UNIVERSITY OF CALIFORNIA

Santa Barbara

SSUM: Signals and Systems Using MATLAB

An Effective Application for Teaching Media Signal Processing to Artists and Engineers

A project submitted in partial satisfaction of the requirements for the degree

Master of Science in Graduate Media Arts & Technology

by

Bob L. Sturm

Committee in charge:

Professor Jerry Gibson, Chair

Professor Curtis Roads

Professor Stephen Pope

June 2004

The project of Bob L. Sturm is approved.

Dr. Curtis Roads

Dr. Stephen Pope

Dr. Jerry D. Gibson, Committee Chairperson

June 2004

Dedicated to Carla Marlene Townsend, soon to be Sturm!

CURRICULUM VITA OF BOB L. STURM
June 2004

Education

Bachelor of Arts: Physics
1994 — 1998, *University of Colorado,* Boulder, CO
Master of Arts: Music, Science, and Technology
1998 — 1999, *Stanford University,* Stanford, CA
Master of Science: Multimedia Engineering (expected)
2002 — 2004, *University of California,* Santa Barbara, CA

Professional Employment

Apr 2000 — Sept 2000: Sonification Research and Development University of Limerick, Ireland
January 2001 — August 2002: Music Director/Band Conductor, Fern Street Marching Band, San Diego
April 2001 — August 2002: Scientific Programmer/Analyst II, Scripps Institution of Oceanography, La Jolla
January 2003 — June 2004: Teaching assistant, Department of Physics, and Media Arts and Technology, University of California, Santa Barbara
June 2003 — June 2004: MATLAB programmer, Dr. Jerry Gibson, Professor of Engineering, Media Arts and Technology, University of California, Santa Barbara

Selected Publications

"Pulse of an Ocean: Sonification of Ocean Buoy Spectral Data"

Submitted for publication in *Leonardo Journal of Arts and Sciences*.

"Spectral Characteristics of the Musical Iced Tea Can"

Proceedings of the International Computer Music Conference, Miami, Florida. November 2004.

"Composing for an Ensemble of Atoms: The Metamorphosis of Scientific Experiment into Music"

Organised Sound, Vol 6, No. 2, Fall 2001. Cambridge University Press.

Awards

Second Place, Sherill C. Corwin—Metropolitan Theatres Awards for Excellence in Composition 2003-2004 for "Pacific Pulse":

Recordings

"Music from the Ocean" CD

Composerscientistrecordings, 2002.

"100:200111 torrey pines outer buoy" (track)

Lowercase 2.0 Compilation CD, 2002. Bremsstrahlung Recordings.

"50 Particles in a Three-Dimensional Harmonic Potential: An Experiment in 5 Movements" (track)

Organised Sound CD, Vol 7, No. 1, Spring 2002. Cambridge University Press.

ABSTRACT

SSUM: Signals and Systems Using MATLAB: An Effective Application for Teaching Media Signal Processing to Artists and Engineers

by

Bob L. Sturm

A problem exists in many media arts programs of how to effectively teach students with little mathematical practice the principles of media signal processing (MSP). For these students blackboard lectures and elementary engineering textbooks lead to consternation and apathy; such a course can become more of a struggling math class than anything else. This robs the student of the unique opportunity to learn, explore and apply MSP, an inherently multimedia field. To address this challenge, I have created a large set of exploratory demonstrations and applications programmed in MATLAB to teach principles and applications of MSP using multimedia. “Signals and Systems Using MATLAB (SSUM),” can supplement any course concerned with these topics. It provides an effective way to illustrate the essential concepts. SSUM is presented here, and its use in a course designed to teach MSP to media arts students is discussed. SSUM can be obtained for free from http://www.mat.ucsb.edu/~b.sturm.

List of Abbreviations

201A	MAT course: media signal processing using MATLAB
CBE	computer-based education
DSP	digital signal processing
FIR	finite impulse response
GUI	graphical user interface
GUIDE	graphical user interface development environment
IIR	infinite impulse response
LPC	linear predictive coding
MAD	MATLAB Auditory Demonstrations
MAT	Graduate Program in Media Arts and Technology
MSP	media signal processing
SSUM	Signals and Systems Using MATLAB
STFT	short-time Fourier transform
UCSB	University of California, Santa Barbara

Table of Contents

Introduction

There is no doubt that learning media signal processing (MSP) should be a required portion of any media arts program; students should at least understand the algorithms behind the software they use, the specifications of their hardware, and be able to effectively communicate with engineers. To begin to understand these things however, a student must possess an ability and confidence in mathematics beyond what most media arts students have. This creates the difficult problem of effectively teaching MSP to students who do not satisfy prerequisites that even most freshmen engineering students do. The question “what should be taught,” becomes “what can be taught?” Further magnifying this problem is the inherent heterogeneity of media arts students. The numerous backgrounds and abilities require numerous ways of saying the same thing. Some students may be more comfortable with sound than with images; some students may be adept at programming but not math.

Without employing mathematics more complex than algebra, a class of MSP can become dull and pedantic. At a level just above comfort the students can only be taught to add or multiply phasors, convolve two short signals on paper, or derive the magnitude response of a linear constant-coefficient feed-forward system. Without more complex math they can forget about learning analysis and synthesis, non-linear systems, and signal compression. But how beneficial is it to just teach a student in the media arts these mechanical and undemonstrative skills? Is the student any closer in the end to understanding the specifications of a microphone, or digital camera?

As expected, when teaching using the blackboard and assigning written homework, most of the students develop frustration, apathy, and quickly learn the minimum motions necessary to slide by. Instead of finding creative applications for the concepts they are learning, students spend most of their time working on elementary problems, such as adding phasors. By the end a student may be able to do convolution on paper, but has little knowledge of how it can, or even why it should, be applied.

There are three things working against a successful syllabus for teaching MSP without advanced mathematics. First and foremost, to do anything interesting requires mathematics. The course then becomes more concerned with complex numbers and trigonometry than MSP. Second, the pace of the course is compromised by the need to address the mathematics. Even after ten weeks, the interesting topics can still be weeks away. Finally, since MSP includes sound and image, the number of topics that can be addressed greatly increases. Even though the theory behind filtering audio is the same as filtering visuals, there is a huge leap to understand spatial frequencies from the time domain. If among the pursuits of media arts programs is the aim to imbue artists with practical digital media engineering knowledge, then a more effective and ultimately useful presentation of MSP needs to be employed.

There are several published texts that attempt to make concepts of MSP accessible. Most of these texts approach the topics using examples of sound, image and video ([1], [2], [3], [4], [5]). These texts however are either still too advanced for someone with little math, or too general to be interesting to the artist. *DSP First: A Multimedia Approach* [1] is perhaps the best text, and attempts to make the material more accessible by including a CD-ROM that has tutorials, movies, and MATLAB [6] demonstrations. The laboratories and movies included on the CD-ROM are nice, but they are not of much interest to artists; the five MATLAB demonstrations included are neither interesting nor inspiring. Books like [4] and [5] are great for students interested in sound, but for topics of images and video they have no content. Research into other approaches of teaching MSP has revealed an active field of technological pedagogy.

Clausen and Spanias describe the creation and use of an on-line digital signal processing (DSP) laboratory programmed in Java [7]. This application is used to present visualizations and interactive demonstrations to university students. Radke and Kulkarni have designed a similar application for their DSP lab, but programmed in MATLAB [8]. Rahkila and Karjalainen describe the benefit of computer-based education (CBE) for teaching DSP by virtue of it being multimedia [9]. Illustrating complex functions like filtering by actually applying it to a sound and hearing its effects can create more thorough comprehension and a longer-lasting impression than just deriving its frequency response on the blackboard. Their application is programmed in LiSP and served over a network to Macintosh computers [10]. In addition to applications created for visualizing DSP concepts, courses have been designed around exercises in programming.

The text *DSP First* [1] includes exercises to be done in MATLAB that demonstrate key concepts. Joaquim et al. describe an engineering course based on MATLAB to provide “a fast and natural way to DSP concepts and practical applications;” almost immediately students can begin creating and understanding DSP concepts and algorithms [11]. Many other engineering DSP courses use programming exercises, and some even program DSP hardware, but typically the prerequisites for these courses are too advanced for students in media arts.

It is clear that since many students in media arts programs are unique and individual—visual artists, sound composers and designers, and multimedia engineers—what is required is “hand-crafted course-ware” [12] that addresses the multimedia nature of MSP, and takes advantage of the creative motivation inherent to these students. Artists are curious, and by being enrolled in a media arts program, they should be interested in applying technology toward creative ends.

To alleviate the difficulties of teaching MSP to students who do not possess a penchant for mathematics beyond algebra, I have used MATLAB to create a large collection of exploratory demonstrations and applications designed to motivate and inspire students to learn and apply concepts of MSP. Its main goal is to provide a set of effective and interesting programs that can be used during lectures and on the student’s own time. These include illustrations of basic principles, to full applications that create sounds and images.

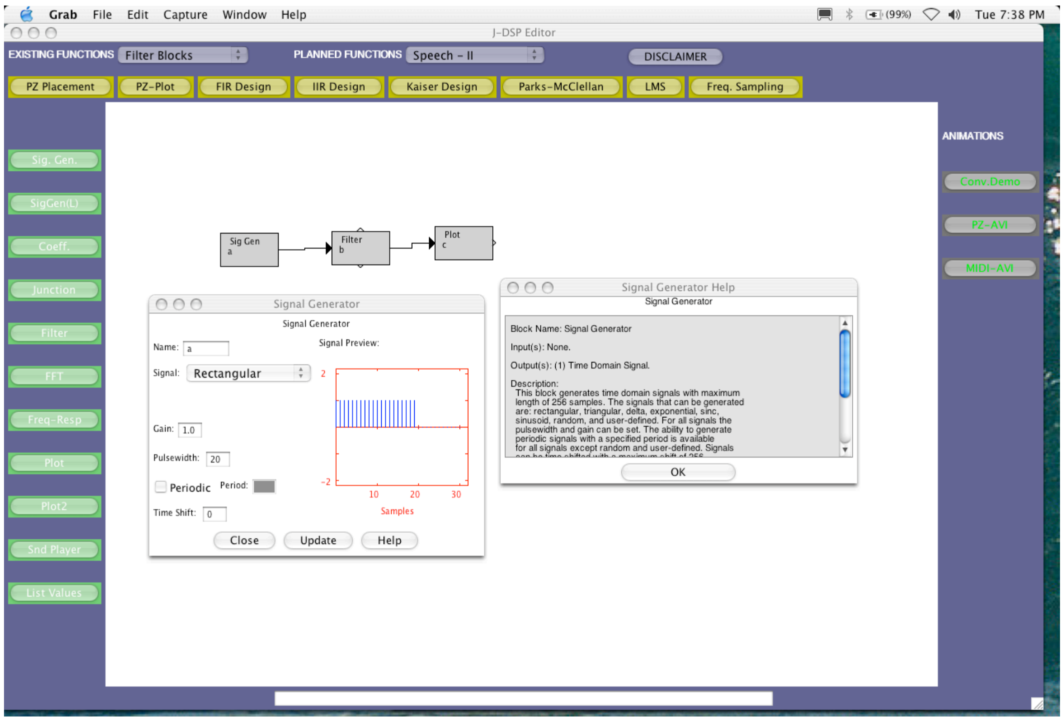
Exploratory demonstrations extend past traditional demonstrations. They demonstrate numerous concepts while retaining flexibility, such as allowing several parameters to be altered, and saving the output. The viewer thus becomes active rather than passive in learning the concept. As long as the user understands the parameters they are changing, the learning experience can be very effective. Each program has a help page that directs users.

Within this thesis I present this suite of applications and review its use for a MSP class taught to graduate students in a media arts program. It is hoped that by speaking to their interests and showing the artistic usefulness of the concepts, the students will be more likely to accept the learning curve and apply themselves.

Choice of MATLAB as the development software

There are several goals for the development of a suite of effective exploratory demonstrations and applications illustrating MSP. First, the concepts must be presented in a clear way with little interfering information. For instance, when explaining Fourier series one doesn’t want to have to also explain the effects of sampling and interpolation. Second the applications should be direct, flexible, and fast. The application should be presented in a unifying framework such as a graphical user interface (GUI), should allow variables to be changed, and shouldn’t be slow to give results. Third, images and audio must be used to demonstrate concepts. Multimedia in DSP education has been shown to increase comprehension and interactivity [9]. Fourth, the demonstrations should leave plenty to explore; they should satisfy a student’s curiosity in addition to a lecturer’s needs. Fifth, a student should be able to look into the code to understand how it works, and perhaps borrow from it. It is important to demystify the concepts and their implementation. Sixth, the demonstrations should be multiplatform. And finally, the cost to students to be able to run the applications on their own computer should be minimal.

There are a several low-level programming languages that can be used to demonstrate the application of DSP, such as C++, and Java. Clausen and Spanias [7] have created the Java Digital Signal Processing Editor (J-DSP) [13], which lets the users construct block diagrams of signal flow (Figure 1). It is a virtual lab space where a student assembles different applications using function blocks. Written in JAVA, J-DSP is multiplatform and free to anyone, but the code is inaccessible. The great number of choices and the patchwork environment is bewildering to someone who doesn’t first understand the concepts. But once a student has a grasp on the fundamentals of systems this package can become very beneficial.



Name: a

Signal: Rectangular

Gain: 1.0

Pulsewidth: 20

Periodic Period:

Time Shift: 0

Signal Generator

Block Name: Signal Generator

Input(s): None.

Output(s): (1) Time Domain Signal.

Description:
This block generates time domain signals with maximum length of 256 samples. The signals that can be generated are: rectangular, triangular, delta, exponential, sinc, sinusoid, random, and user-defined. For all signals the pulsewidth and gain can be set. The ability to generate periodic signals with a specified period is available for all signals except random and user-defined. Signals

Figure 1: Screenshot of the JAVA DSP Editor

Sound processing languages SuperCollider [14] or the graphical programming applications pd [15], and Max/MSP [16], can also be used to create interesting demonstrations—though only for sound. Though these have excellent real-time capability, they have marginal abilities for visual data display. Showing something as simple as sampling would be difficult. SuperCollider and pd are free, but Max/MSP costs over $200 for students.

There are several high-level software packages that can be used to teach signal processing, such as Simulink [17], Mathematica [18], Maple [19], Octave [20], Labview [21], and MATLAB [6]. A good overview of these and other packages in terms of engineering education can be found in [23]. Mathematica and Maple are expensive programs meant more for solving symbolic math than creating applications. Though they have good visualization abilities, they don't easily handle external data like sound and image.

Simulink, produced by the makers of MATLAB, is like J-DSP in that it is a patchwork environment. Figure 2 shows an example Simulink application that demonstrates audio reverberation using a delay line and feedback. Currently the interface is clunky and its responsiveness is slow, but it does show promise for visualizing systems.

Similar to Simulink, LabVIEW uses a graphical environment for designing systems (Figure 3), but it is at the high-end of engineering software with its data-acquisition capabilities and external instrument control. Being such it is an extremely complex and expensive package.

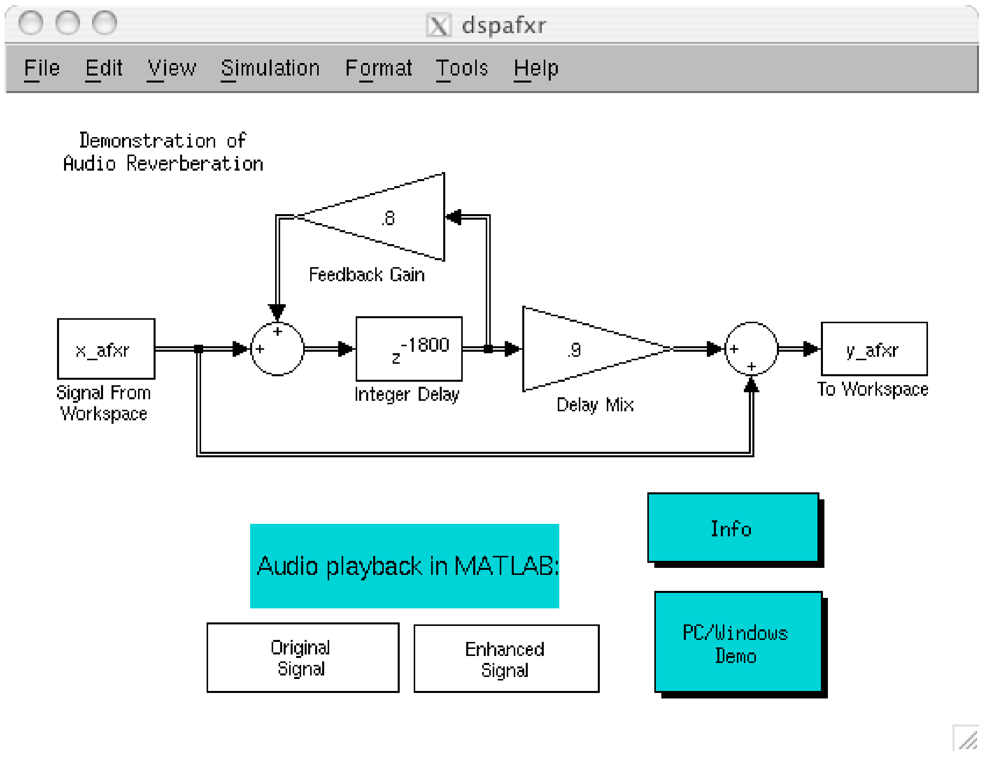


Figure 2: A SIMULINK demonstration of reverberation

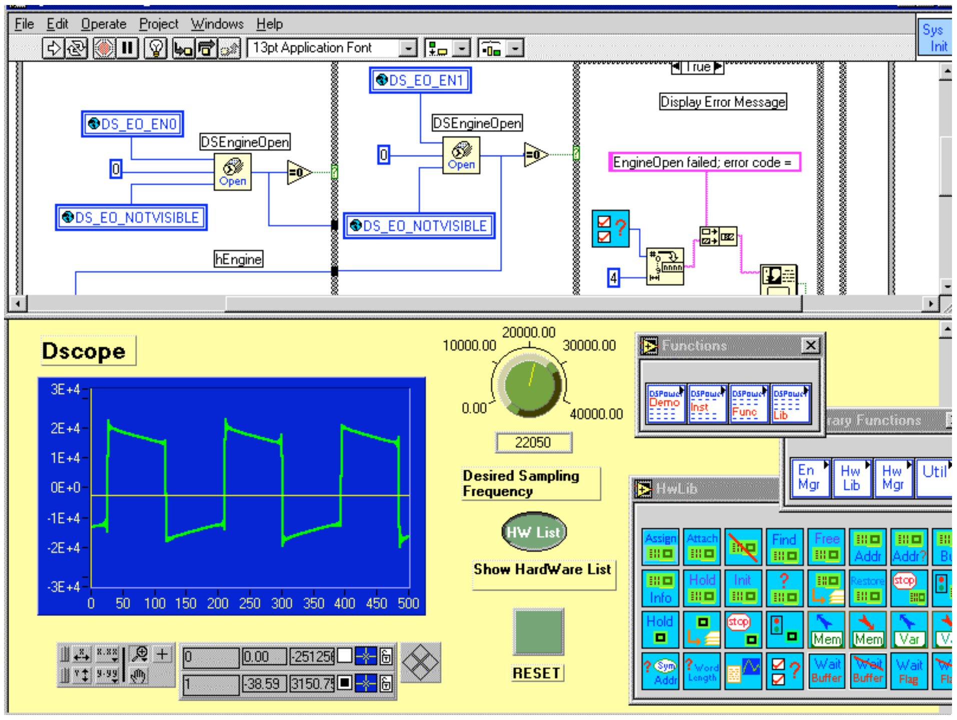


Figure 3: LabVIEW Screenshot (from http://www.signalogic.com)

Octave, a free open-source and multi-platform mathematics programming environment, is also available. It is mostly compatible with MATLAB, but a review of the currently implemented functions shows a great lack of necessary signal processing and visualization routines. In addition there is no easy way to create GUIs. These must be created using platform dependent graphics libraries.

MATLAB provides an integrated development environment that is easy to use and understand, and cheap for students.[1] MATLAB is platform independent, has superior graphics handling and visualization capabilities, and has a great GUI development environment for wrapping applications. It has an extensive library of routines, and “toolboxes” can be purchased to add specialized functionality, such as advanced signal or image processing

[1] At the time of this writing MATLAB 6.5 costs $99 for students.

routines.[2] Applications written in MATLAB are open; any user can look at the code. Furthermore countless institutions, both academic and corporate, as well as many independent users worldwide,[3] use MATLAB for algorithm development, prototyping, complex modeling and problem solving. For these reasons it is clear that MATLAB is the best choice for developing applications that satisfy the seven criteria above.

Though there should be some familiarity with vectors and matrixes, the MATLAB programming language is easy to learn and intuitive. It is an interpreted language rather than a compiled one, unlike C++ and JAVA. Commands can be typed in and feedback is given immediately. The drawback to using MATLAB however is its lack of real-time functions, like tracking a sound as it plays, or visualizing a spectrogram straight from the sound input—as can be done by the software "Baudline" [24].

There are excellent examples of multimedia pedagogical applications written using MATLAB. The "MATLAB Auditory Demonstrations" (MAD) is perhaps the best and most relevant to signal processing. Created by Cooke et al., MAD provides a large suite of interactive demonstrations for exploring psychoacoustics and concepts of auditory perception [25, 26]. It was created to provide a multi-modal interactive environment for teaching speech and hearing. In the past this has usually been done using passive demonstrations such as audio compact discs [27]. With MAD a teacher can quickly demonstrate, for instance, the effect of interrupted speech, or window size on frequency resolution (Figure 4).

---

[2] At the time of this writing MATLAB toolboxes cost $30 each for students.
[3] A large on-line MATLAB user group can be found at MATLAB Central: http://www.mathworks.com/matlabcentral/

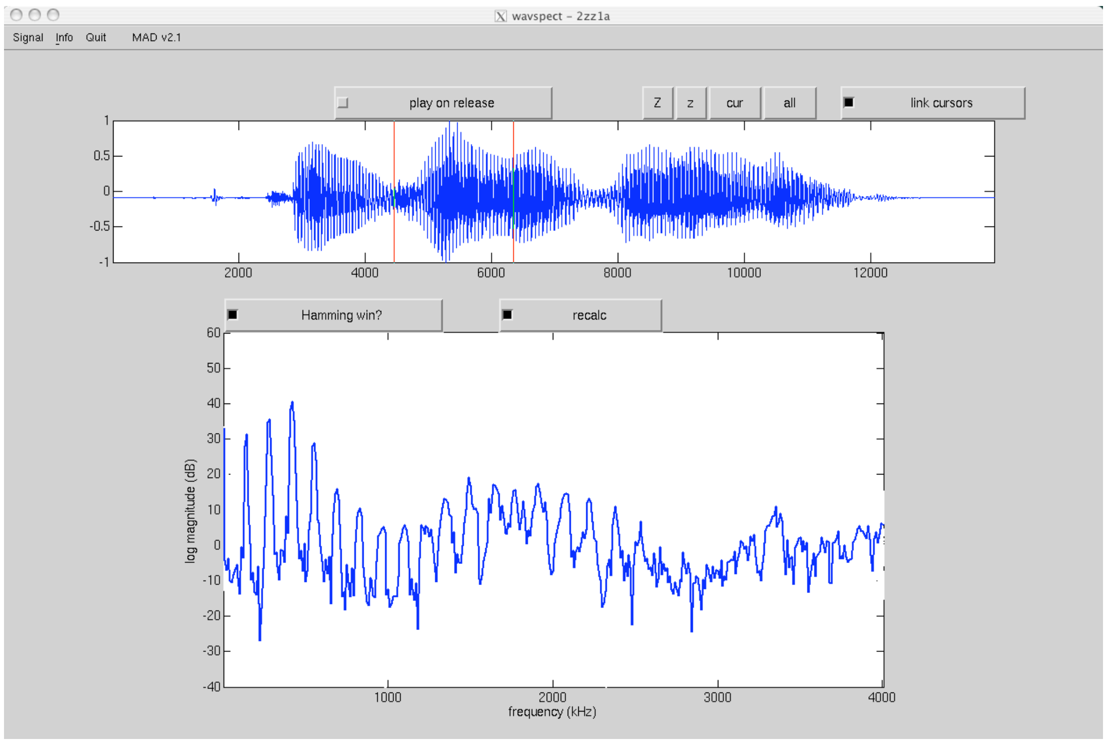


Figure 4: MAD program wavspect

Using MAD as a model, I developed “Signals and Systems Using MATLAB” (SSUM) to aid in the teaching of MSP, specifically to media arts students at the graduate Media Arts and Technology (MAT) program at the University of California, Santa Barbara (UCSB).

