.com/articles_files/SDRFMP2.pdf

7. "DIY Plug-Ins", Computer Music, http://www.computermusic.co.uk/tutorial/diy1/diy1.asp

8. "DirectX 9.0 for C++", MSDN Library Archive, Microsoft, Inc., http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c/directx/htm/introductiontodirectshowfilterdevelopment.asp

9. "Microsoft Saves Helicopter from Disaster", Electronic Software Publishing Corporation (Elsop), San Jose, Ca. http://www.elsop.com/wrc/humor/ms_helic.htm

10. Peter E. Goodmann, "Lab Exercise 4: Introduction to Realtime DSP using DirectX", http://www.ecet.ipfw.edu/~goodmanp/courses/ECET357/web_objects/documents/LabEx4.doc

11. The AppWizards described in this paper may be downloaded at: http://www.ecet.ipfw.edu/~goodmanp/courses/ECET357/DirectX.htm

12. Peter E. Goodmann, "Lab Exercise 8: Convolution with DirectX", http://www.ecet.ipfw.edu/~goodmanp/courses/ECET357/web_objects/documents/LabEx8.doc

PETER E. GOODMANN, P.E. is an assistant professor of Electrical and Computer Engineering Technology at IPFW. He earned his BS degree in Electrical Engineering from Rose-Hulman Institute of Technology and his MS degree in Electrical Engineering from Purdue University. He has worked for 25 years in industry and education, and is a member of the IEEE and the ASEE.