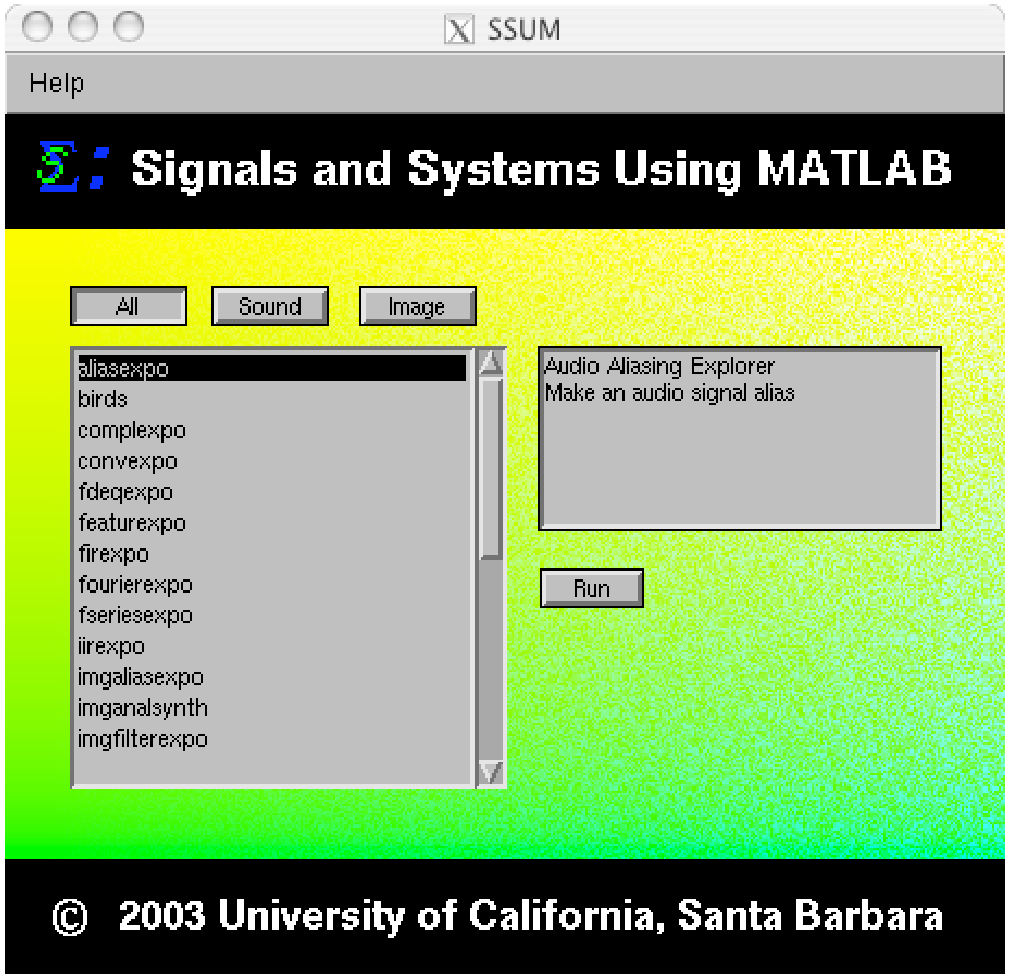
SSUM: Signals and Systems Using MATLAB

SSUM is a suite of exploratory demonstrations and applications programmed in MATLAB. These programs are designed specifically to entice and inspire students who do not yet possess the mathematical knowledge necessary for thorough research in MSP. To use SSUM MATLAB must be installed as well as its signal processing toolbox. The cost of this software to the student is a bit more than a good engineering text, but it is hoped that after becoming acquainted with the power of MATLAB, the student will continue to use it to work with data, develop algorithms, and apply it to their creative work.[4]

SSUM demonstrates essential principles and concepts of MSP without requiring rigorous mathematics; exploration and learning is done first using software rather than paper. SSUM currently has 31 exploratory demonstrations and applications illustrating concepts of waveforms, modulation, sampling and interpolation, aliasing, the time and frequency domains, finite difference equations, convolution, and filtering, pole-zero diagrams, analysis and synthesis, and signal statistics. Many of these are applied to sounds and images. There are also applications that demonstrate interesting topics such as sound cross-synthesis, additive synthesis of birdsong, and sine wave speech. SSUM is perfect for use in lectures, labs, homework, and creative work. All the programs in SSUM are wrapped in GUIs, so there is no need for typing commands at the prompt. Many of the applications are integrated as well. For instance, if one is creating a waveform in one application, it can be sent to another application for filtering, or to another to see its frequency domain representation.

[4] I have used MATLAB as computer music composition software for six years.

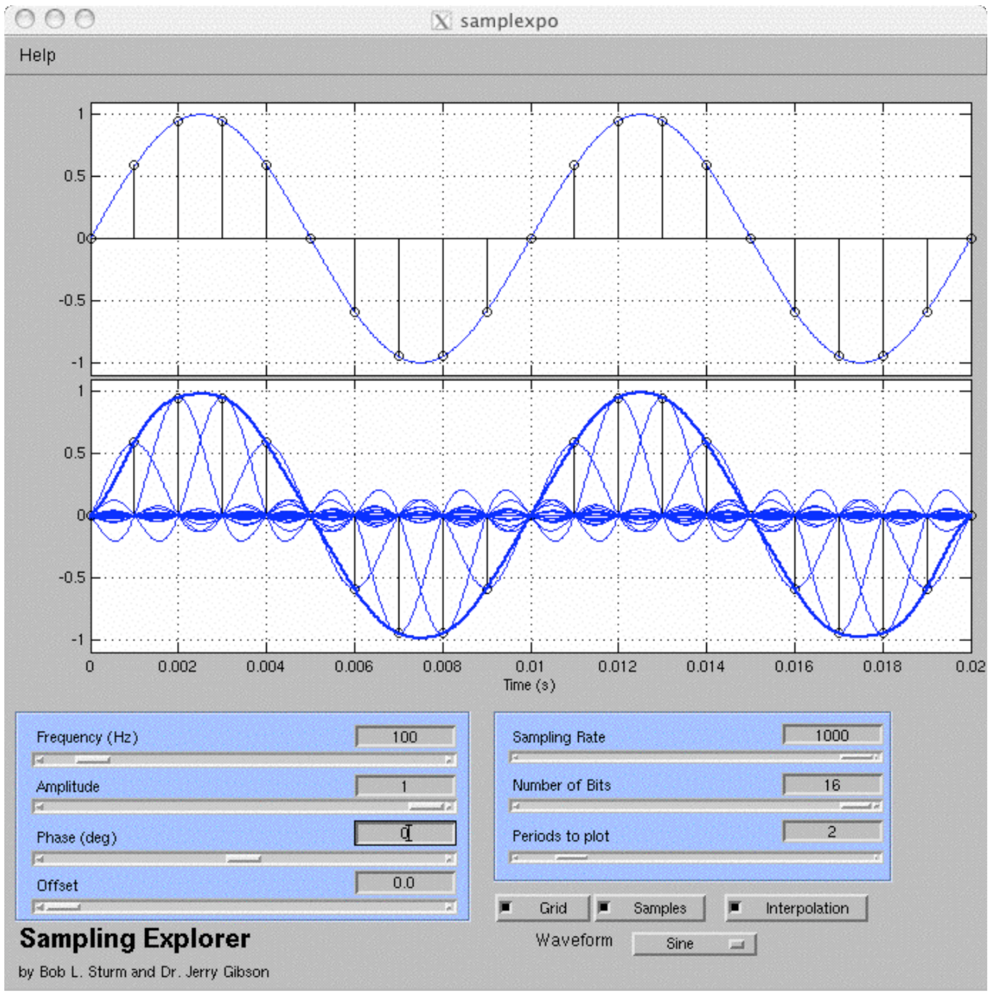
SSUM can be executed by typing `ssum` in the SSUM main directory from the MATLAB command window. The main application window will appear (Figure 5), from which you can run an application. Information on the selected program is displayed in the box to the right. The "Help" menu item will display a browser containing the main help page, which contains links to help pages for all available programs. The "All," "Sound," and "Image" buttons filter the list of relevant applications. An application can be executed by selecting its name and clicking "Run."



Audio Aliasing Explorer
Make an audio signal alias

© 2003 University of California, Santa Barbara

Figure 5: SSUM main application window



Frequency (Hz) 100
Amplitude 1
Phase (deg) 0
Offset 0.0
Sampling Rate 1000
Number of Bits 16
Periods to plot 2
Waveform Sine

Sampling Explorer

by Bob L. Sturm and Dr. Jerry Gibson

Figure 6: SSUM Sampling Explorer

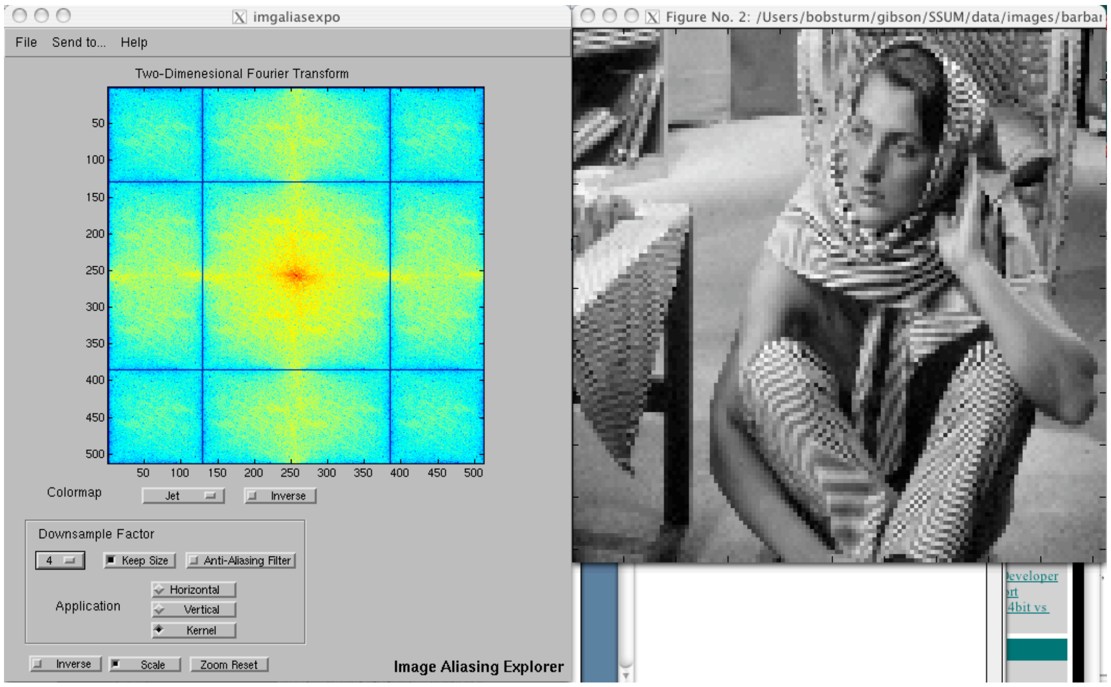
A few exemplary applications from SSUM will now be presented. These show the concepts behind the applications, such as use of sound, visuals, and sharing of data between programs. A complete presentation of all current programs is given in Appendix A: Current SSUM Applications.

Figure 6 shows *Sampling Explorer*, which demonstrates how continuous signals can be digitized. The top plot represents the continuous input signal and the position of the samples. The bottom plot shows the result of interpolating the samples back to a continuous signal. With the sliders and text boxes the user can change the input frequency, amplitude, phase, and offset, as well as the sampling rate and number of bits used to represent the signal. The input waveform can also be changed to sine, square, triangle, sawtooth, and a random wave. The plots can be altered by changing the number of periods to plot, turning on and off the grid, the lollipops marking the samples, and the interpolation.

Using *Sampling Explorer* one can investigate the cause and effect of aliasing, the effects of quantization, and how to turn digital signals into analog using ideal lowpass filtering, or sinc interpolation. Another apparent effect is the edge effects from the interpolation. The beginning and end of the interpolated waveform doesn't quite match the continuous signal. Thus in one program there exist several demonstrations of concepts that range from simple to complex!

Image Aliasing Explorer (Figure 7) allows experimentation with sampling images. This demonstration gives quick visual feedback about the relationship between pixels and spatial frequencies. The user first loads an image from the "File" menu, which is then displayed on the right. The program computes the two-dimensional Fourier transform and displays it in the main GUI window. The user can change the appearance of the transform using the two widgets directly below the plot. The appearance of the image can be changed using the menu items at the bottom of the GUI.

Once an image is loaded it can be downsampled with or without using an anti-aliasing filter. The downsample factor can be selected and applied in the horizontal or vertical directions, or using square blocks. From the File menu the image can be saved to an image file. From the "Send to…" menu the altered image can be sent to other programs, like *Image Filtering Explorer*, and *Image Spectrum Explorer*. The chosen program is started with the image as its application data.

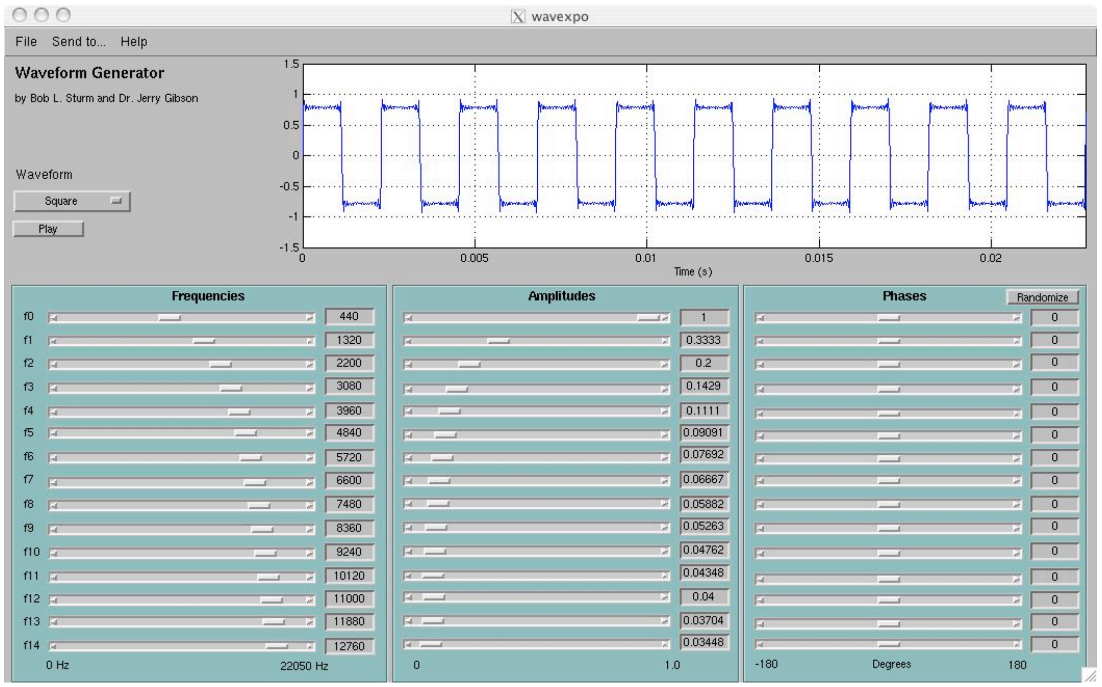

imgaliasexpo
File
Send to...
Help
Two-Dimenesional Fourier Transform
Figure No. 2: /Users/bobsturm/gibson/SSUM/data/images/barbar
Inverse
Scale
Zoom Reset
Image Aliasing Explorer

Colormap: Jet

Downsample Factor: 4 — Keep Size — Anti-Aliasing Filter

Application: Horizontal, Vertical, Kernel

Figure 7: SSUM Image Aliasing Explorer



Waveform: Square

	Frequencies	Amplitudes	Phases
f0	440	1	0
f1	1320	0.3333	0
f2	2200	0.2	0
f3	3080	0.1429	0
f4	3960	0.1111	0
f5	4840	0.09091	0
f6	5720	0.07692	0
f7	6600	0.06667	0
f8	7480	0.05882	0
f9	8360	0.05263	0
f10	9240	0.04762	0
f11	10120	0.04348	0
f12	11000	0.04	0
f13	11880	0.03704	0
f14	12760	0.03448	0
	0 Hz – 22050 Hz	0 – 1.0	-180 – 180 Degrees

Figure 8: SSUM Waveform Generator

Figure 8 shows *Waveform Explorer*, an application demonstrating the superposition of oscillators to create other waveforms. The user is able to adjust frequency, amplitude, and phase for fifteen oscillators, as well as select predefined waveforms like square or sawtooth. This way the student can begin to understand superposition and Fourier series. There is also the capability to hear the sound over speakers, and save it to a sound file. One interesting phenomena is changing the waveform by randomizing the phase. Even though the waveform looks completely different it sounds the same to our ears. A user can send a waveform created within this program to other programs, like *Sonogram Explorer*, *Fourier Explorer*, or *Aliasing Explorer*. This integration of tools within SSUM is important for giving the student flexible views of the same thing.

Fourier Explorer, shown in Figure 9, enables one to look at the Fourier spectrum of a sound. After loading a soundfile, the user can drag a window (vertical red bars) across the time-domain representation of a signal and watch the spectrum change. Changing the size and shape of the analysis window leads to different resolutions, demonstrating the time-frequency trade-off, and the effects of different windows on the transform.

The MAD program “wavspect” (Figure 4) [25] was used as a starting point to create this application, specifically its real-time updating of the display. What would be nice is for the sound to play while the spectrum changes, but there is currently no way in MATLAB of synchronizing the audio playback with the display.

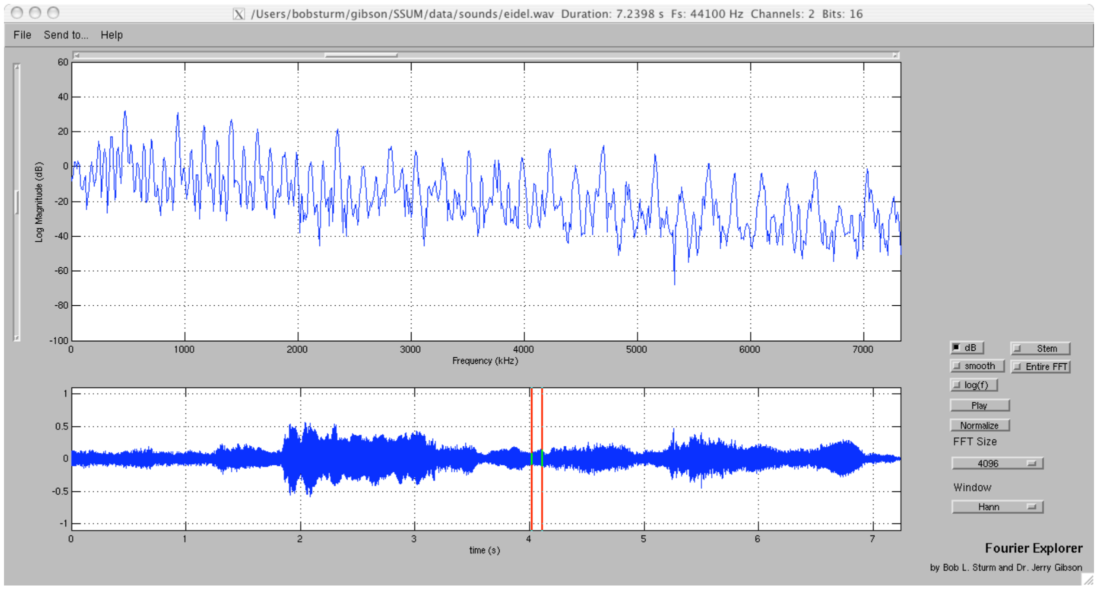

/Users/bobsturm/gibson/SSUM/data/sounds/eidel.wav Duration: 7.2398 s Fs: 44100 Hz Channels: 2 Bits: 16
File
Send to...
Help
Log Magnitude (dB)
Frequency (kHz)
time (s)
dB
Stem
smooth
Entire FFT
log(f)
Play
Normalize
FFT Size
4096
Window
Hann
Fourier Explorer
by Bob L. Sturm and Dr. Jerry Gibson

Figure 9: SSUM Fourier Explorer

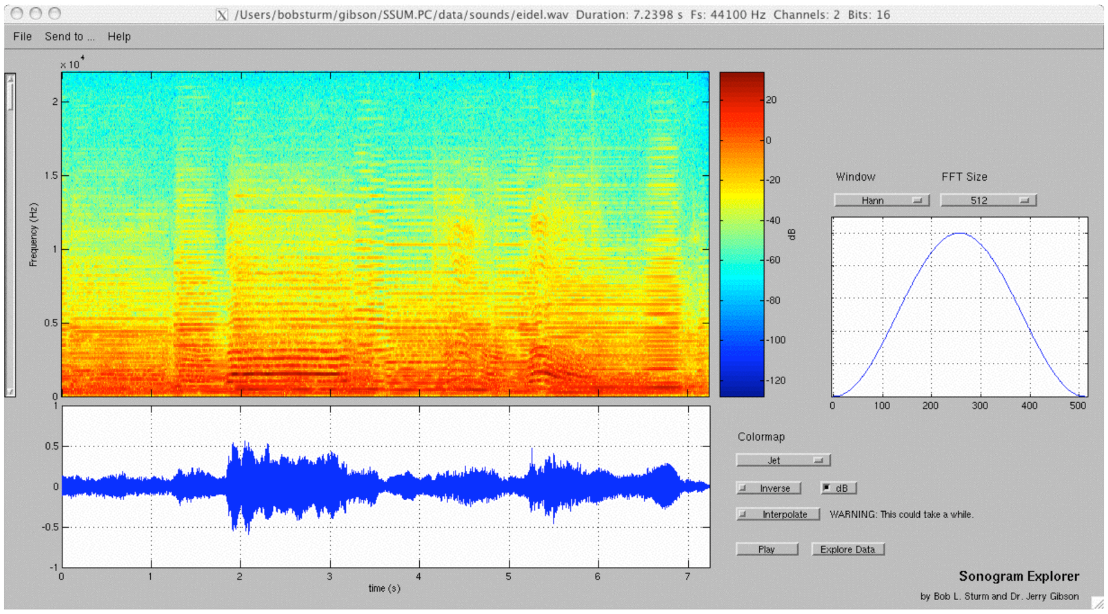
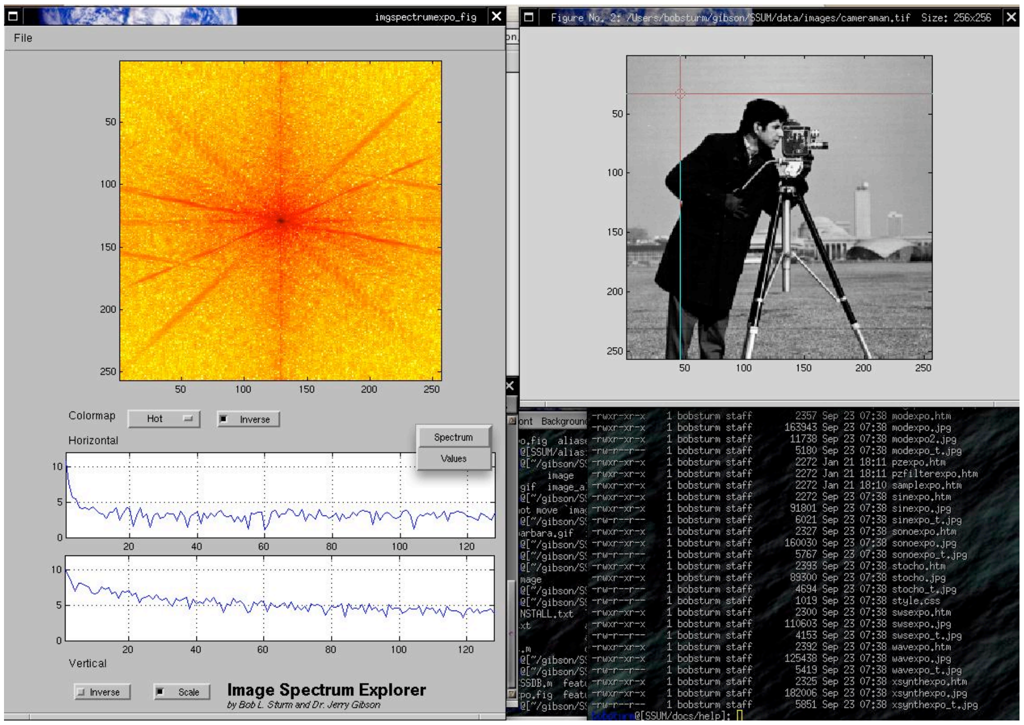

/Users/bobsturm/gibson/SSUM.PC/data/sounds/eidel.wav Duration: 7.2398 s Fs: 44100 Hz Channels: 2 Bits: 16
File
Send to ...
Help
Frequency (Hz)
dB
time (s)
Window
FFT Size
Hann
512
Colormap
Jet
Inverse
dB
Interpolate
WARNING: This could take a while.
Play
Explore Data
Sonogram Explorer
by Bob L. Sturm and Dr. Jerry Gibson

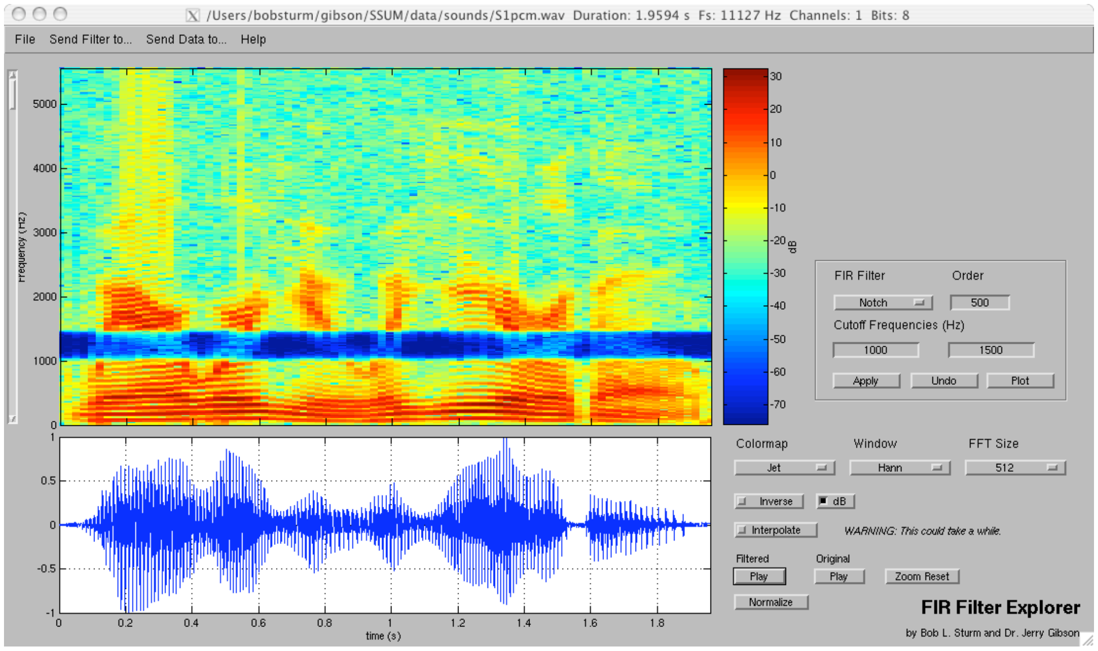
Figure 10: SSUM Sonogram Explorer

A similar application is *Sonogram Explorer* (Figure 10), which presents the user with the short-time Fourier transform (STFT) of sounds. As in many SSUM programs, the display can be changed with log amplitude scaling and color maps. The user can zoom in on parts of the sound, either in the STFT or the time-domain waveform, and both plots will update to reflect the new domain. Additionally the effects of window shapes and sizes on the STFT can be seen. The analysis window can be altered by the options on the right. An interesting use of this application is the "Explore Data" function. When clicked the user can click in the STFT and obtain frequency, magnitude, and timing data. Using this data a user can resynthesize a sound using additive synthesis with envelopes.



```
-rwxr-xr-x    1 bobsturm staff      2357 Sep 23 07:38 modexpo.htm
-rwxr-xr-x    1 bobsturm staff    163943 Sep 23 07:38 modexpo.jpg
-rwxr-xr-x    1 bobsturm staff     11738 Sep 23 07:38 modexpo2.jpg
-rw-r--r--    1 bobsturm staff      5180 Sep 23 07:38 modexpo_t.jpg
-rwxr-xr-x    1 bobsturm staff      2272 Jan 21 18:11 pzexpo.htm
-rwxr-xr-x    1 bobsturm staff      2272 Jan 21 18:11 pzfilterexpo.htm
-rwxr-xr-x    1 bobsturm staff      2272 Jan 21 18:10 samplexpo.htm
-rwxr-xr-x    1 bobsturm staff      2272 Sep 23 07:38 sinexpo.htm
-rwxr-xr-x    1 bobsturm staff     91801 Sep 23 07:38 sinexpo.jpg
-rw-r--r--    1 bobsturm staff      6021 Sep 23 07:38 sinexpo_t.jpg
-rwxr-xr-x    1 bobsturm staff      2327 Sep 23 07:38 sonoexpo.htm
-rwxr-xr-x    1 bobsturm staff    160030 Sep 23 07:38 sonoexpo.jpg
-rw-r--r--    1 bobsturm staff      5767 Sep 23 07:38 sonoexpo_t.jpg
-rwxr-xr-x    1 bobsturm staff      2393 Sep 23 07:38 stocho.htm
-rwxr-xr-x    1 bobsturm staff     89300 Sep 23 07:38 stocho.jpg
-rw-r--r--    1 bobsturm staff      4694 Sep 23 07:38 stocho_t.jpg
-rw-r--r--    1 bobsturm staff      1019 Sep 23 07:38 style.css
-rwxr-xr-x    1 bobsturm staff      2300 Sep 23 07:38 swsexpo.htm
-rwxr-xr-x    1 bobsturm staff    110603 Sep 23 07:38 swsexpo.jpg
-rw-r--r--    1 bobsturm staff      4153 Sep 23 07:38 swsexpo_t.jpg
-rwxr-xr-x    1 bobsturm staff      2392 Sep 23 07:38 wavexpo.htm
-rwxr-xr-x    1 bobsturm staff    125438 Sep 23 07:38 wavexpo.jpg
-rw-r--r--    1 bobsturm staff      5419 Sep 23 07:38 wavexpo_t.jpg
-rwxr-xr-x    1 bobsturm staff      2325 Sep 23 07:38 xsynthexpo.htm
-rwxr-xr-x    1 bobsturm staff    182006 Sep 23 07:38 xsynthexpo.jpg
-rw-r--r--    1 bobsturm staff      5851 Sep 23 07:38 xsynthexpo_t.jpg
bobsturm@[SSUM/docs/help]:
```

Figure 11: SSUM Image Spectrum Explorer



/Users/bobsturm/gibson/SSUM/data/sounds/S1pcm.wav Duration: 1.9594 s Fs: 11127 Hz Channels: 1 Bits: 8

FIR Filter: Notch
Order: 500
Cutoff Frequencies (Hz): 1000 1500
Colormap: Jet
Window: Hann
FFT Size: 512

Figure 12: SSUM FIR Filter Explorer

Image Spectrum Explorer (Figure 11) allows one to look at the spatial frequencies in images. Once an image is loaded its two-dimensional Fourier transform is displayed. On the image is a cross hair that can be moved around the image. This shows the row and column of pixels used to calculate horizontal and vertical spatial frequencies. As this sampling point is moved the two spectra change, shown in the bottom of the left window. The row and column pixel values can be plotted as well.

The sonogram is an intuitive way to show dynamic frequency content of signals over time. Many of the applications in SSUM take advantage of this. *FIR Filter Explorer*, shown in Figure 12, demonstrates the effects of filtering on the frequency content of a signal using the sonogram. Four different types of filters can be selected (lowpass, highpass, bandpass, notch). For each filter, the order and cutoff frequencies can be specified. The filter can be applied to a loaded sound and compared with the original. Clicking “Plot” will plot the frequency and impulse response of the filter. From the “Send Filter to…” menu, the filter design can be sent to other applications, like *Pole-Zero Explorer* (Figure 13) or *Pole-Zero Filter Explorer* (Figure 14), which will display the poles and zeros of the filter, and *Convolution Explorer* (Figure 15), which will convolve different signals with the filter’s impulse response. Similar to *FIR Filter Explorer* is *IIR Filter Explorer*.

Pole-Zero Explorer (Figure 13) allows the user to move poles and zeros around on the z-plane, and watch the magnitude, phase, and impulse response of the filter change. The “Load” menu item provides a set of filters approximating formants of speech vowel sounds. This program was created from the MAD program “polezero” [25]. Using the “Send to…” menu item the designed filter can be sent to *Pole-Zero Filter Explorer* (Figure 14) and applied to any sound. Within this application the poles and zeros can be moved around as well. The impulse response of the filter can also be sent to *Convolution Explorer* (Figure 15).

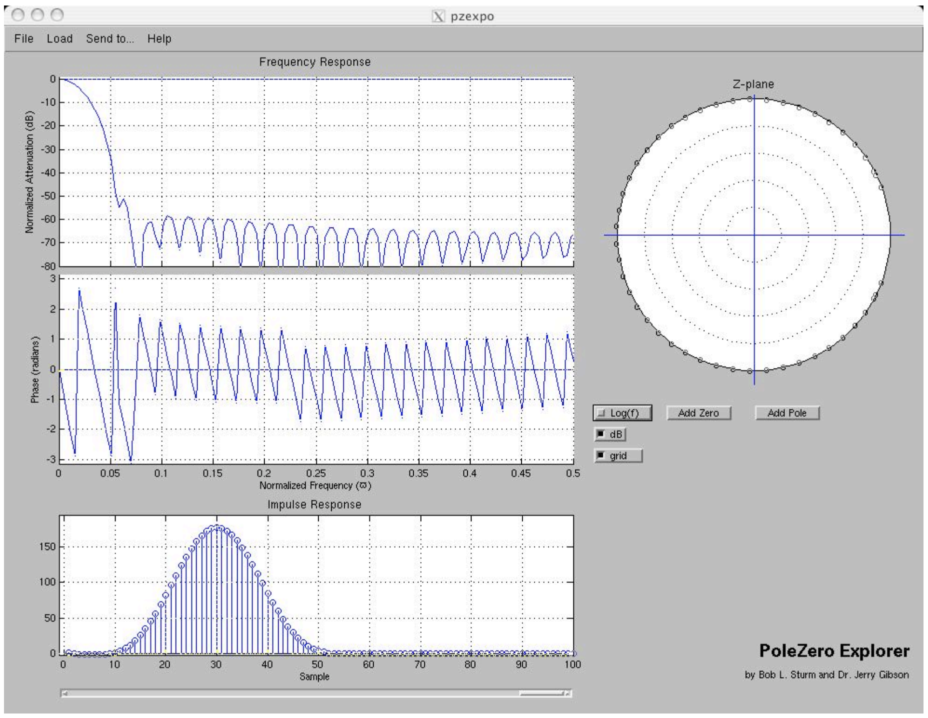
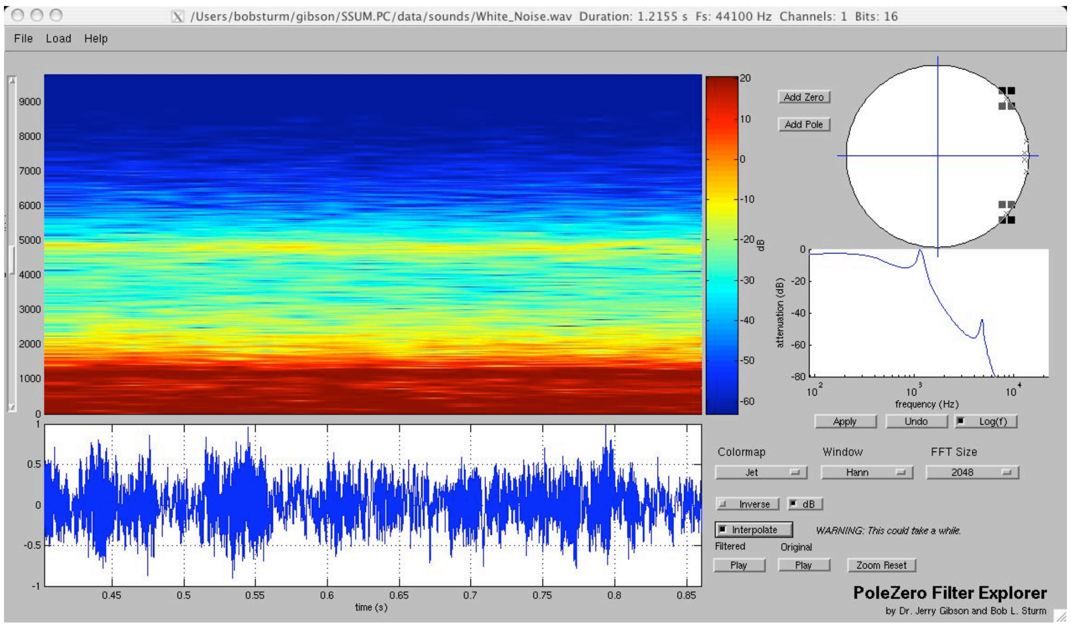


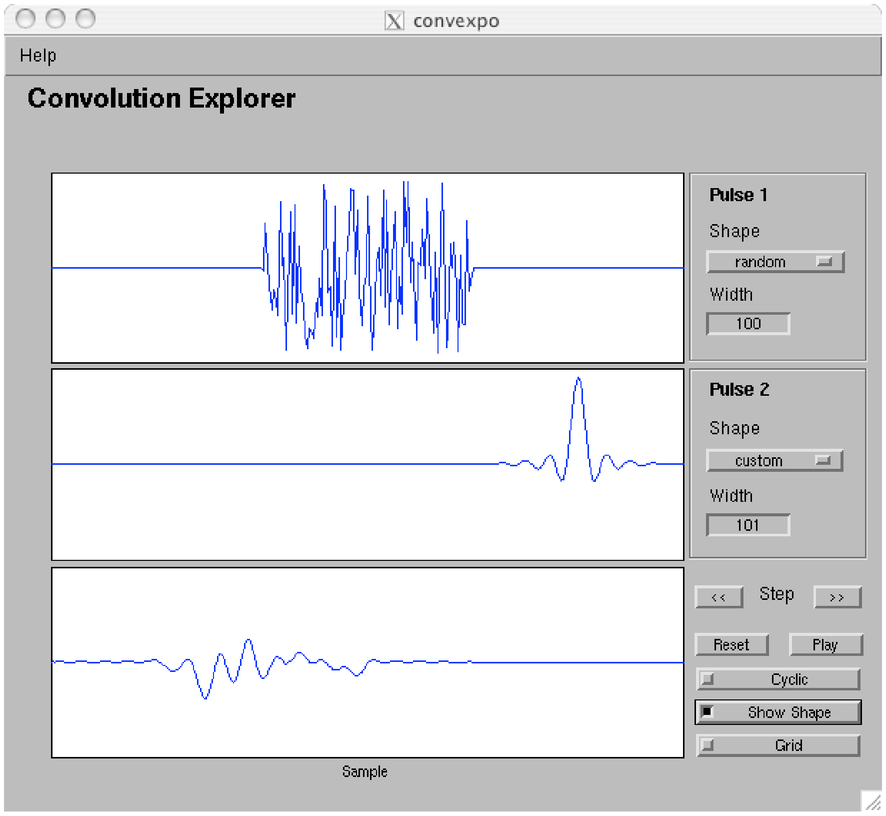
Figure 13: SSUM Pole-Zero Explorer


/Users/bobsturm/gibson/SSUM.PC/data/sounds/White_Noise.wav Duration: 1.2155 s Fs: 44100 Hz Channels: 1 Bits: 16

Colormap: Jet
Window: Hann
FFT Size: 2048

WARNING: This could take a while.

Figure 14: SSUM Pole-Zero Filter Explorer


convexpo

Convolution Explorer

Pulse 1
Shape: random
Width: 100

Pulse 2
Shape: custom
Width: 101

Figure 15: SSUM Convolution Explorer

Convolution Explorer (Figure 15) animates the convolution operation. Several signal shapes and lengths can be chosen to illustrate its various effects. Cyclic and non-cyclic convolution is available as well. As seen in Figure 15 a noisy signal is being convolved, step-by-step, with the impulse response of a lowpass filter. The result, shown in the bottom plot is the output.

Filtering images can be explored using *Image Filter Explorer* (Figure 16). Once an image is loaded, its two-dimensional Fourier transform is displayed. Several filters are available including the moving average, Gaussian, del Gaussian, and median filter. The spatial frequency response of each filter can be plotted, except for the non-linear ones. The filters can be applied to only the horizontal or vertical directions, or by blocks or kernels. Noise can be added to any image and its effects on filtering seen. Students find the median filter's ability to remove speckle noise startling.

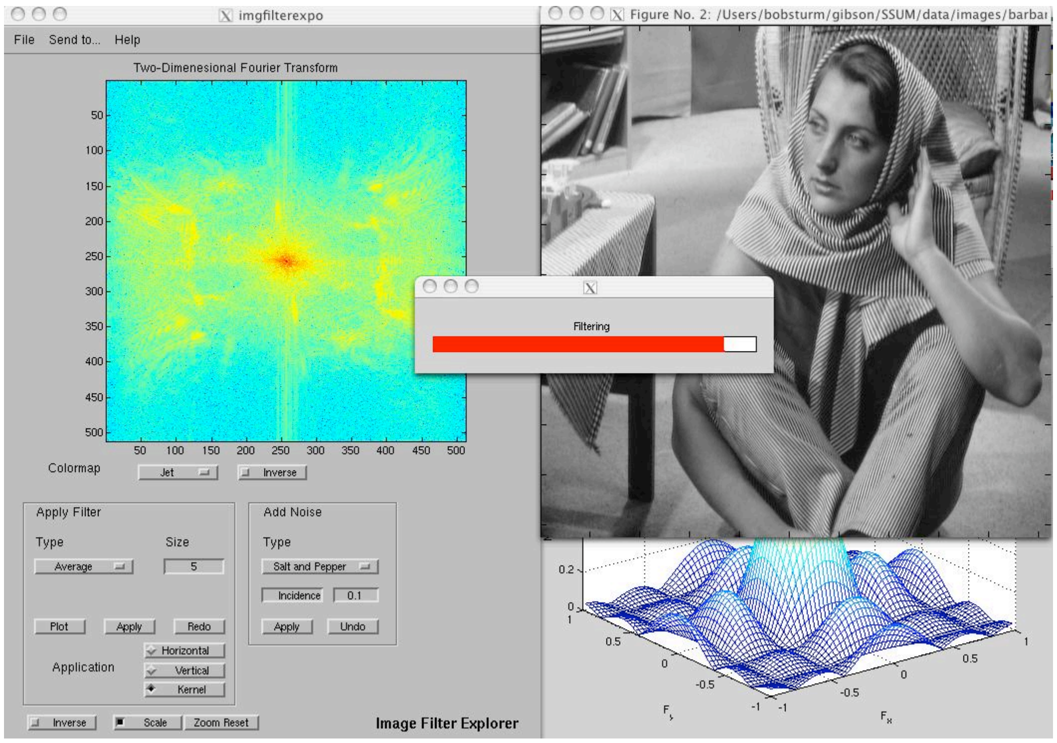


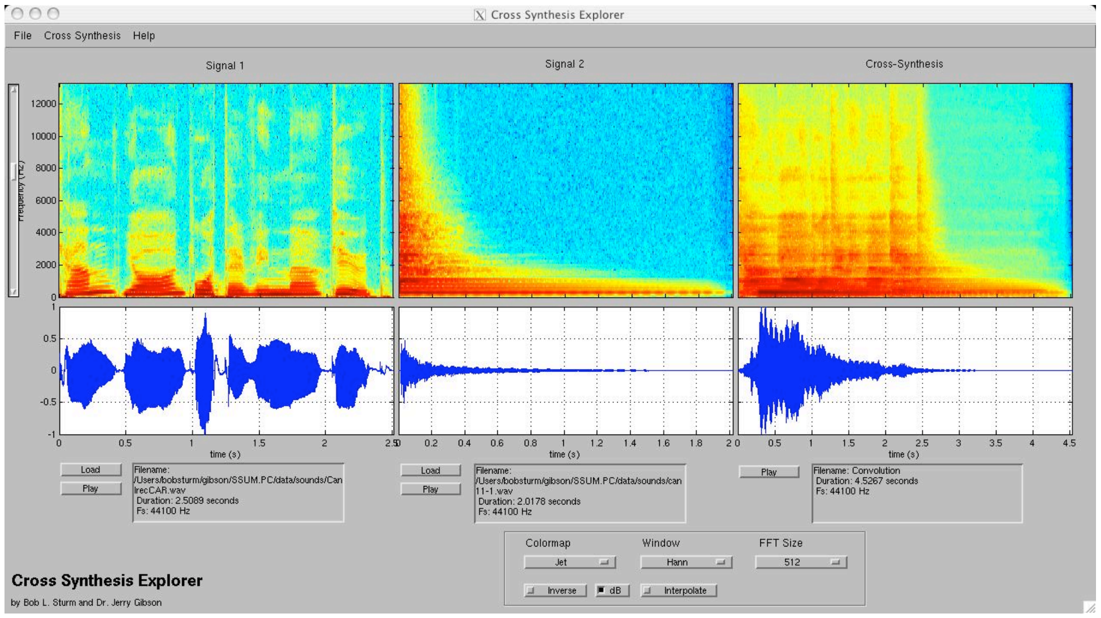
Figure 16: SSUM Image Filtering Explorer

Some applications in SSUM demonstrate curiosities of signal processing. While these don't directly demonstrate essential concepts of MSP, they do provide illuminating insights of their own. Figure 17 shows *Cross-Synthesis Explorer*. This application allows three different methods for cross-synthesizing sounds: convolution of the two sounds,[5] amplitude modulation of one signal by the other, and linear predictive coding (LPC)—using one sound as the model and the other as the source.

Once sounds are loaded, their time-domain waveforms and STFTs will be displayed. One of the three cross-synthesis routines can be executed, and when finished its waveform and STFT will be displayed. Students really enjoy this demonstration and begin to realize what convolution does; suddenly the mystery of digital reverberation disappears. They particularly enjoy hearing a gong or crow speak.

Another interesting topic is sinewave speech [22]. Using linear prediction a speech signal is reduced to a number of modulated oscillators that, together, remain intelligible to a startling degree. *Sinewave Speech Explorer* (Figure 18) demonstrates this curious phenomenon using four oscillators to approximate a given signal. Any sound can be loaded, and any combination of the four components can be heard. This demonstration was appropriated from the MAD sinewave speech program "sws" [25].

[5] The convolution routine uses multiplication in the frequency domain to decrease the latency.



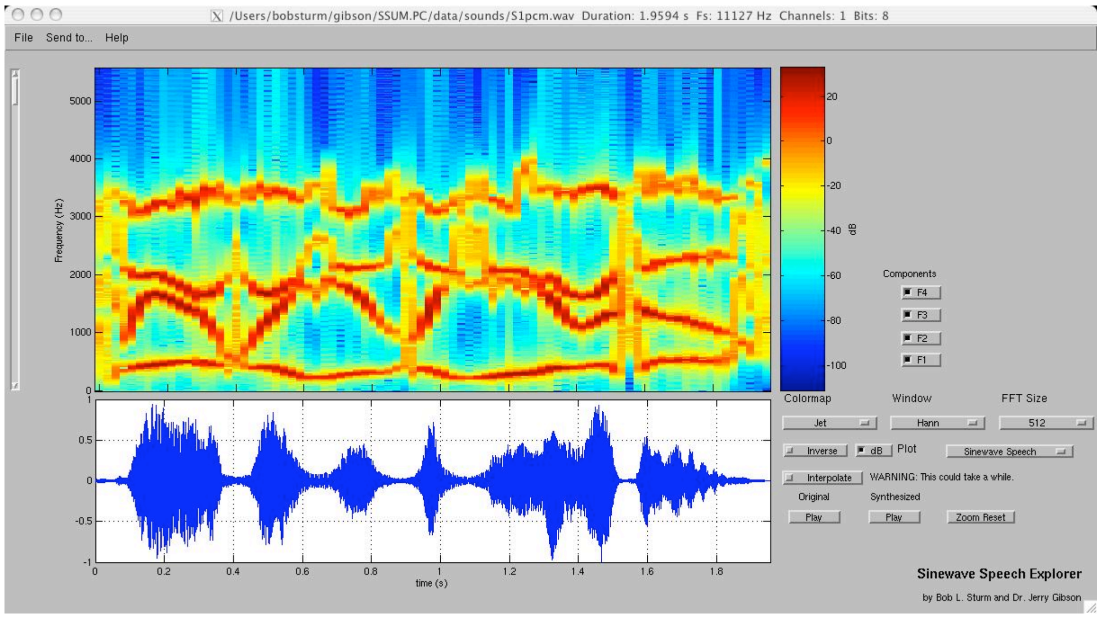
Filename:
/Users/bobsturm/gibson/SSUM.PC/data/sounds/CanIrecCAR.wav
Duration: 2.5089 seconds
Fs: 44100 Hz

Filename:
/Users/bobsturm/gibson/SSUM.PC/data/sounds/can11-1.wav
Duration: 2.0178 seconds
Fs: 44100 Hz

Filename: Convolution
Duration: 4.5267 seconds
Fs: 44100 Hz

Colormap: Jet
Window: Hann
FFT Size: 512

Figure 17: SSUM Cross-Synthesis Explorer

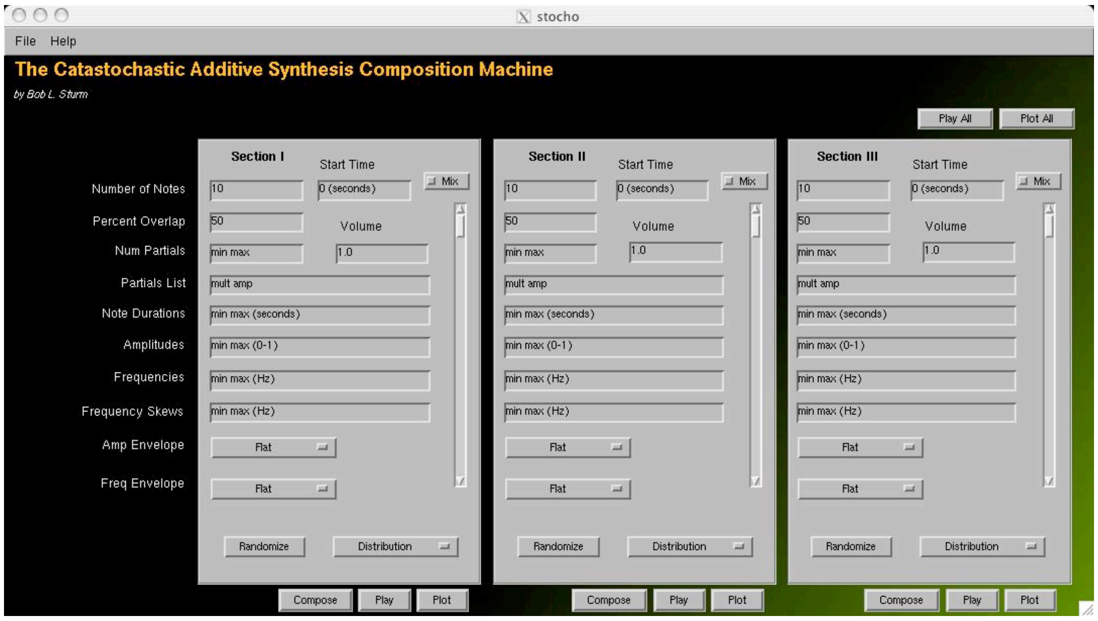


/Users/bobsturm/gibson/SSUM.PC/data/sounds/S1pcm.wav Duration: 1.9594 s Fs: 11127 Hz Channels: 1 Bits: 8

Colormap: Jet
Window: Hann
FFT Size: 512
Plot: Sinewave Speech

WARNING: This could take a while.

Figure 18: SSUM Sinewave Speech Synthesis Explorer



	Section I	Section II	Section III
Start Time	0 (seconds)	0 (seconds)	0 (seconds)
Volume	1.0	1.0	1.0
Number of Notes	10	10	10
Percent Overlap	50	50	50
Num Partials	min max	min max	min max
Partials List	mult amp	mult amp	mult amp
Note Durations	min max (seconds)	min max (seconds)	min max (seconds)
Amplitudes	min max (0-1)	min max (0-1)	min max (0-1)
Frequencies	min max (Hz)	min max (Hz)	min max (Hz)
Frequency Skews	min max (Hz)	min max (Hz)	min max (Hz)
Amp Envelope	Flat	Flat	Flat
Freq Envelope	Flat	Flat	Flat

Figure 19: SSUM Additive Synthesis Composition Machine

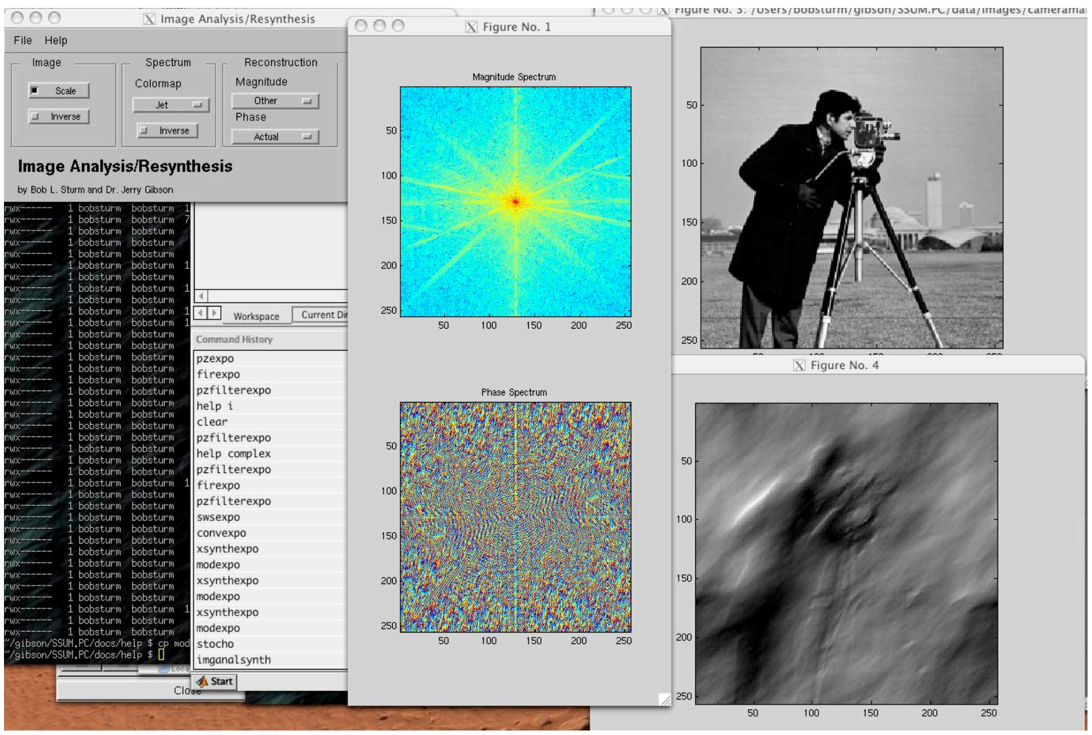
SSUM contains demonstrations of MATLAB programming for creating sound, music, and image. A student can experiment with *Catastochastic*[6] *Additive Synthesis Composition Machine* (Figure 19), to synthesize a "composition" in three parts using various envelopes for amplitudes and frequencies and several other parameters. Parameter ranges can be specified for durations, amplitudes, frequencies, and amplitude and frequency changes. Each section can be visualized and played, and can also be sent to *Sonogram Explorer* (Figure 10) for further analysis.

Image Analysis/Resynthesis Explorer (Figure 20) allows one to change the frequency and phase content of an image for reconstruction. Figure 20 shows an image that was reconstructed using the spectral magnitudes from a different image. The phases of images can similarly be swapped. What is demonstrated is that the phase information in an image is crucial for maintaining edges, while magnitudes are necessary for maintaining the pixel

[6] The word "catastochastic" is a mixture of catastrophic and stochastic.

brightness. Students enjoy using this application to create new images from two. They begin to realize the advantages to working in both the spatial and frequency domains.

A much more complex application is *MATConcat* [28], a concatenative sound synthesis [29] application. This program (Figure 21) uses feature vectors to synthesize a sound from pieces of other sounds using distance metrics specified by the user. For instance, a voice recording can be synthesized using a recording of a flute and specifying an acceptable range for root-mean squared (RMS) or spectral centroid. Several additional options lead to surprising results and numerous creative possibilities, as well as unfortunate issues of copyright infringement.[7]

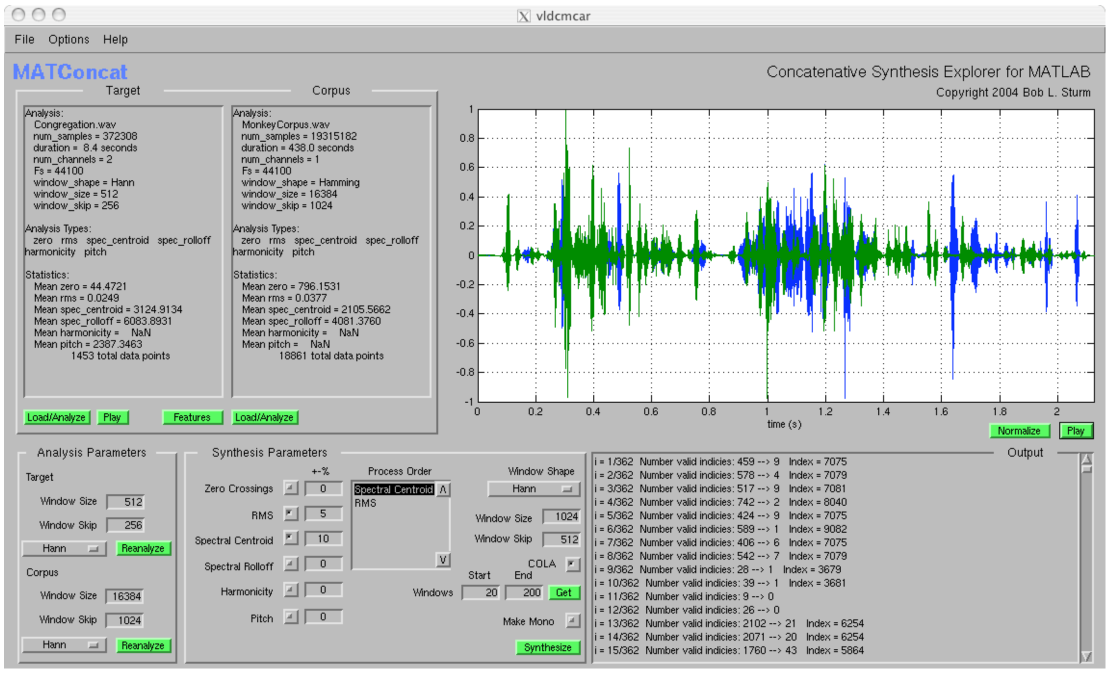


```
pzexpo
firexpo
pzfilterexpo
help i
clear
pzfilterexpo
help complex
pzfilterexpo
firexpo
pzfilterexpo
swsexpo
convexpo
xsynthexpo
modexpo
xsynthexpo
modexpo
xsynthexpo
modexpo
stocho
imganalsynth
```

```
~/gibson/SSUM.PC/docs/help $ cp mod
~/gibson/SSUM.PC/docs/help $
```

Figure 20: SSUM Image Analysis/Resynthesis Explorer

[7] I have used this application in depth for composing several computer music pieces, including “Dedication to George Crumb, American Composer,” and “Gates of Heaven and Hell: Concatenative Variations of a Passage by Mahler.”



```
Analysis:
  Congregation.wav
  num_samples = 372308
  duration =  8.4 seconds
  num_channels = 2
  Fs = 44100
  window_shape = Hann
  window_size = 512
  window_skip = 256

Analysis Types:
  zero   rms   spec_centroid   spec_rolloff
harmonicity   pitch

Statistics:
  Mean zero = 44.4721
  Mean rms = 0.0249
  Mean spec_centroid = 3124.9134
  Mean spec_rolloff = 6083.8931
  Mean harmonicity =     NaN
  Mean pitch = 2387.3463
            1453 total data points
```

```
Analysis:
  MonkeyCorpus.wav
  num_samples = 19315182
  duration = 438.0 seconds
  num_channels = 1
  Fs = 44100
  window_shape = Hamming
  window_size = 16384
  window_skip = 1024

Analysis Types:
  zero   rms   spec_centroid   spec_rolloff
harmonicity   pitch

Statistics:
  Mean zero = 796.1531
  Mean rms = 0.0377
  Mean spec_centroid = 2105.5662
  Mean spec_rolloff = 4081.3760
  Mean harmonicity =     NaN
  Mean pitch =     NaN
            18861 total data points
```

Analysis Parameters

Target: Window Size 512; Window Skip 256; Hann

Corpus: Window Size 16384; Window Skip 1024; Hann

Synthesis Parameters

	+-%
Zero Crossings	0
RMS	5
Spectral Centroid	10
Spectral Rolloff	0
Harmonicity	0
Pitch	0

Process Order: Spectral Centroid, RMS

Window Shape: Hann; Window Size 1024; Window Skip 512; COLA

Windows: Start 20, End 200

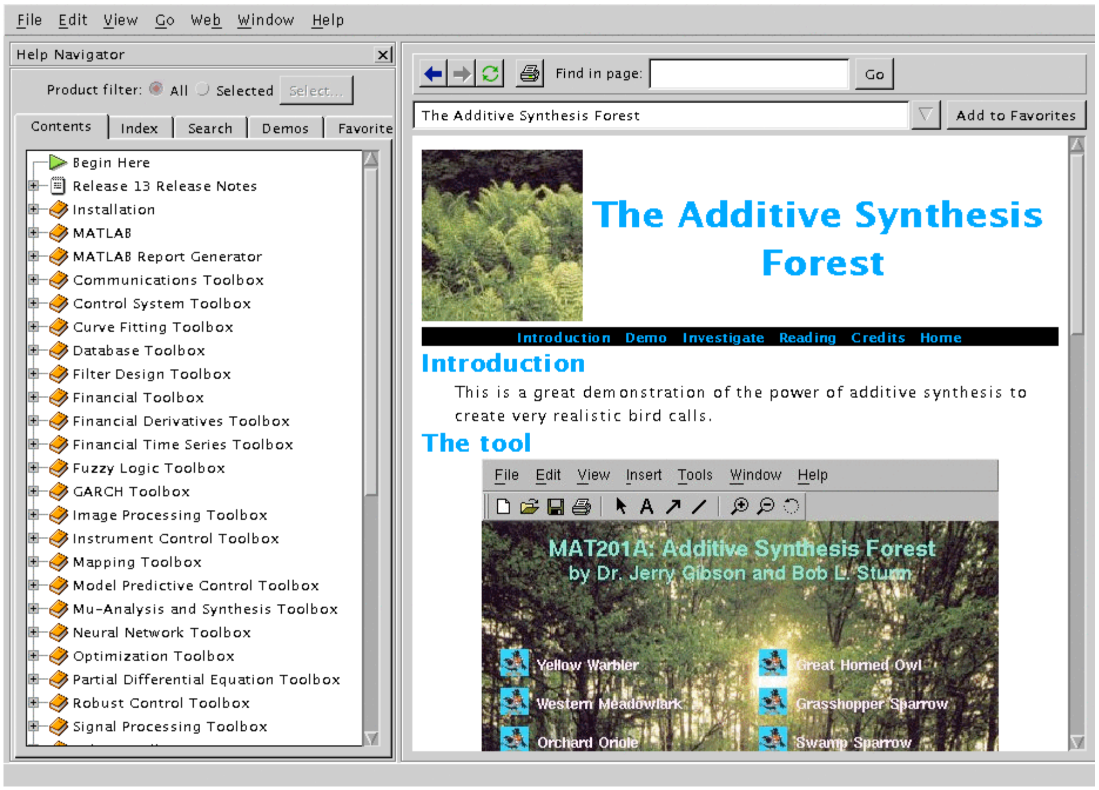
Make Mono

```
i = 1/362  Number valid indicies: 459 --> 9   Index = 7075
i = 2/362  Number valid indicies: 578 --> 4   Index = 7079
i = 3/362  Number valid indicies: 517 --> 9   Index = 7081
i = 4/362  Number valid indicies: 742 --> 2   Index = 8040
i = 5/362  Number valid indicies: 424 --> 9   Index = 7075
i = 6/362  Number valid indicies: 589 --> 1   Index = 9082
i = 7/362  Number valid indicies: 406 --> 6   Index = 7075
i = 8/362  Number valid indicies: 542 --> 7   Index = 7079
i = 9/362  Number valid indicies: 28 --> 1   Index = 3679
i = 10/362  Number valid indicies: 39 --> 1   Index = 3681
i = 11/362  Number valid indicies: 9 --> 0
i = 12/362  Number valid indicies: 26 --> 0
i = 13/362  Number valid indicies: 2102 --> 21   Index = 6254
i = 14/362  Number valid indicies: 2071 --> 20   Index = 6254
i = 15/362  Number valid indicies: 1760 --> 43   Index = 5864
```

Figure 21: MATConcat: Concatenative Synthesis Explorer

These applications, while not clearly demonstrative of MSP concepts like *Sampling Explorer* (Figure 6), or *Convolution Explorer* (Figure 15), are still valuable parts of SSUM because they demonstrate interesting applications of the concepts toward creative ends. For an artist such as a composer they should provide inspiration and foster creative experimentation.

The applications presented so far give only an overview of what SSUM has to offer. But they demonstrate several key aspects of SSUM: its ability to work with sounds and images, to share data between applications, to give quick feedback, and the ease of demonstrating many essential concepts of MSP. These make SSUM an excellent pedagogical tool for lectures, labs, and homework. But with all the options in a given application, where does the student know where to begin? How can a student who doesn't know MSP understand what the sampling rate slider means?


File Edit View Go Web Window Help
Help Navigator
Product filter: All Selected Select...
Contents Index Search Demos Favorites
Begin Here
Release 13 Release Notes
Installation
MATLAB
MATLAB Report Generator
Communications Toolbox
Control System Toolbox
Curve Fitting Toolbox
Database Toolbox
Filter Design Toolbox
Financial Toolbox
Financial Derivatives Toolbox
Financial Time Series Toolbox
Fuzzy Logic Toolbox
GARCH Toolbox
Image Processing Toolbox
Instrument Control Toolbox
Mapping Toolbox
Model Predictive Control Toolbox
Mu-Analysis and Synthesis Toolbox
Neural Network Toolbox
Optimization Toolbox
Partial Differential Equation Toolbox
Robust Control Toolbox
Signal Processing Toolbox
Find in page: Go
The Additive Synthesis Forest
Add to Favorites
Introduction Demo Investigate Reading Credits Home
File Edit View Insert Tools Window Help
MAT201A: Additive Synthesis Forest
by Dr. Jerry Gibson and Bob L. Sturm
Yellow Warbler
Great Horned Owl
Western Meadowlark
Grasshopper Sparrow
Orchard Oriole
Swamp Sparrow

The Additive Synthesis Forest

Introduction

This is a great demonstration of the power of additive synthesis to create very realistic bird calls.

The tool

Figure 22: An SSUM Application Help Page

Like in MAD, every program in SSUM has a help page that presents it, gives directions for its use, and poses a few key questions and topics to explore with the program. These help pages can be accessed either from the main SSUM help page, or from the “Help” menu item in every program. Figure 22 shows a help page that is loaded within the MATLAB environment.

Developing SSUM

As mentioned above, SSUM uses and extends the programming style employed in the excellent “MATLAB Auditory Demonstrations” (MAD) application [25, 26]. Several MAD programs served as starting points for SSUM programs. Whereas MAD presents demonstrations of auditory perception, SSUM presents demonstrations of all things MSP.

Like MAD, all of SSUM is open-source and freely modifiable. Each application resides in its own directory under the main SSUM directory. The code is very modularized in an attempt to work in an object-oriented way. Functions that are shared among many applications, such as those responsible for loading and saving audio and image data, are stored in the “library” directory. This modularization has been crucial to the maintenance and rapid development of SSUM.

All the GUI code is separated from the functional code of the programs. This way modification can be made to the interface without having to break functionality. When a new application is desired it is quite easy to start from an existing one. Whereas before it would take a day to create a new application, using the framework and functionality of an existing SSUM application makes this task one of only hours.

All of the GUIs in SSUM are created using the MATLAB Graphical User Interface Development Environment (GUIDE). The code that is automatically generated is then minimally augmented with callbacks to a main function file. This function file controls the flow of processes and data and updates the data displayed in the GUI. Modifications of GUI elements by other GUI elements (for instance a volume slider changing a volume text box, or mutually exclusive radio buttons) are usually left in the GUI callback function file.

In the case of *FIR Filter Explorer* (Figure 12) there are four files: *firexpo.fig, firexpo.m, firexpogui.m, firexpofn.m*. The .fig file is the GUI layout file used by GUIDE. The *firexpo.m* file contains all the callbacks to the GUI elements. The file *firexpogui.m* is an exported version of the GUI, which merges the design with the callbacks.[8] Finally the *firexpofn.m* file contains the functions called by the program.

Within the file *firexpo.m* are the callbacks for the GUI elements. Figure 23 shows four callbacks from this file. These callbacks are executed when the corresponding GUI element is acted upon. The "play_Callback" is executed when the play button is pressed; when the filter menu is changed, the "filtermenu_Callback" is run. And when the "Apply" filter button is pressed, the "doFilter_Callback" is executed. Two of these callbacks call the *firexpofn.m* file with some action, such as 'playsound,' or 'apply_filter.' A portion of this file is shown in Figure 24.

```
% --- Executes on button press in play.
function play_Callback(hObject, eventdata, handles)
    firexpofn 'playsound';

% --- Executes on selection change in filtermenu.
function filtermenu_Callback(hObject, eventdata, handles)
    contents = get(hObject,'String');
    switch lower(contents{get(hObject,'Value')})
        case {'bandpass','notch'}
            set(handles.cut2,'Visible','on');
        otherwise
            set(handles.cut2,'Visible','off');
    end

% --- Executes on button press in doFilter.
function doFilter_Callback(hObject, eventdata, handles)
    firexpofn 'apply_filter';
```

Figure 23: A portion of GUI code from firexpo.m

[8] Exporting the GUI code makes the .fig file unnecessary for distribution. The exported code is compatible with MATLAB versions before 6.1, whereas the .fig file is not.

```
function firexpofn(action,datastruct)
      handles = get(SSUMfigure,'UserData');
      switch action
          . . .
          case 'playsound'
              if isfield(handles,'audiodata')
                  audiodata = handles.audiodata;
                  button = handles.play;
                  play_audiodata(audiodata, button);
              end
              . . .
          case 'apply_filter'
              if isfield(handles, 'audiodata')
                  handles = apply_filter(handles);
                  updatePlots;
              end
              . . .
      end
      set(f,'UserData',handles);
```

Figure 24: A portion of code from firexpofn.m

As can be seen the function file does most of the work behind the GUI. When the play button is pressed, the “play_Callback” is executed, which then calls the ‘playsound’ action in the function file. This function checks to see if there is any audio data, and then passes the audio data to the “play_audiodata” function, located in the SSUM function library. Similarly, when the “Apply” filter button is pressed, the GUI calls “doFilter_Callback” which then calls the ‘apply_filter’ action in the function file, which then calls a filtering function and updates the plots in the GUI. These specialized functions are located in the function file as other functions.[9]

Most of the callbacks call for some action in the corresponding function file; but in Figure 23 we see the “filtermenu_Callback” just changes the state of some elements in the GUI. In this case the code is located in *firexpo.m*

[9] In MATLAB one can program any number of functions in a single file, like C++. There has to be one main function though, which shares the same name as the file it is contained in.

rather than in the function file. All of the SSUM applications are programmed using this technique.

The modularization of functions and separation of functional code from GUI code makes the application much more accessible and modifiable. Students who are interested in using a particular part of some application don’t have to look through the esoteric code of the GUI to find it. The GUI action can just be traced to the function file, and in there the code can be grabbed.

SSUM and Teaching Media Signal Processing

As stated in the introduction, three things work against a successful syllabus for teaching MSP to media arts students. First, to explain or do anything interesting requires more than a cursory look at the mathematics involved. Second, the pace of the course could be hindered by the need to address the mathematics. And third, the number of topics that can be addressed increases when including discussions of sounds and images. By focusing on using exploratory demonstrations and applications, and programming rather than written homework, an effective syllabus for teaching MSP to the media arts student can be designed.

SSUM is designed first for practical and effective demonstrations, second to provide an interactive experience to enhance one's comprehension of MSP, and third to serve as a repository of algorithms and code. SSUM nicely satisfies these three goals, and creates a fruitful multimedia experience for teaching and learning MSP. All of the applications are quick to compute and display results, so there is little worry for the learning process to spiral out of control or come to a halt. In addition to its ability to teach MSP, SSUM also teaches how to program MATLAB.

A syllabus that uses SSUM can quite naturally choose MATLAB as the programming environment. Devens describes why and how Virginia Polytechnic Institute & State University chose MATLAB as the required software by the engineering and mathematics department [30]. Included in his list are many of the reasons stated in my chapter on MATLAB as the development software. In addition MATLAB can be easily integrated into existing courses, and circumvents the need for students to learn other software packages that can be replaced by it.

By using MATLAB as a tool for a course, one is able to introduce applications first, and thus motivate the students to experiment and learn how they work, as well as create applications of their own. [31] describes the use of MATLAB to “help reconcile the declarative (what is) and imperative (how to) points of view on signals and systems.” During thirteen labs students explore sampling and filtering sounds and images, as well as modulation and control systems. Other uses of MATLAB in DSP classes and laboratories are described in [32, 33]. In addition, because of the relative ease of programming in MATLAB as compared to C++ or JAVA, a student will have more time to concentrate on algorithms rather than compiler errors.

SSUM then becomes a rich collection of routines that work with sound and image; students can use the applications as models to guide their own creative work. Students should be encouraged to discover how the applications work, and are free to use the code for their own work, provided they extend it in other directions. Having working examples at their disposal demonstrates that interesting and complex applications are possible.

Dr. Gibson and I created and implemented a new syllabus using MATLAB and SSUM for the MSP class offered in MAT at UCSB. As a core course in the MAT graduate curriculum, Media Signal Processing Using MATLAB (201A) introduces the concepts of MSP to students who are more versed in art and music than mathematics. Its focus on teaching principles of MSP using MATLAB provides an experimental playground in which students learn by doing, and are motivated by their own artistic interests. It requires however all enrolled students to purchase MATLAB, and to be comfortable with at least trigonometry, complex numbers, and elementary series—which the students learn through a course the quarter before 201A is offered.

201A is not intended to be a survey, but the students should finish with at least an understanding of digital signals (e.g. samples), digital operations (e.g. sampling), the frequency domain (e.g. spectra), conversion between analogue and digital signals (e.g. interpolation), filtering (e.g. convolution), and time-frequency analysis (e.g. Fourier transform) and synthesis (e.g. LPC). With this knowledge in place the media arts student is more equipped to attempt the complex technological issues of digital media, whether they are composing or programming.

201A: Media Signal Processing with MATLAB

A new design for 201A was offered during spring quarter 2004. For two days a week, two hours each day, lectures are presented in two parts. During the first hour material is presented that is elucidated by SSUM. After a short break the rest of the time is spent answering questions, reviewing assignments, and demonstrating programming concepts in MATLAB. The syllabus for the 201A is shown in Table 1.

Date YYYYMMDD	**Topics covered**
Week 0	
20040329	Overview of class, introduction to sampling
20040331	Analog to digital conversion, aliasing; Introduction to MATLAB, SSUM
Week 1	
20040405	Mathematical representations of signals, Fourier Series
20040407	Combinations of Sines; Introduction to Filtering; introduction to building GUIs in MATLAB
Week 2	
20040412	More filtering; introduction to analysis. MATLAB GUI callbacks.
20040414	The Fourier Transform. More MATLAB GUI callbacks.
Week 3	
20040419	Modulation and Spectra
20040421	"Mortuos Plango, Vivos Voco:" Spectral Morphing; Sonification of Data

Date YYYYMMDD	**Topics covered**
Week 4	
20040426	Filtering, the impulse response, convolution
20040428	More convolution, filtering, z-plane
Week 5	
20040503	Introduction to z-transforms, poles and zeros
20040505	FIR filters: deriving the impulse and frequency response
Week 6	
20040510	IIR filters: z-transform, stability
20040512	More IIR filters
Week 7	
20040517	FT, DFT, DTFT, STFT, and FFT; effects of windowing
20040519	Block Diagrams; Direct Form I, II
Week 8	
20040524	Linearity, Time-Invariance, and LPC
20040526	Application of perception to signal processing
Week 9	
20040531	Memorial Day: NO CLASS
20040602	Acoustics of the Musical Iced Tea Cans
Week 10	
20040607	Class presentations

Table 1: Syllabus for 201A Media Signal Processing with MATLAB

201A is designed around MATLAB programming and SSUM. Lectures are enhanced with demonstrations of concepts such as sampling and aliasing. Students are required to work with SSUM, find creative applications of the topics presented, and explore them using MATLAB. More emphasis is placed on programming media processing algorithms using MATLAB, than on solving for filter coefficients by hand. Therefore we are relying on the illustrative power of SSUM and MATLAB to increase the potential for comprehension and inspire the students to create their own applications using SSUM as a model. The class has therefore been transformed from one requiring only mathematical practice, to one requiring signal processing programming in MATLAB. In this way the class provides a theoretical and practical experience, rather than just a pedantic one.

There is no required textbook for this class, but MATLAB and the signal processing toolbox is necessary to complete the homeworks and the final project. The grading is broken down into the following parts: homeworks 40%, attendance 10%, and the final project 50%. The homework assignments and requirements for the final project can be found in Appendix B: Assignments for 201A. Though no text was required, several articles and photocopies of texts were distributed during the quarter illustrating particular aspects of MSP. These included sections from [1, 34, 35, 36, 37, 38]. Some handouts were created as well to demonstrate certain mathematics, like summing phasors in the complex domain.

Reactions

Response to SSUM was very positive. It was essential for quick demonstrations of complex concepts like the frequency domain, Fourier series, and filtering. Using SSUM provided a natural progression of topics. It was quite easy to move between applications to show, for instance, the finite difference equation, the poles and zeros, and the frequency and impulse response of the system. Then the filter could be applied to numerous sounds to hear its effects. By week three we were already presenting filtering and the Fourier transform. By the end of week five, after a whirlwind tour through the topics of MSP, it was time to return back to beginning and take a closer look at the math.

It was difficult to get the students to use SSUM on their own time. When it became required on homework, many people suddenly complained that it wasn’t working—revealing that they hadn’t tried it on their computers. The platform compatibility issues that arose were quickly addressed and easily solved due to the modularity of the SSUM code.

Finding a balance between class topics and increasing the student’s skill with MATLAB was tough. As the students didn’t have enough knowledge to begin working intelligently with signals, for instance, some other topic needed to be used as a conduit for learning MATLAB. The best topic was in gradually learning how SSUM works, from making the GUIs to writing the functions. As can be seen from the homework assigned (Appendix B: Assignments for 201A), about 200 of the 500 points possible had something to do with making interfaces. Many complained during the course that too much time and importance was being spent on making GUIs.

As soon as the students had enough background in digital signals and the MATLAB language, the homework focus shifted from making interfaces to creating and working with signals. By the fourth homework they were given the task of building their own complete application using frequency modulation to synthesize musical instrument tones.

In response to several students asking for more “signal processing” related work, a choice was given on the last homework: analyze and resynthesize recorded birdsong, or investigate the properties of windowing on two different signals. Even though the latter was touted as being the harder one, most people chose it. What was expected was a more formal analysis of windowing, complete with programming examples and diagrams. What was received was confused work[10] that used SSUM to find the answers, instead of

[10] Some of the words used to describe the effects of the windows: “spikier,” and “pointier.”

programs to explore windowing. Because it wasn’t specified that the assignment required advanced programming, students found it to be quite easy using SSUM.

After the fifth homework it was apparent that, with only four weeks left in the quarter, the students needed to spend most of their time on their project. The requirements for the final project were discussed (Appendix B: Assignments for 201A) in the third week of class, giving the students ample time to think of and research project topics. At the end of the sixth week they turned in proposals, which were then reviewed and returned to the students for revision. Most of the projects were not practical, so we guided them to more realistic goals and suggested places to begin.

Weekly contact with the students was maintained to ensure the students were working on projects. Only a few students took advantage of this and had excellent starts; but a few were left in the final weeks having to start over because they didn’t heed our advice. In the end the projects came together nicely, and most of them displayed a high level of sophistication. All of them demonstrated hard work; and the students showed genuine interest in the topics they chose.

The applications they wrote in MATLAB made good use of the functionality and GUI programming taught in the homework. A few students mentioned that had we not taught GUI programming, their applications would have been less manageable. This demonstrates a good point: using a GUI can relieve a considerable amount of overhead when working with many variables. Loading an image, specifying a filter, and displaying a result are much easier with a few mouse clicks, than editing scripts and functions. Several students admitted the process was tough, but were pleased with and proud of their results.

Discussion

When a student more versed in music, video, and art, is bombarded with unintuitive complex mathematics and homework involving convolution by hand for example, the results are alienation, discontent, and apathy toward the material. This was discovered in the 201A class taught in 2003 by Dr. Gibson and myself; even though most of the thirteen graduate students had been through a previous course that refreshed their math skills, they had difficulty relating to the material, and consequently didn't see its use or value. Instead of finding creative applications for the concepts they were attempting to learn, the students spent most of their time working out problems on paper, such as adding phasors, computing spectra, and performing convolutions.

Even though most students did well, the lectures, homework, midterm and final, were not effective for exciting discussion and revealing the importance of this subject to their field. Furthermore the need to address the mathematics severely slowed class progress. The students also complained about the required text [1] .[11]

When questioned a year later about the usefulness of 201A, most of the students from the 2003 class responded that the class had been of no use to them. Their retention of the information, such as what an IIR filter is, was very low. Only the most technical students from that class responded positively about it. Most of the criticism focused on the text and the lack of examples and demonstrations. With SSUM the lack of demonstrations is well taken care of.

The response to the material in the 2004 class has been a lot different. Much more time was spent in class talking about applications, presenting demonstrations, and working in MATLAB, than reviewing homework problem sets, and preparing students for the midterm and final exams. Little attempt

[11] I have yet to find an introductory signal processing text that is exciting and friendly to artists.

was made to “dumb-down” the material; instead, advanced mathematics (integral calculus) was used, but proofs and derivations were avoided. The concepts were always elucidated with SSUM. The students were more responsive to the material and asked more questions. There were several moments when I saw “lights turn on,” particularly when we found the expression for the discrete Fourier transform from a finite difference equation. Suddenly the big black box called “fft” was a little less mysterious.

It might be stated that focusing on MATLAB in the new syllabus replaces the difficulty of using mathematics with the difficulty of learning programming. Thus the class will become more about programming MATLAB than learning MSP. However, due to the multimedia nature of MSP, it makes more sense to concentrate on learning the theory through building applications than struggling with abstract mathematics.

Some students suggested that MATLAB should be learned on their own time and that only the first homework should be devoted to learning MATLAB techniques. Assuming that the students would do this would be a big mistake. Since MATLAB is a required portion of the final project it is absolutely necessary to create assignments that will force them to learn and use MATLAB. The “MATLAB manual” (the electronic manual that comes with a MATLAB installation) was constantly assigned as reading, but from their questions it was apparent not many students complied.

More focus was placed on learning GUI programming in the beginning of the course for four reasons. First the students needed to learn MATLAB by doing something interesting. Second they needed to learn functional programming, rather than just writing scripts. Third, in order to even understand how SSUM works, and thus to pick out code that interests them, they needed to become comfortable with the GUI structure in front of the applications. Finally, their final project should either be a demonstration of a concept (like SSUM), or an application. Knowing how to wrap your program in a GUI makes it much easier and interesting to use.

It could be argued that the homework still didn't have enough to do with the class content. Even though students were required to work with frequency modulation, it wasn't specifically covered in class, and no one took it further by, for example, looking at how the index of modulation affects the frequency content. They could have easily looked at the sounds using *Sonogram Explorer,* or *Fourier Explorer*.

As mentioned above, it was tough to get the students to use SSUM on their own time. This lack of voluntary exploration is echoed in [40], originally written in 1995 when the campus network system was a MS-DOS token ring, and only 68% of the students knew about e-mail. Though the computer resources are much more user-friendly now, students must still be "academically compelled" to use the tools [40]. Though during 201A they may have not used SSUM, it became essential for their final projects as at least a collection of code from which they borrowed.

Catering to the creative motivations of the media arts student, and using plenty of demonstrations, a class can approach the difficult concepts of MSP with enthusiasm rather than dread. "If the teachers can create an enduring fascination for the subject matter, the job's almost over: the more the students love the subject, the less help they need in their studies" [39]. SSUM was essential for providing a practical and entertaining experience. And if in the end students still find nothing useful in MSP, at least they have gained programming experience.

Conclusion

SSUM has been developed to satisfy the needs for illustrative and inspiring demonstrations to make MSP a discipline approachable by students who may never posses an ability in advanced mathematics. Using SSUM students not only receive an interactive introduction to MSP, but also learn how to program algorithms using MATLAB. SSUM and the syllabus presented above provide highly effective methods for teaching MSP to any introductory student in this equally creative and technical field. SSUM became absolutely essential to the smooth and quick pace of our course on MSP. Without it the class would have remained dull, pedantic, and lost in mathematics. The applicability of SSUM to other media arts programs and even introductory engineering courses is very clear.

One of the disadvantages of CBE discussed in [40] is the time spent preparing the computer resources. One must weigh the benefit for the students with the cost of preparing the materials. For SSUM this is not an issue. It has taken a year to develop SSUM into its current state, but since it was developed using modularity, its upkeep is minimal. MATLAB will remain for a long time a leader of academic and institutional engineering software, and so SSUM will continue to grow and be useful toward its intended goals.

SSUM will be maintained and extended as a project from MAT. It is predicted that as more people use SSUM they will create other interesting demonstrations. Incorporating these into SSUM will further enrich it as a resource. SSUM can be downloaded for free from http://www.mat.ucsb.edu/~b.sturm.

Acknowledgments

The MathWorks, Inc., the makers of MATLAB, has supported this research by providing full multi-platform licenses to MATLAB. Thanks to Dr. Gibson for allowing me to run free with the design and implementation of SSUM. This research was supported with financial assistance from Dr. Gibson and the graduate program in Media Arts & Technology, UCSB.

References

1. J. H. McClellan and R. Schafer and M. A. Yoder, DSP First: A Multimedia Approach, Prentice Hall, New Jersey, 1998.
2. J. H. McClellan and R. Schafer and M. A. Yoder, Signal Processing First, Prentice Hall, New Jersey, 2003.
3. K. Steiglitz, A DSP Primer: with Applications to Digital Audio and Computer Music, Addison Wesley, Menlo Park, CA, 1996.
4. P. R. Cook, Real Sound Synthesis for Interactive Applications, A. K. Peters, Massachusetts, 2002.
5. U. Zoelzer (Editor), DAFx: Digital Audio Effects, Wiley, New York, 2002.
6. MATLAB is created by The MathWorks, Inc. http://www.mathworks.com/
7. A. Clausen and A. Spanias, "An Internet-based Computer Laboratory for DSP Courses", in *Proceedings of Frontiers in Education*, 1998. Available at: http://fie.engrng.pitt.edu/fie98/
8. R. J. Radke and S. Kulkarni, "An Integrated Matlab Suite for Introductory DSP Education," in *Proceedings of the First Signal Processing Education Workshop*, 2000. Available at: http://www.ee.princeton.edu/~rjradke/papers/radkedsp00.pdf
9. M. Rahkila and M. Karjalainen, "Considerations Of Computer Based Education In Acoustics And Signal Processing," in *Proceedings of Frontiers in Education*, 1998. Available at: http://fie.engrng.pitt.edu/fie98/
10. M. Rahkila, “A Computer Based Education System for Signal Processing,” Helsinki Univeristy of Technology, Department of Electrical Engineering, Master’s Thesis, 1996. Available at: http://www.acoustics.hut.fi/~mara/cbe/mst/

11. M. B. Joaquim and J. C. Pereira and V. A. de Oliveira, "Course On DSP Design Using MATLAB," in *Proceedings of Frontiers in Education*, 1998. Available at: http://fie.engrng.pitt.edu/fie98/
12. A. C. Hague, Towards Deeper Learning with Hand-Crafted Courseware, (PhD Thesis) University of York, Department of Computer Science, U.K., 1997. Available at: http://citeseer.nj.nec.com/hague97towards.html
13. JAVA Digital Signal Processing Editor. Available at: http://www.eas.asu.edu/~midle/jdsp/jdsp.html
14. J. McCartney, "SuperCollider: A new real-time sound synthesis language," in *Proceedings of the International Computer Music Conference*, 1996. Available at: http://www.audiosynth.com
15. M. Puckette, "Pure Data," in *Proceedings of the International Computer Music Conference*, 1996. Available at: http://www.crca.ucsd.edu/~msp/Publications/icmc96.ps
16. Max/MSP is distributed by Cycling74: http://www.cycling74.com/
17. Simulink is created by The MathWorks, Inc. http://www.mathworks.com/
18. Mathematica is created by Wolfram Research, Inc. http://www.wolfram.com/
19. Maple is created by Maplesoft. http://www.maplesoft.com/
20. Octave. Available at: http://www.octave.org
21. LabVIEW is created by National Instruments. http://www.ni.com/labview/
22. R. Remez, P. Rubin, D. Pisoni, and T. Carrell, "Speech perception without traditional speech cues," *Science*, Vol. 212, 947-950, 1981.
23. M. Nagrial, "Education and Training in Engineering Software and Applications," in *Proceedings of the International Conference on Engineering Education*, 2002. Available at: http://citeseer.nj.nec.com/560624.html
24. Baudline is free software for Linux only, available at: http://www.baudline.com

25. M. Cooke, H. Parker, G. J. Brown and S. N. Wrigley, "The interactive auditory demonstrations project," *Eurospeech Conference*, 1999. Available at: http://www.dcs.shef.ac.uk/~martin/MAD/docs/articles.htm
26. M. Cooke, et al., “MAD: MATLAB Auditory Demonstrations,” 1999. Available at: http://www.dcs.shef.ac.uk/~martin/MAD/docs/mad.htm
27. A. S. Bregman and P. Ahad, “Demonstrations of auditory scene analysis: the perceptual organization of sound,” CD, MIT Press, Cambridge, Massachusetts, 1995.
28. B. L. Sturm, “MATConcat: An Application for Exploring Concatenative Synthesis in MATLAB,” to be published in 2004 *Proceedings of the International Conference of Computer Music*, Miama, FL, 2004.
29. A. Hunt and A. Black, “Unit selection in a concatenative speech synthesis system using a large speech database,” ICASSP 1(1), 373–376, 1996.
30. P. E. Devens, "MATLAB & Freshman Engineering," in *Proceedings of the ASEE Annual Conference & Exposition*, 1999. Available at: http://www.succeed.ufl.edu/search/seepaper.asp?paperid=284
31. E. A. Lee, “Designing a Relevant Lab for Introductory Signals and Systems", in *Proceedings of the First Signal Processing Education Workshop*, 2000. Available at: http://ptolemy.eecs.berkeley.edu/publications/papers/00/spe2/
32. D. E. Melton, C. J. Finelli and L. M. Rust, "A Digital Signal Processing Laboratory with Style," in *Proceedings of 29th ASEE/IEEE Frontiers in Education Conference*, 1999. Available at: http://fie.engrng.pitt.edu/fie99/
33. U. Rajashekar and A. C Bovik, "Interactive DSP Education Using MATLAB Demos", in *Proceedings of the First Signal Processing Education Workshop*, 2000. Available at: http://www.ece.utexas.edu/~umesh/publications.htm

34. F. R. Moore, “An Introduction to the Mathematics of Digital Signal Processing: Part I: Algebra, Trigonometry, and the most Beautiful Formula in Mathematics,” *Computer Music Journal*, Vol. 2, No. 1, 1978.

35. F. R. Moore, “An Introduction to the Mathematics of Digital Signal Processing: Part II: Sampling, Transforms, and Digital Filtering,” *Computer Music Journal*, Vol. 2, No. 2, 1978.

36. J. Harvey, “Mortuos Plango, Vivos Voco: A Realization at IRCAM,” *Computer Music Journal*, Vol. 5, No. 4, 1981.

37. F. J. Harris, “On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform,” *Proceedings of the IEEE*, Vol. 66, No. 1, 1978.

38. B. L. Sturm, “Surf Music: Sonification of Ocean Buoy Spectral Data,” in *Proceedings of the International Conference on Auditory Display*, Kyoto, Japan, 2002.

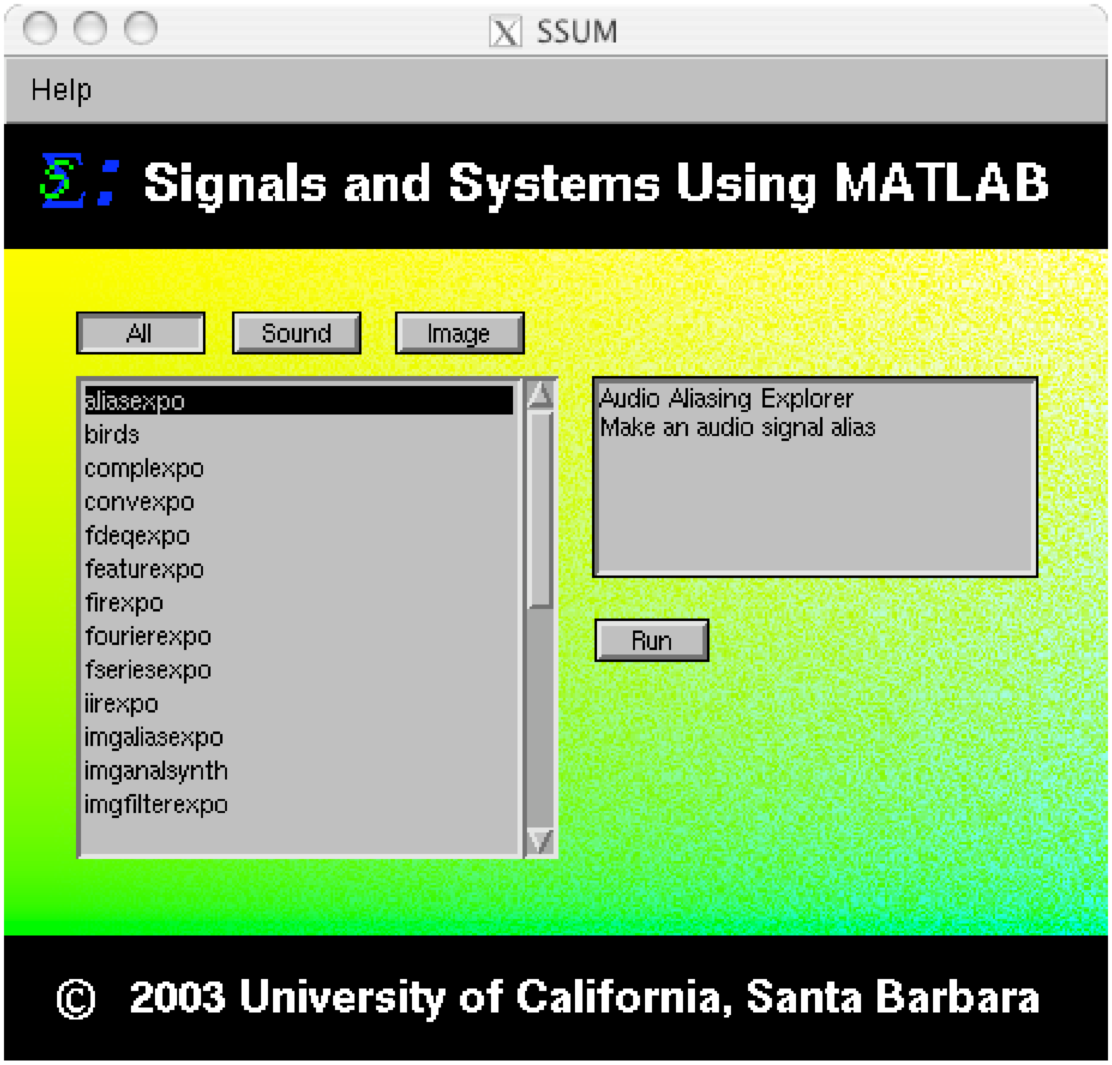
39. J. Koumi, “Designing for Learning---Effectiveness with Efficiency,” In Effective Screenwriting for Educational Television, ed. R. Hoey, Kogan Page Ltd, U.K., pp 230 – 239, 1994.

40. C. A. Cañizares and Z. T Faur, “Advantages and disadvantages of using various computer tools in electrical engineering courses,” *IEEE Trans. On Education*, vol. 40, No. 3, 1997, pp 166 – 171.

Appendix A: Current SSUM Applications

SSUM Main GUI

Highlight an application to learn what it does. Click "Run" to execute the application. The "All", "Sound", and "Image" toggle buttons filter the relevant applications. Selecting "Sound" will show all applications having to do with sound.



Additive Synthesis Forest

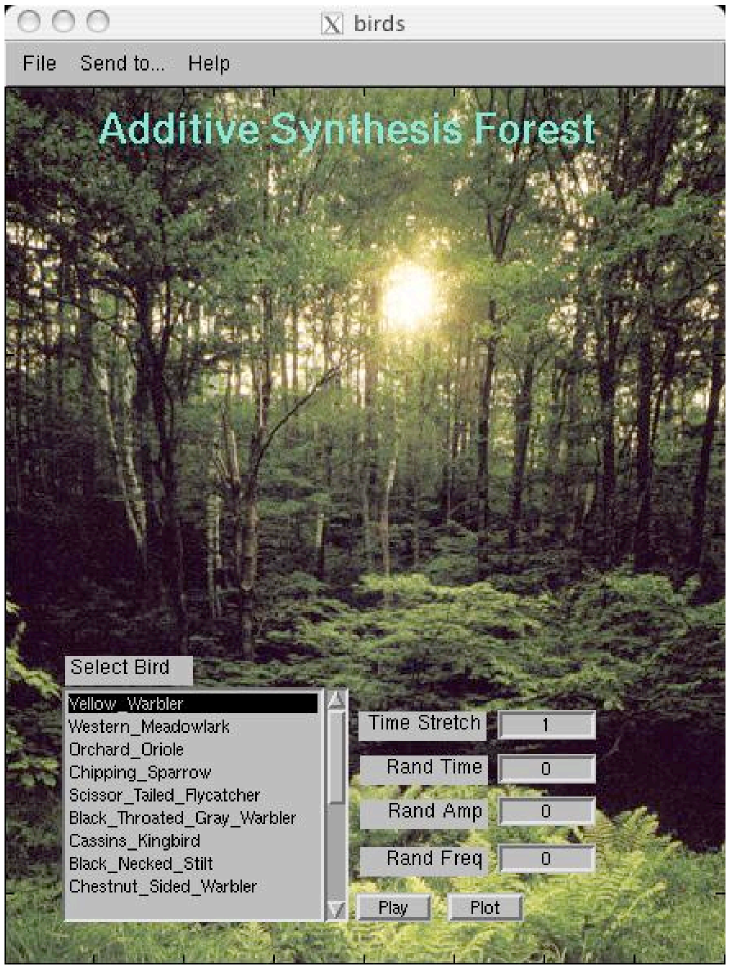
This is a great demonstration of the power of additive synthesis to create realistic bird song.

Simply select a bird and click “Play.” The random bird is a mixture of all the bird calls with random parameters. The “Plot” button will show the frequency and amplitude envelopes of the synthesis parameters.

The synthesis can be modified using the text boxes to the right. The Time Stretch will increase the duration of the envelopes. “Rand Time,” “Rand Amp,” and “Rand Freq,” will change times, amplitudes, and frequencies within the envelopes.

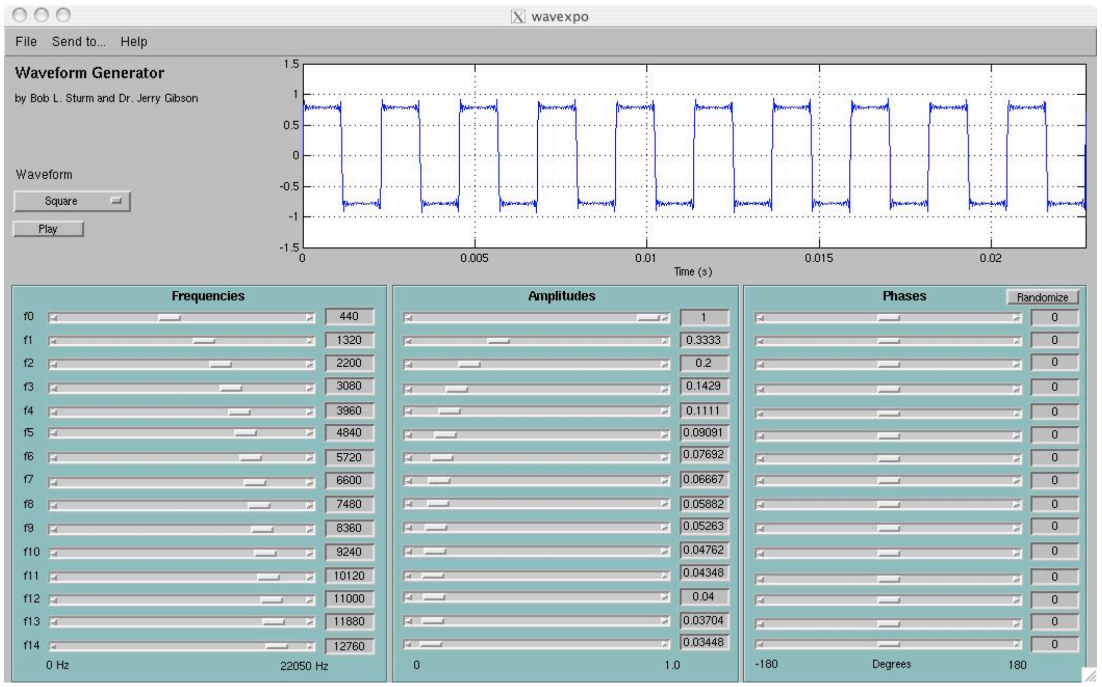
From the menu the synthesized bird song can be sent to the Sonogram Explorer or Fourier Explorer. The resulting signal can be saved to a soundfile from the File menu.

The original code for these birds was in Common Lisp Music, (CLM), developed at the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University. I translated this code during summer 2003.



Additive Synthesis Waveform Explorer

This is a demonstration of adding sine waves to produce different waveforms.



Waveform Generator

by Bob L. Sturm and Dr. Jerry Gibson

Waveform: Square

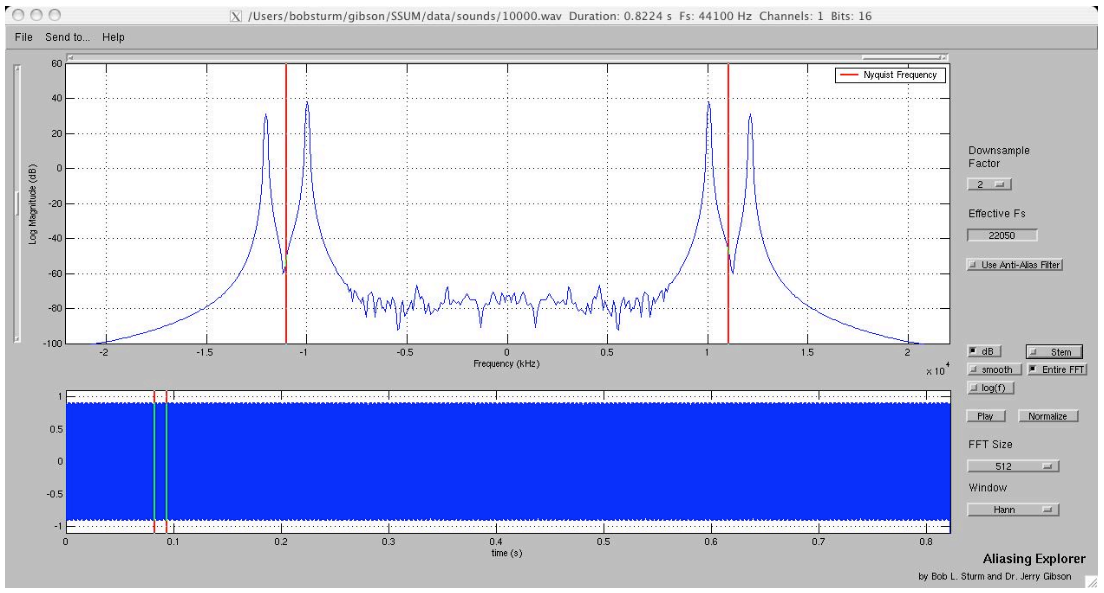
	Frequencies	Amplitudes	Phases
f0	440	1	0
f1	1320	0.3333	0
f2	2200	0.2	0
f3	3080	0.1429	0
f4	3960	0.1111	0
f5	4840	0.09091	0
f6	5720	0.07692	0
f7	6600	0.06667	0
f8	7480	0.05882	0
f9	8360	0.05263	0
f10	9240	0.04762	0
f11	10120	0.04348	0
f12	11000	0.04	0
f13	11880	0.03704	0
f14	12760	0.03448	0
	0 Hz – 22050 Hz	0 – 1.0	-180 – Degrees – 180

Begin by selecting a waveform from the menu. You can then adjust any of the parameters to see its effect on the waveform. After selecting a different fundamental (f0), select a waveform again to synthesize its shape. The signal can be played using the "Play" button. In the phases box, the "Randomize" button will randomize all 15 phases.

Using the menu above, the signal can be sent to the Fourier Explorer, Sonogram Explorer, or the Aliasing Explorer. The signal can also be saved to a sound file from the File menu.

Audio Aliasing Explorer

This application allows to user to simulate the effects of aliasing in audio.



After loading a soundfile using the menu at the top, its time-domain waveform will appear in the bottom plot. The two red bars on this plot denote the window over which the Fourier transform is performed. The magnitudes of the transform are displayed in the top plot. The red bars in the frequency plot denote the Nyquist frequency. The vertical slider scales the range of the plot; the horizontal slider scales the frequency domain plotted. By clicking and holding on a red bar in the time-domain plot the window can be dragged over the sound to show how the spectrum changes.

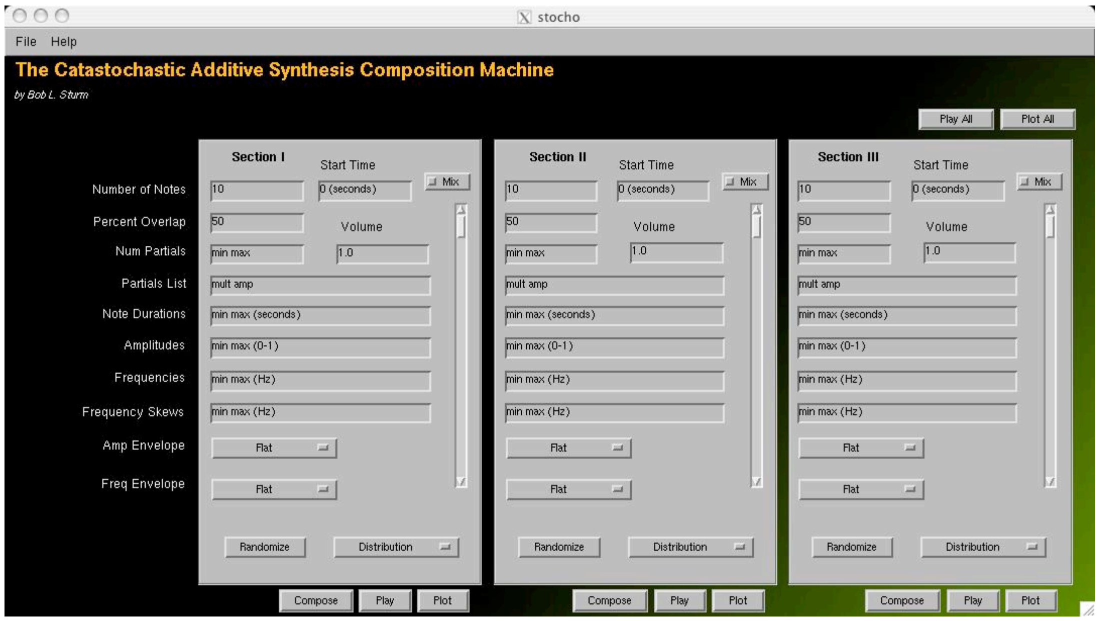
By choosing a downsample factor from the menu, the original signal is decimated without filtering, and redisplayed. The signal is resampled to the original sample rate, but the effective sampling rate is Fs/factor. An anti-aliasing filter can be applied before decimation by clicking on “Use Anti-Alais Filter.”

The menus on the lower right side of the figure can change the fast Fourier transform window size and shape. The sound can be normalized and played. The check boxes change how the transform is displayed. Using the menu at the top, the loaded sound data can be sent to other applications, like the Sonogram Explorer.

Some code to make this demonstration was taken from the MAD program "wavspect."

Catastochastic Additive Synthesis Composition Machine

This application composes music using additive synthesis parameters supplied by the user.



	Section I	Section II	Section III
Number of Notes	10	10	10
Start Time	0 (seconds)	0 (seconds)	0 (seconds)
Percent Overlap	50	50	50
Volume	1.0	1.0	1.0
Num Partials	min max	min max	min max
Partials List	mult amp	mult amp	mult amp
Note Durations	min max (seconds)	min max (seconds)	min max (seconds)
Amplitudes	min max (0-1)	min max (0-1)	min max (0-1)
Frequencies	min max (Hz)	min max (Hz)	min max (Hz)
Frequency Skews	min max (Hz)	min max (Hz)	min max (Hz)
Amp Envelope	Flat	Flat	Flat
Freq Envelope	Flat	Flat	Flat

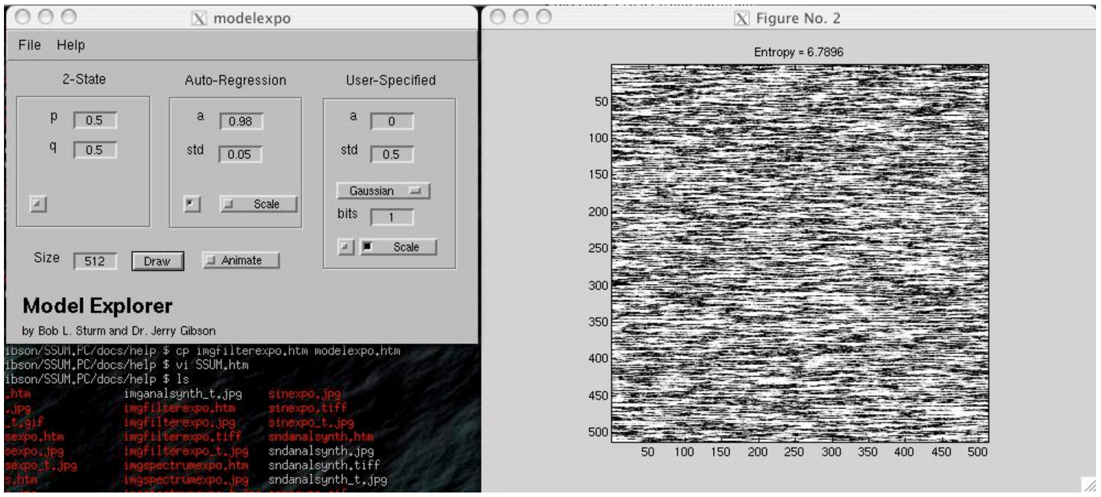
Begin by pressing "Compose" for any section. Default values are loaded in the fields. For any section the number of notes can be modified. The start time of each section can be changed. By clicking on the "Mix" box, that section will be mixed into the final section. The "Percent Overlap" value determines how many sounds are playing at once. The volume of any section can be changed either using the text box or the vertical slider.

The additive synthesis parameters are the number of partials, the list of partials in multiples of the fundamental and amplitude (for instance [1 1 2.1 0.5 3 0.2]). Ranges can be specified for the note durations, amplitudes, frequencies, and frequency skews. Finally different envelopes can be chosen for the amplitude and frequency of each note. Once entered, press "Compose" to synthesize the section. Upon finishing a time domain plot will be shown.

This program is not intended as a demonstration of anything in particular. It is included as an example of something that can be done using MATLAB and basic signal processing. The resulting signal can be saved to a soundfile from the File menu.

Communication Models Explorer

This application demonstrates the propagation of error in different models for communication systems.



```
ibson/SSUM.PC/docs/help $ cp imgfilterexpo.htm modelexpo.htm
ibson/SSUM.PC/docs/help $ vi SSUM.htm
ibson/SSUM.PC/docs/help $ ls
.htm              imganalsynth_t.jpg     sinexpo.jpg
.jpg              imgfilterexpo.htm      sinexpo.tiff
_t.gif            imgfilterexpo.jpg      sinexpo_t.jpg
sexpo.htm         imgfilterexpo.tiff     sndanalsynth.htm
sexpo.jpg         imgfilterexpo_t.jpg    sndanalsynth.jpg
sexpo_t.jpg       imgspectrumexpo.htm    sndanalsynth.tiff
s.htm             imgspectrumexpo.jpg    sndanalsynth_t.jpg
```

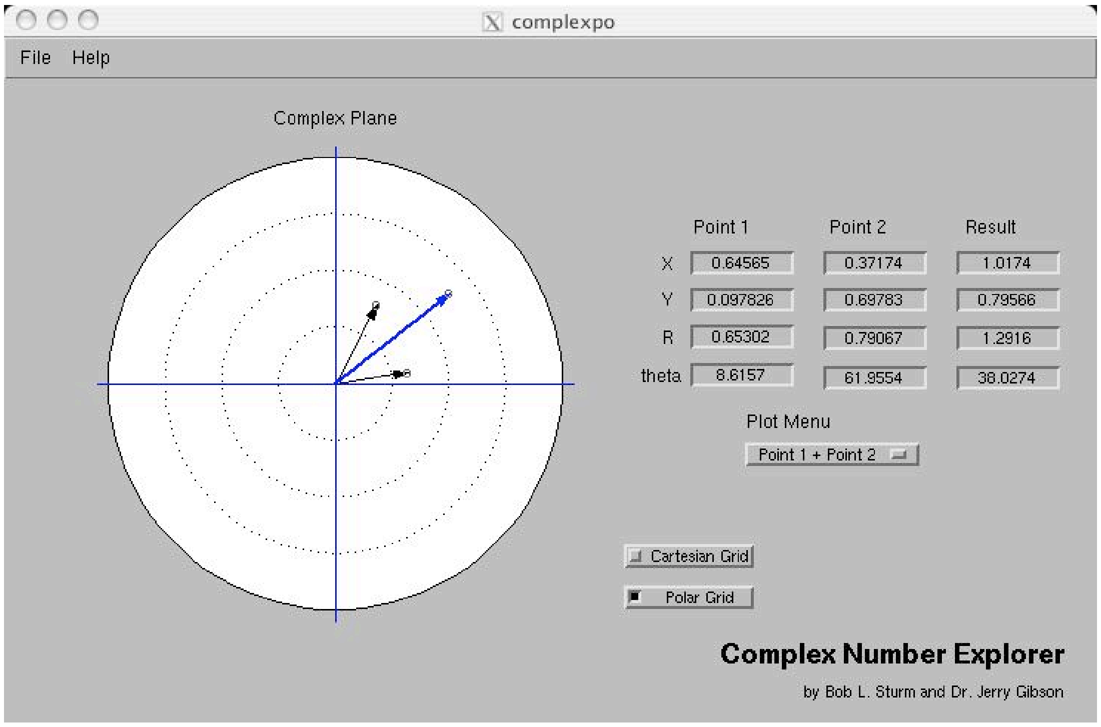
Select a model and click “Draw.” An image will be created that shows the propagation of error. A white pixel represents an error. When “Animate” is selected, the drawing will be incremental. At the top of the image will be the calculated entropy of the particular trial.

The “2-State” model is a basic Markov chain. The probability p determines the likelihood of encountering an error; the probability q determines the likelihood of returning from an error. The “Auto-Regression” model is a Markov chain with memory. The number “a” is the percent of the output fed back. The standard deviation “std” determines the noise in the system. When “Scale” is selected the range of pixel values will be rescaled to the maximum and minimum values in the image. The “User Specified” allows different types of noise to be introduced, and the number of bits used for each pixel.

The resulting image can be saved from the File menu.

Complex Number Explorer

This is a demonstration of complex numbers and vectors.



	Point 1	Point 2	Result
X	0.64565	0.37174	1.0174
Y	0.097826	0.69783	0.79566
R	0.65302	0.79067	1.2916
theta	8.6157	61.9554	38.0274

Plot Menu: Point 1 + Point 2

Cartesian Grid

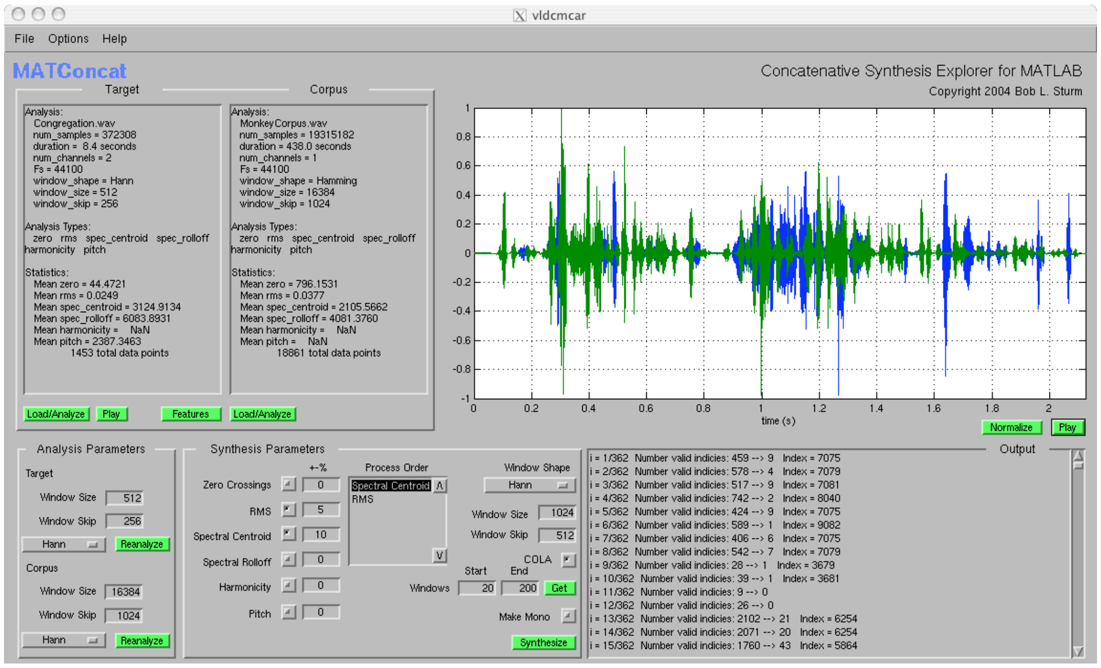
[✓ Polar Grid]

After starting the program you can either enter the x, y, r, or theta values in the text boxes, or click and drag the point around the plot. From the plot menu you can select which point to display, and which operation to perform (addition, subtraction). The thick blue arrow represents the result of the operation. The result is also shown in the text boxes under "Result."

The points can be plotted in either rectangular or polar coordinates, depending on the selected check box.

Concatenative Synthesis Explorer

This application demonstrates concatenative synthesis using signal feature extraction.



```
Analysis:
  Congregation.wav
  num_samples = 372308
  duration =  8.4 seconds
  num_channels = 2
  Fs = 44100
  window_shape = Hann
  window_size = 512
  window_skip = 256

Analysis Types:
  zero   rms   spec_centroid   spec_rolloff
harmonicity   pitch

Statistics:
  Mean zero = 44.4721
  Mean rms = 0.0249
  Mean spec_centroid = 3124.9134
  Mean spec_rolloff = 6083.8931
  Mean harmonicity =    NaN
  Mean pitch = 2387.3463
           1453 total data points
```

```
Analysis:
  MonkeyCorpus.wav
  num_samples = 19315182
  duration = 438.0 seconds
  num_channels = 1
  Fs = 44100
  window_shape = Hamming
  window_size = 16384
  window_skip = 1024

Analysis Types:
  zero   rms   spec_centroid   spec_rolloff
harmonicity   pitch

Statistics:
  Mean zero = 796.1531
  Mean rms = 0.0377
  Mean spec_centroid = 2105.5662
  Mean spec_rolloff = 4081.3760
  Mean harmonicity =    NaN
  Mean pitch =    NaN
           18861 total data points
```

Analysis Parameters

Target: Window Size 512; Window Skip 256; Hann

Corpus: Window Size 16384; Window Skip 1024; Hann

Synthesis Parameters

	+-%
Zero Crossings	0
RMS	5
Spectral Centroid	10
Spectral Rolloff	0
Harmonicity	0
Pitch	0

Process Order: Spectral Centroid, RMS

Window Shape: Hann; Window Size 1024; Window Skip 512; COLA; Windows Start 20, End 200; Make Mono

```
i = 1/362  Number valid indicies: 459 --> 9   Index = 7075
i = 2/362  Number valid indicies: 578 --> 4   Index = 7079
i = 3/362  Number valid indicies: 517 --> 9   Index = 7081
i = 4/362  Number valid indicies: 742 --> 2   Index = 8040
i = 5/362  Number valid indicies: 424 --> 9   Index = 7075
i = 6/362  Number valid indicies: 589 --> 1   Index = 9082
i = 7/362  Number valid indicies: 406 --> 6   Index = 7075
i = 8/362  Number valid indicies: 542 --> 7   Index = 7079
i = 9/362  Number valid indicies: 28 --> 1   Index = 3679
i = 10/362  Number valid indicies: 39 --> 1   Index = 3681
i = 11/362  Number valid indicies: 9 --> 0
i = 12/362  Number valid indicies: 26 --> 0
i = 13/362  Number valid indicies: 2102 --> 21   Index = 6254
i = 14/362  Number valid indicies: 2071 --> 20   Index = 6254
i = 15/362  Number valid indicies: 1760 --> 43   Index = 5864
```

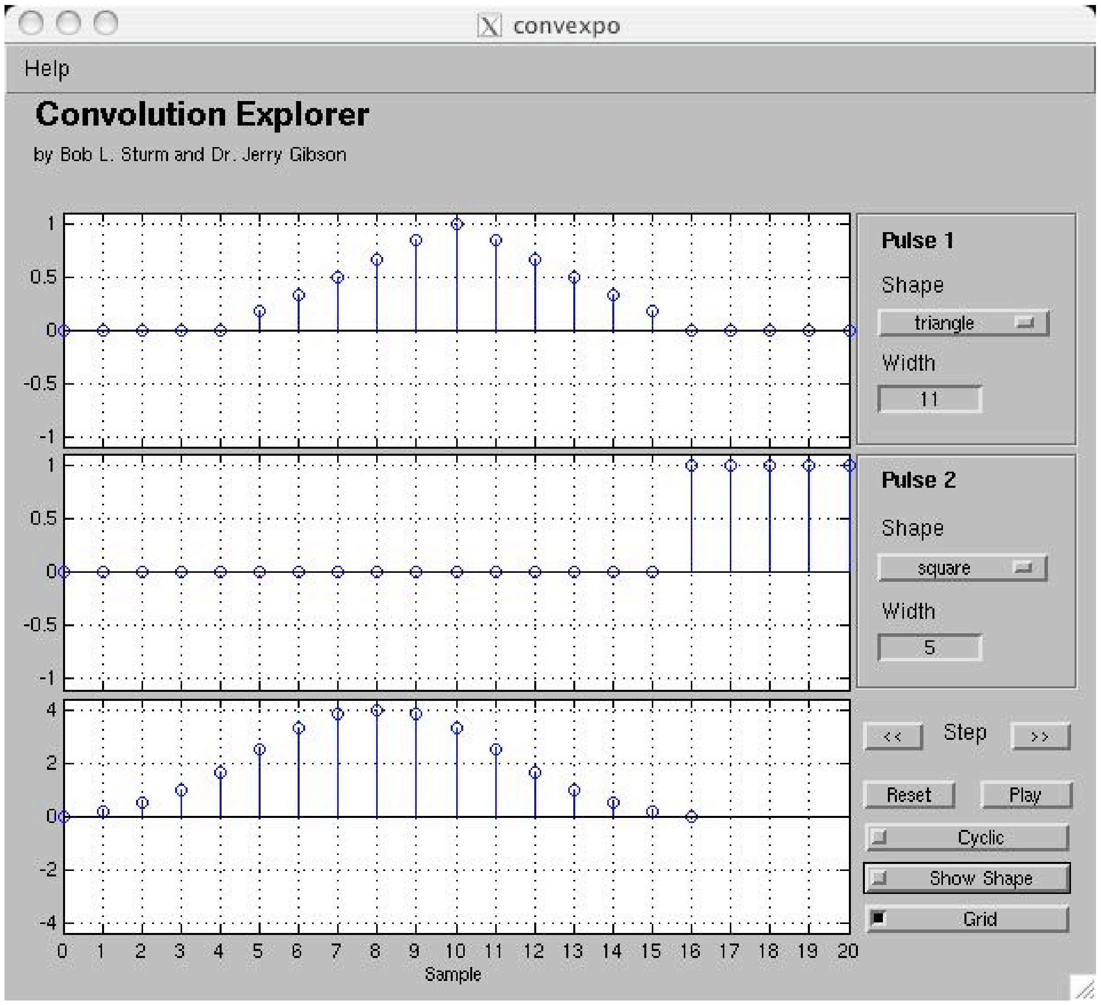
MATConcat is a complex application that synthesizes a target sound using samples from a dictionary of sounds. It does this using feature extraction and matching criteria.

Begin by analyzing a target sound using the parameters set in the “Analysis Parameters” box. Load or analyze a corpus sound. Specify matching criteria from the “Synthesis Parameters” box. Click “Synthesize” to begin the process. Once finished the resulting signal will be displayed in the plot on the right, and the matching results will be displayed in the text box below it.

The synthesis can be saved from the File menu.

Convolution Explorer

This is a demonstration of discrete convolution.

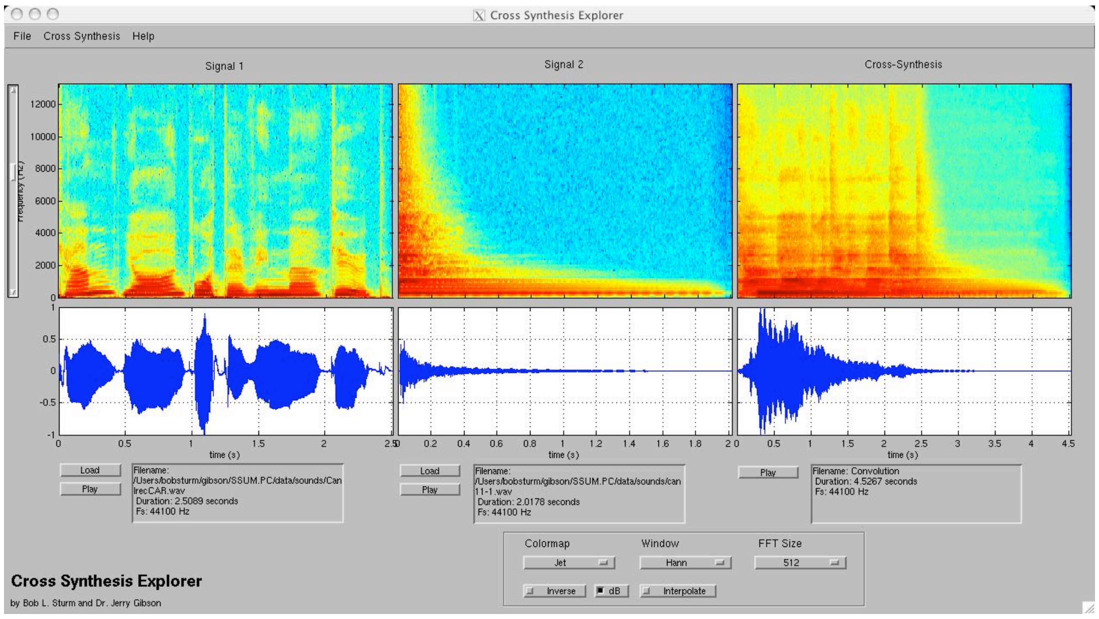


Begin by adjusting the widths of the pulses you wish to convolve. The shape of the pulses can be changed using the shape menu. The convolution can either be stepped through (in any direction), or played as an animation. Once finished it can be stepped in reverse, or reset and played again. Selecting the “Cyclic” box performs cyclic convolution.

By selecting the “Show Shape” box the stem plots will be replaced by line plots. The grid box turns on and off the grids.

Cross-Synthesis Explorer

This application demonstrates the cross-synthesizing of sounds.



Filename:
/Users/bobsturm/gibson/SSUM.PC/data/sounds/Can
lrecCAR.wav
Duration: 2.5089 seconds
Fs: 44100 Hz

Filename:
/Users/bobsturm/gibson/SSUM.PC/data/sounds/can
11-1.wav
Duration: 2.0178 seconds
Fs: 44100 Hz

Filename: Convolution
Duration: 4.5267 seconds
Fs: 44100 Hz

Colormap: Jet
Window: Hann
FFT Size: 512

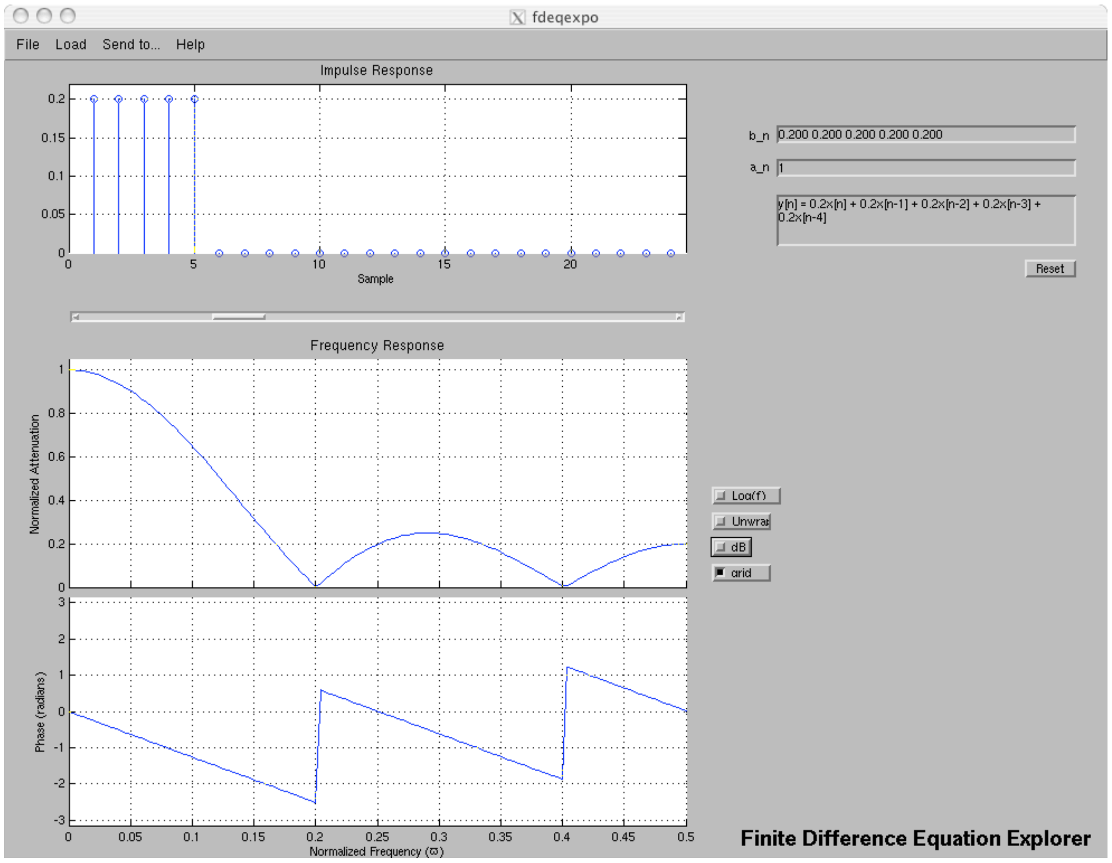
Cross Synthesis Explorer
by Bob L. Sturm and Dr. Jerry Gibson

Begin by loading two signals using the “load” buttons. Once loaded the signal’s sonogram and its time-domain waveform will be displayed. From the “Cross Synthesis” menu select a method. The convolution method just convolves the two signals. The amplitude envelope method applied the RMS envelope of the second signal to the first. The linear predictive coding method creates a model from the first signal, and uses the second signal as the excitation for the model.

The sonogram displays can be changed using the choices at the bottom of the figure. The vertical slider on the left scales the frequency domain of the sonograms. The resulting signal can be saved to a soundfile from the File menu.

Finite Difference Equation Explorer

This application allows the exploration of finite difference equations and their frequency responses.



b_n 0.200 0.200 0.200 0.200 0.200

a_n 1

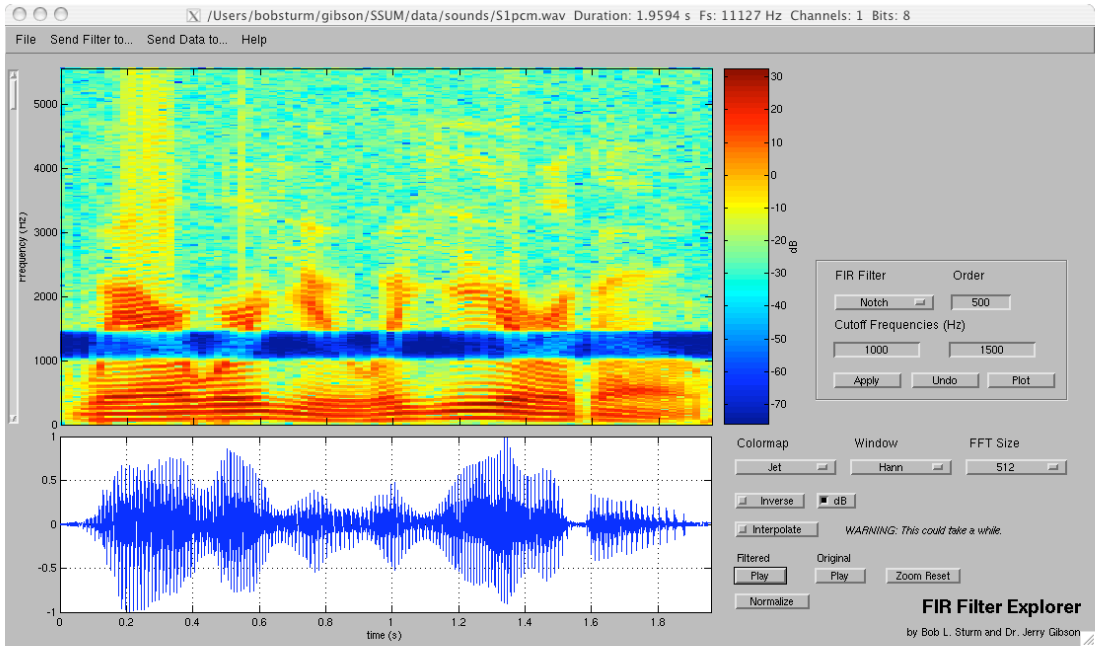
y[n] = 0.2x[n] + 0.2x[n-1] + 0.2x[n-2] + 0.2x[n-3] + 0.2x[n-4]

You can either type in the coefficients in the text boxes, or load a given filter from the menu. The display can be altered using the checkboxes to the right. The slider below the impulse response changes the time-domain display.

You can send the filter to the Pole Zero Explore, the Pole Zero Filter Explorer, or the Convolution Explorer.

FIR Filter Explorer

This application allows FIR filtering of sounds and displays spectral information using a sonogram.

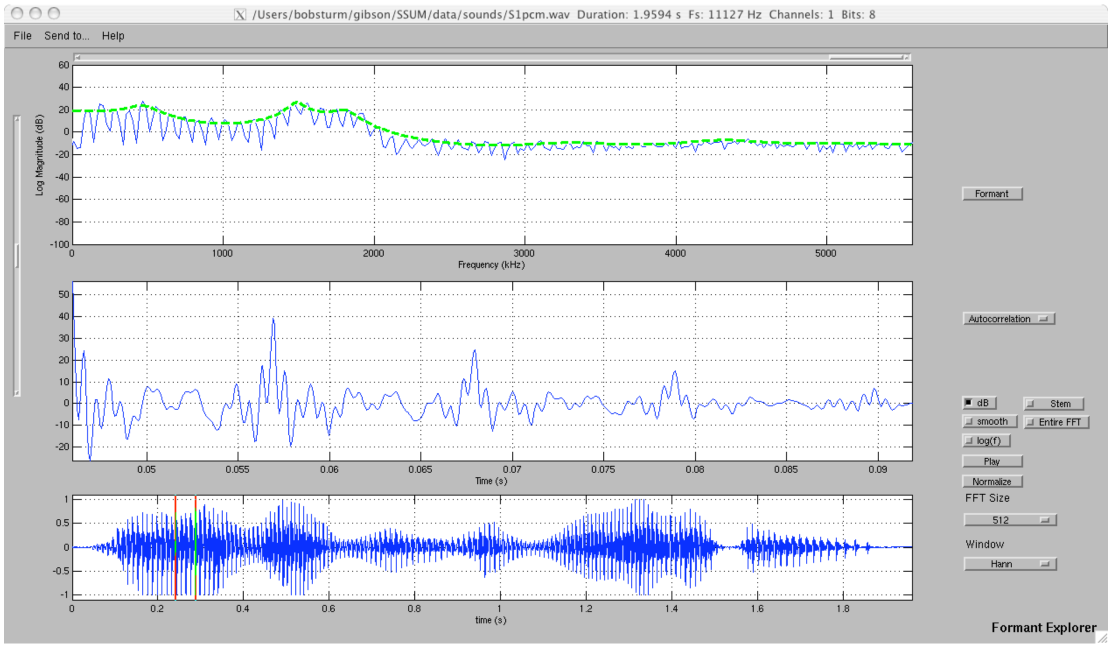


First load a soundfile from the menu. Once analyzed its sonogram will appear. Change the colormap and dB scale to see different features. Zooming in the waveform time-display will also zoom the sonogram. (If at any time this does not work, click on the “zoom reset” button to restore this functionality.) The "Interpolate" box will smooth the sonogram, but this will take a long time to complete for large portions showing—so use wisely. The play buttons will play only the waveform showing. The slider to the left of the sonogram will scale the frequency axis.

Once a sound is loaded it can be filtered using the parameters specified in the box on the right. You can apply and undo these changes. Click on “plot” to see the frequency response and impulse response, of the filter. Using the menu at the top, the filter can be sent to the Pole-Zero Explorer or Pole-Zero Filter Explorer to show the positions of the filter’s poles and zeros. The impulse response can be sent to the Convolution Explorer. The resulting signal can be saved to a soundfile from the File menu.

Formant Explorer

This application displays the spectrum and formants of a sound over a window. Also available for display are the autocorrelation, and cepstrum.


/Users/bobsturm/gibson/SSUM/data/sounds/S1pcm.wav Duration: 1.9594 s Fs: 11127 Hz Channels: 1 Bits: 8

FFT Size 512

Window Hann

After loading a soundfile using the menu at the top, its time-domain waveform will appear in the bottom plot. The two red bars denote the window over which the Fourier transform is performed. The magnitudes of the transform are displayed in the top plot, along with the formants derived from an LPC analysis. The anti-formants can be displayed by clicking on the button to the right of the plot. The vertical slider scales the range of the plot; the horizontal slider scales the frequency domain plotted. By clicking and holding on a red bar the window can be dragged over the sound to show how the formants change.

The middle plot can display the autocorrelation of the windowed signal, or the cepstrum. This can be selected from the pull-down menu.

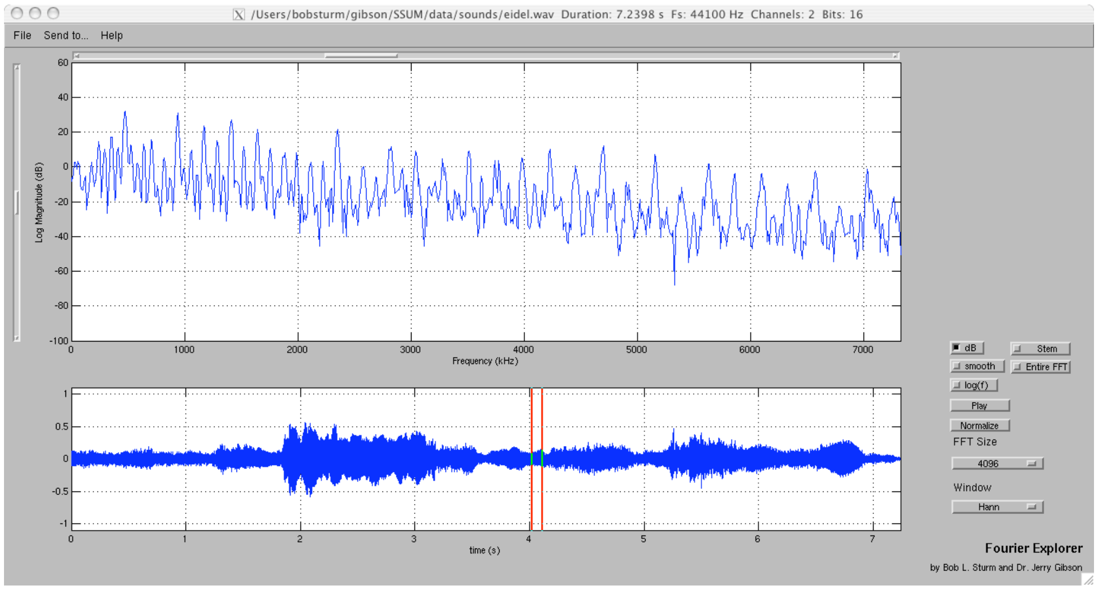
The menus on the lower right side of the figure can change the fast Fourier transform window size and shape. The sound can be normalized and played. The check boxes change how the transform is displayed.

Using the menu at the top, the loaded sound data can be sent to other applications, like the Sonogram Explorer, or Aliasing Explorer.

Some code to make this demonstration was taken from the MAD program "wavspect" and “lpc.”

Fourier Explorer

This application displays spectral information using the Fourier transform.



/Users/bobsturm/gibson/SSUM/data/sounds/eidel.wav Duration: 7.2398 s Fs: 44100 Hz Channels: 2 Bits: 16

FFT Size: 4096

Window: Hann

After loading a soundfile using the menu at the top, its time-domain waveform will appear in the bottom plot. The two red bars denote the window over which the Fourier transform is performed. The magnitudes of the transform are displayed in the top plot. The vertical slider scales the range of the plot; the horizontal slider scales the frequency domain plotted. By clicking and holding on a red bar the window can be dragged over the sound to show how the spectrum changes.

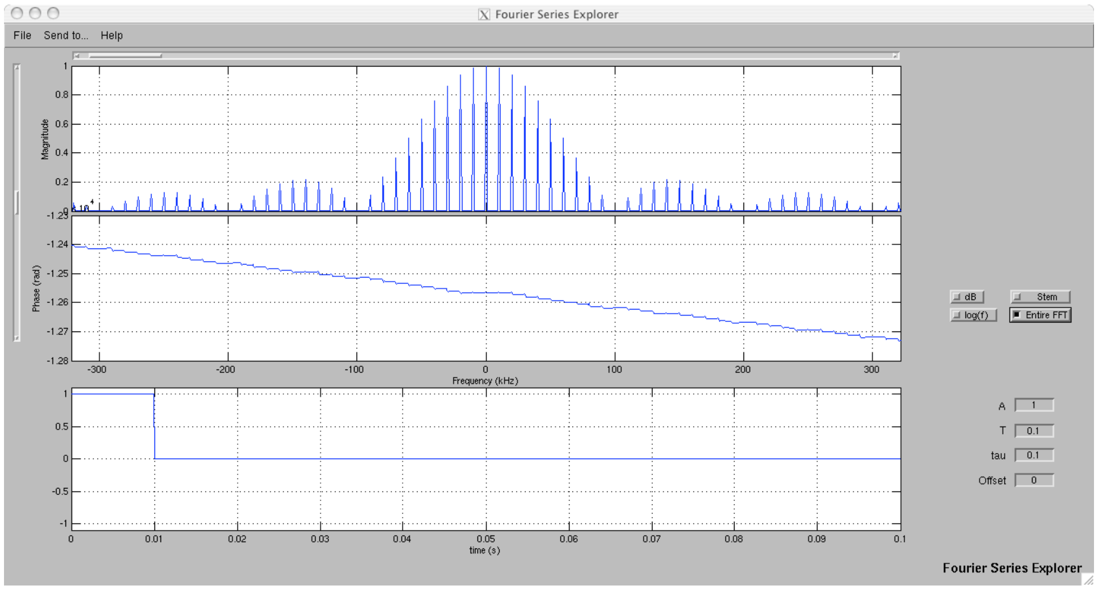
The menus on the lower right side of the figure can change the fast Fourier transform window size and shape. The sound can be normalized and played. The check boxes change how the transform is displayed.

Using the menu at the top, the loaded sound data can be sent to other applications, like the Sonogram Explorer, or Aliasing Explorer.

Some code to make this demonstration was taken from the MAD program "wavspect."

Fourier Series Explorer

This application demonstrates the Fourier series for a periodic step function.



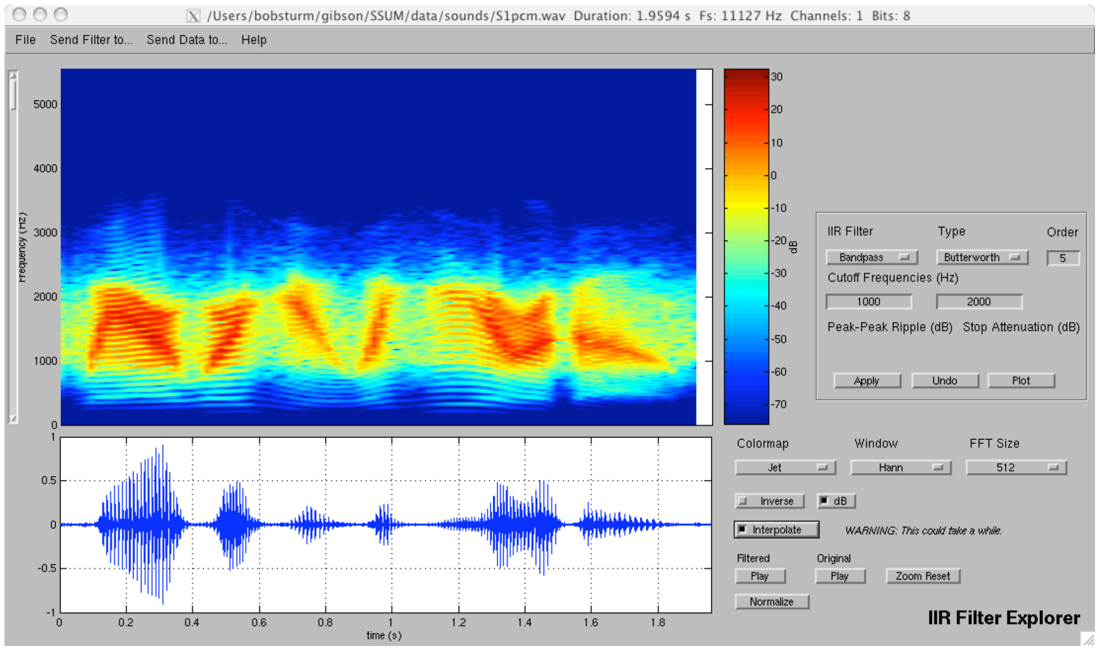
A 1
T 0.1
tau 0.1
Offset 0

By adjusting the period “T” and step duration “tau,” the magnitude Fourier series shown in the top plot changes. The phase spectrum is shown to be linear. The amplitude and offset of the step function can also be changed. The horizontal slider changes the frequency domain displayed; the vertical slider changes the magnitude range.

The generated periodic step signal can be sent to the Fourier Explorer, or the Sonogram Explorer from the menu.

IIR Filter Explorer

This application allows IIR filtering of sounds and displays spectral information using a sonogram.



/Users/bobsturm/gibson/SSUM/data/sounds/S1pcm.wav Duration: 1.9594 s Fs: 11127 Hz Channels: 1 Bits: 8

IIR Filter: Bandpass; Type: Butterworth; Order: 5

Cutoff Frequencies (Hz): 1000, 2000

Colormap: Jet; Window: Hann; FFT Size: 512

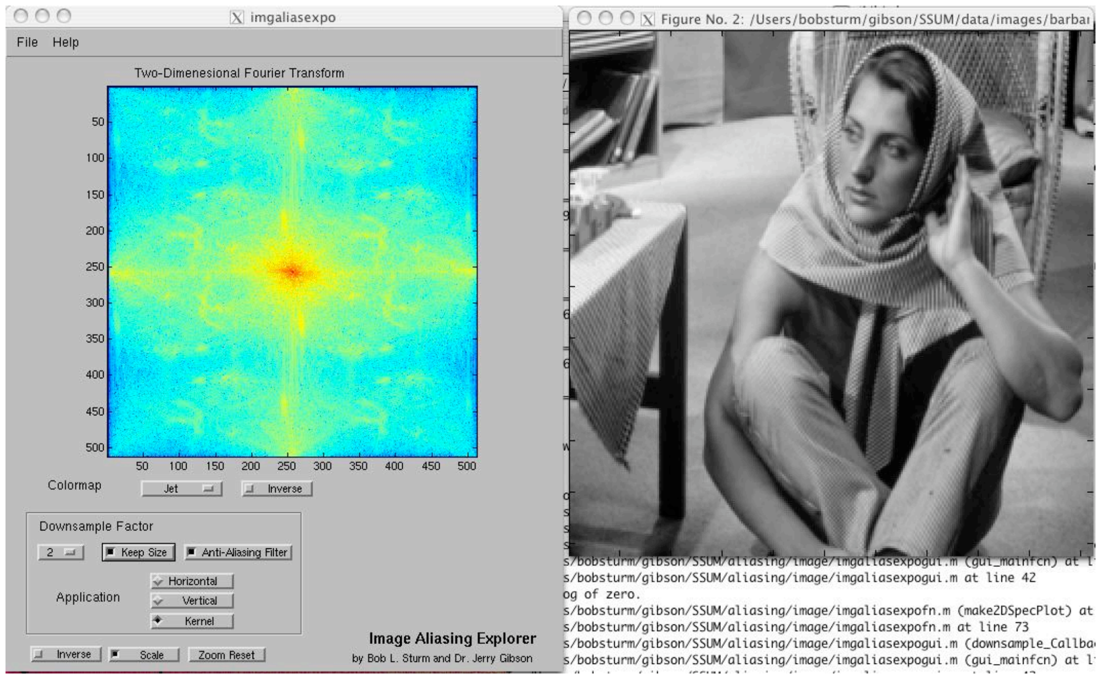
WARNING: This could take a while.

First load a soundfile from the menu. Once analyzed its sonogram will appear. Change the colormap and dB scale to see different features. Zooming in the waveform time-display will also zoom the sonogram. (If at any time this does not work, click on the “zoom reset” button to restore this functionality.) The "Interpolate" box will smooth the sonogram, but this will take a long time to complete for large portions showing—so use wisely. The play buttons will play only the waveform showing. The slider to the left of the sonogram will scale the frequency axis.

Once a sound is loaded it can be filtered using the parameters specified in the box on the right. You can apply and undo these changes. Click on “plot” to see the frequency response and impulse response, of the filter. Using the menu at the top, the filter can be sent to the Pole-Zero Explorer or the Pole-Zero Filter Explorer to show the positions of the filter’s poles and zeros. The impulse response can be sent to the Convolution Explorer. The resulting signal can be saved to a soundfile from the File menu.

Image Aliasing Explorer

This application explores aliasing images.



```
s/bobsturm/gibson/SSUM/aliasing/image/imgaliasexpogui.m (gui_mainfcn) at l
s/bobsturm/gibson/SSUM/aliasing/image/imgaliasexpogui.m at line 42
og of zero.
s/bobsturm/gibson/SSUM/aliasing/image/imgaliasexpofn.m (make2DSpecPlot) at
s/bobsturm/gibson/SSUM/aliasing/image/imgaliasexpofn.m at line 73
s/bobsturm/gibson/SSUM/aliasing/image/imgaliasexpogui.m (downsample_Callba
s/bobsturm/gibson/SSUM/aliasing/image/imgaliasexpogui.m (gui_mainfcn) at l
```

Like the Aliasing Explorer for sound, this demonstration creates aliasing in images. Start by loading an image. It will be displayed at the correct screen resolution; its two-dimensional Fourier transform is displayed in the GUI figure. The appearance of the transform can be changed using the options underneath the plot. The appearance of the image can be changed using the options at the bottom of the GUI.

By selecting a downsample factor, the image will be decimated in either the horizontal or vertical directions, or using a kernel—depending on which is checked. When “Keep Size” is checked the image will be upsampled back to the original size. When the “Anti-Aliasing Filter” is checked, the image will be lowpass filtered before decimation.

Using the menu, the image can be sent to the Image Spectrum Explorer or the Image Filter Explorer. The resulting image can be saved to an image file from the File menu.

Image Analysis/Resynthesis Explorer

This demonstration shows the spectral analysis of an image, and its resynthesis from this data.

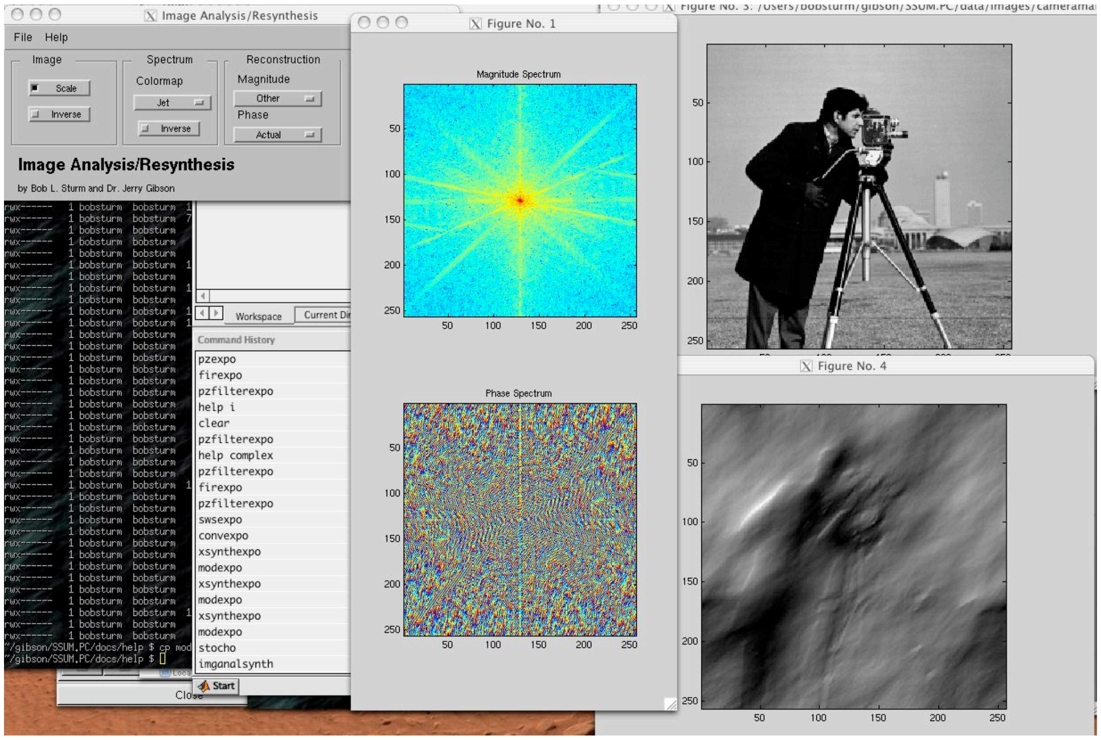


Image: [✓ Scale] Inverse
Spectrum — Colormap: Jet; Inverse
Reconstruction — Magnitude: Other; Phase: Actual

Image Analysis/Resynthesis

by Bob L. Sturm and Dr. Jerry Gibson

```
rwx------   1 bobsturm  bobsturm  1
rwx------   1 bobsturm  bobsturm  7
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm  1
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm  1
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm  1
rwx------   1 bobsturm  bobsturm  1
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm  1
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm  1
rwx------   1 bobsturm  bobsturm
rwx------   1 bobsturm  bobsturm
~/gibson/SSUM.PC/docs/help $ cp mod
~/gibson/SSUM.PC/docs/help $ 
```

```
pzexpo
firexpo
pzfilterexpo
help i
clear
pzfilterexpo
help complex
pzfilterexpo
firexpo
pzfilterexpo
swsexpo
convexpo
xsynthexpo
modexpo
xsynthexpo
modexpo
xsynthexpo
modexpo
stocho
imganalsynth
```

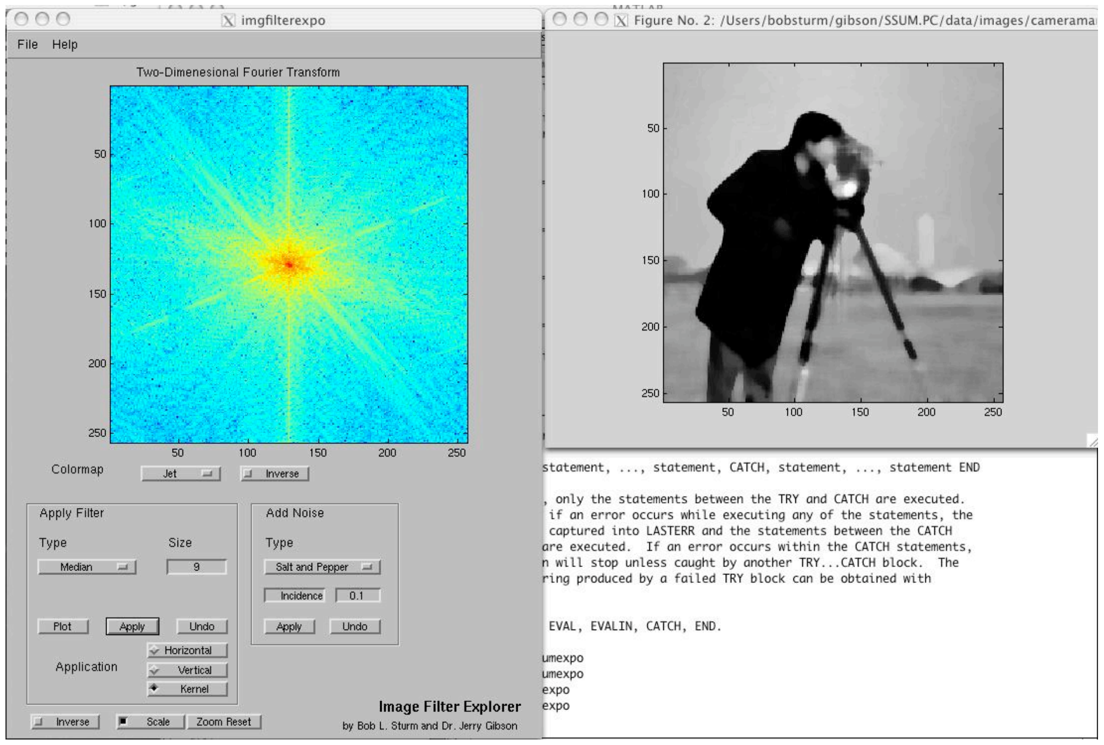
When an image is loaded, it is displayed, a two-dimensional Fourier transform is performed, and the image is synthesized from the spectral data. The original image is seen above in the top right corner; the synthesis is shown below it. The frequency and phase information of the original is shown in the center plots. The appearance of the images can be changed using the option in the “Image” box in the GUI. The appearance of spectral plots can be changed using the options in the Spectrum box of the GUI.

The options in the Reconstruction box affect how the image is synthesized. The magnitudes used to perform the reconstruction can be set to either the original values, random values, or values from the analysis of a different image. The same is true for the phase information, except another choice is to set the phases to zero.

The resulting image can be saved to an image file from the File menu.

Image Filter Explorer

This application explores filtering images. One can use several types of filters to see their effects. One can also add noise to an image and see the effectiveness of a filter for removing it.



```
statement, ..., statement, CATCH, statement, ..., statement END

, only the statements between the TRY and CATCH are executed.
 if an error occurs while executing any of the statements, the
 captured into LASTERR and the statements between the CATCH
are executed.  If an error occurs within the CATCH statements,
n will stop unless caught by another TRY...CATCH block.  The
ring produced by a failed TRY block can be obtained with


 EVAL, EVALIN, CATCH, END.

umexpo
umexpo
expo
expo
```

Start by loading an image. It will be displayed at the correct screen resolution; its two-dimensional Fourier transform is displayed in the GUI figure. The appearance of the transform can be changed using the options underneath the plot. The appearance of the images can be changed using the options at the bottom of the GUI.

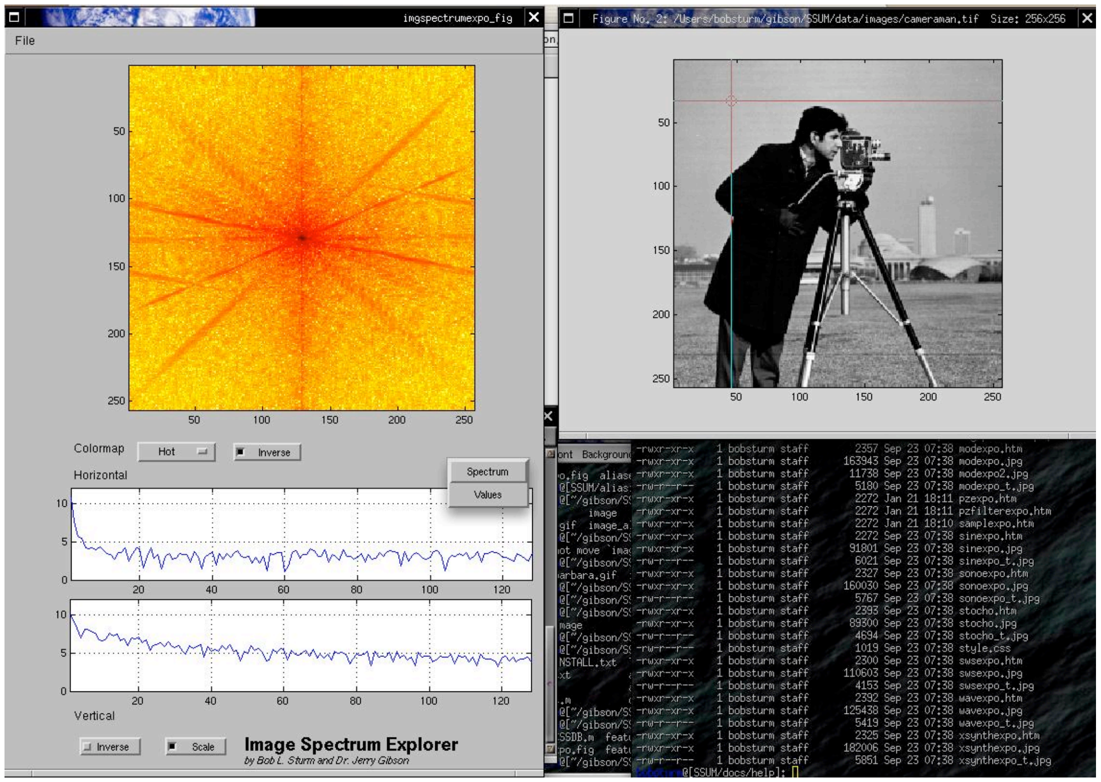
After loading an image it can be filtered. The filter type can be selected, and its size can be set. Choose a direction for its application and press “Apply” to apply the filter. The changes can be undone.

Two types of noise can be applied to the image to see their effects. The incidence of the “Salt and Pepper” gives the percent chance of either black or white pixels. The variance of the “Gaussian” noise gives the spread of noise within a grey scale.

Using the menu, the image can be sent to the Image Spectrum Explorer or the Image Aliasing Explorer. The resulting image can be saved to an image file from the File menu.

Image Spectrum Explorer

The spectral information of an image, or its spatial frequencies, can be explored in this application.


imgspectrumexpo_fig
File
Figure No. 2: /Users/bobsturm/gibson/SSUM/data/images/cameraman.tif Size: 256x256
Colormap
Hot
Inverse
Horizontal
Spectrum
Values
Vertical
Inverse
Scale
Image Spectrum Explorer
by Bob L. Sturm and Dr. Jerry Gibson

```
-rwxr-xr-x   1 bobsturm staff      2357 Sep 23 07:38 modexpo.htm
-rwxr-xr-x   1 bobsturm staff    163943 Sep 23 07:38 modexpo.jpg
-rwxr-xr-x   1 bobsturm staff     11738 Sep 23 07:38 modexpo2.jpg
-rw-r--r--   1 bobsturm staff      5180 Sep 23 07:38 modexpo_t.jpg
-rwxr-xr-x   1 bobsturm staff      2272 Jan 21 18:11 pzexpo.htm
-rwxr-xr-x   1 bobsturm staff      2272 Jan 21 18:11 pzfilterexpo.htm
-rwxr-xr-x   1 bobsturm staff      2272 Jan 21 18:10 samplexpo.htm
-rwxr-xr-x   1 bobsturm staff      2272 Sep 23 07:38 sinexpo.htm
-rwxr-xr-x   1 bobsturm staff     91801 Sep 23 07:38 sinexpo.jpg
-rw-r--r--   1 bobsturm staff      6021 Sep 23 07:38 sinexpo_t.jpg
-rwxr-xr-x   1 bobsturm staff      2327 Sep 23 07:38 sonoexpo.htm
-rwxr-xr-x   1 bobsturm staff    160030 Sep 23 07:38 sonoexpo.jpg
-rw-r--r--   1 bobsturm staff      5767 Sep 23 07:38 sonoexpo_t.jpg
-rwxr-xr-x   1 bobsturm staff      2393 Sep 23 07:38 stocho.htm
-rwxr-xr-x   1 bobsturm staff     89300 Sep 23 07:38 stocho.jpg
-rw-r--r--   1 bobsturm staff      4694 Sep 23 07:38 stocho_t.jpg
-rw-r--r--   1 bobsturm staff      1019 Sep 23 07:38 style.css
-rwxr-xr-x   1 bobsturm staff      2300 Sep 23 07:38 swsexpo.htm
-rwxr-xr-x   1 bobsturm staff    110603 Sep 23 07:38 swsexpo.jpg
-rw-r--r--   1 bobsturm staff      4153 Sep 23 07:38 swsexpo_t.jpg
-rwxr-xr-x   1 bobsturm staff      2392 Sep 23 07:38 wavexpo.htm
-rwxr-xr-x   1 bobsturm staff    125438 Sep 23 07:38 wavexpo.jpg
-rw-r--r--   1 bobsturm staff      5419 Sep 23 07:38 wavexpo_t.jpg
-rwxr-xr-x   1 bobsturm staff      2325 Sep 23 07:38 xsynthexpo.htm
-rwxr-xr-x   1 bobsturm staff    182006 Sep 23 07:38 xsynthexpo.jpg
-rw-r--r--   1 bobsturm staff      5851 Sep 23 07:38 xsynthexpo_t.jpg
bobsturm@[SSUM/docs/help]:
```

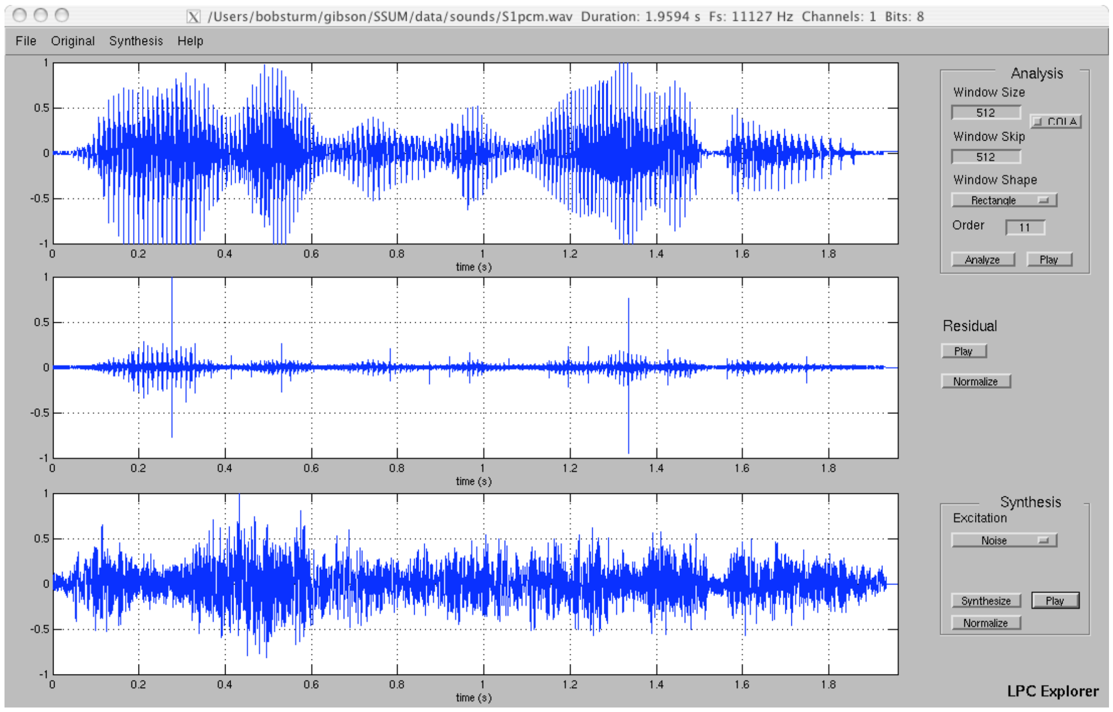
Start by loading an image. It will be displayed at the correct screen resolution; its two-dimensional Fourier transform is displayed in the GUI figure. The appearance of the transform can be changed using the options underneath the plot. The appearance of the images can be changed using the options at the bottom of the GUI.

In the figure containing the loaded image a circle and two lines appears. The circle can be moved around the image. As this happens the two plots in the main figure will correspond to the pixel values in the horizontal and vertical directions. Either the spatial frequencies or the actual pixel values can be displayed.

Using the menu, the image can be sent to the Image Aliasing Explorer or the Image Filter Explorer.

LPC Explorer

This demonstration shows the process of linear predictive coding.



/Users/bobsturm/gibson/SSUM/data/sounds/S1pcm.wav Duration: 1.9594 s Fs: 11127 Hz Channels: 1 Bits: 8

Window Size 512

Window Skip 512

Window Shape Rectangle

Order 11

Excitation Noise

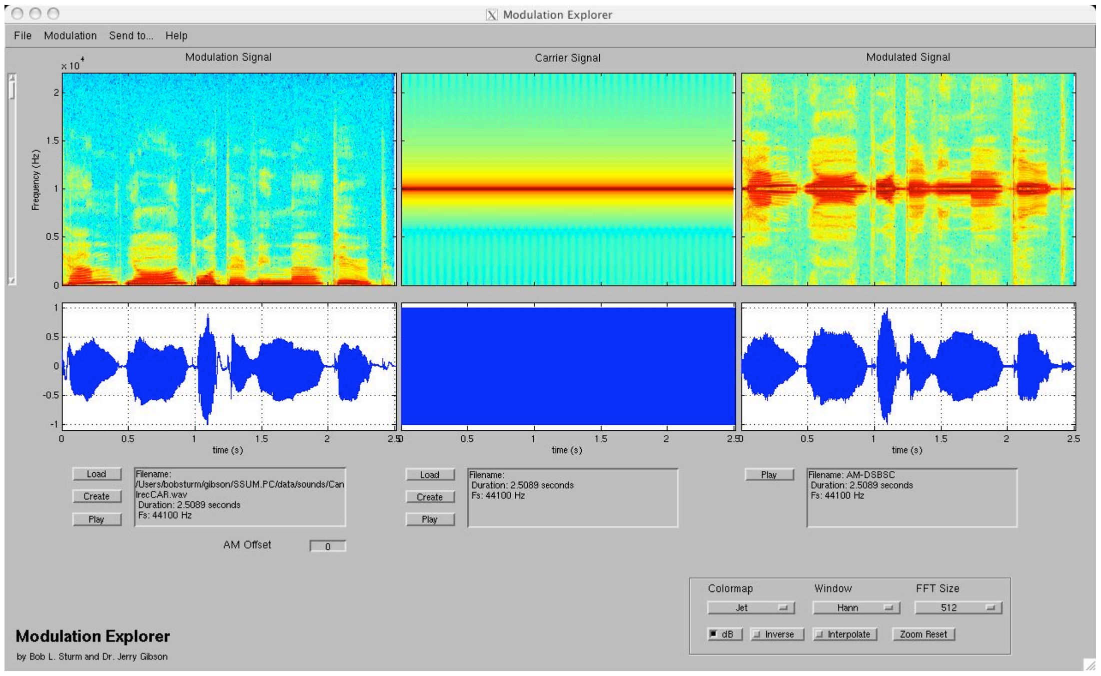
Begin by loading a sound. Its time-domain waveform is shown in the top plot. Select the analysis parameters and click “Analyze.” Once finished the residual signal will be shown in the middle plot. Now select the synthesis parameters and click “Synthesize.” Once finished the synthesis will appear in the bottom plot. Either sound can be played using the respective buttons.

The model filter can be excited using numerous signals including the residual, white noise, impulses, or a sound file.

You can send either the original or the synthesized signal to the Fourier Explorer or Sonogram Explorer. The result can be saved to a sound file from the File menu.

Modulation Explorer

This application demonstrates the amplitude modulation of two signals.



Filename:
/Users/bobsturm/gibson/SSUM.PC/data/sounds/Can
lrecCAR.wav
Duration: 2.5089 seconds
Fs: 44100 Hz

AM Offset 0

Filename:
Duration: 2.5089 seconds
Fs: 44100 Hz

Filename: AM-DSBSC
Duration: 2.5089 seconds
Fs: 44100 Hz

Colormap: Jet
Window: Hann
FFT Size: 512

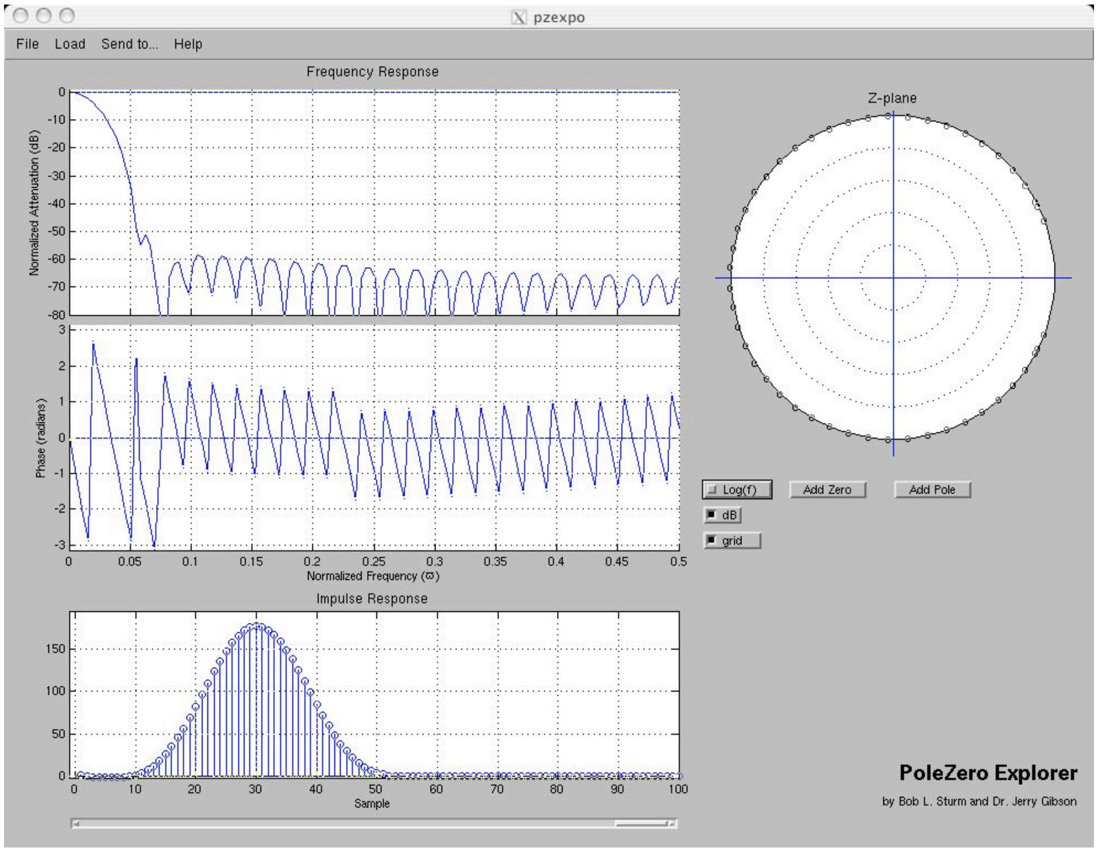
Begin by loading or creating modulation and carrier signals. If you select “Create,” a GUI will appear from which you can synthesize a signal with frequency and amplitude envelopes. Once loaded the signal’s sonogram and its time-domain waveform will be displayed.

From the modulation menu, the two signals are amplitude modulation using either double side-band suppressed or transmitted carrier. The resulting signal is analyzed and displayed in the plot on the right. The modulation signal can be sent to the Fourier Explorer from the menu.

The sonogram displays can be changed using the choices at the bottom of the figure. The vertical slider on the left scales the frequency domain of the sonograms. The resulting signal can be saved to a soundfile from the File menu.

Pole-Zero Explorer

This application allows interactive exploration of building filters using poles and zeros.



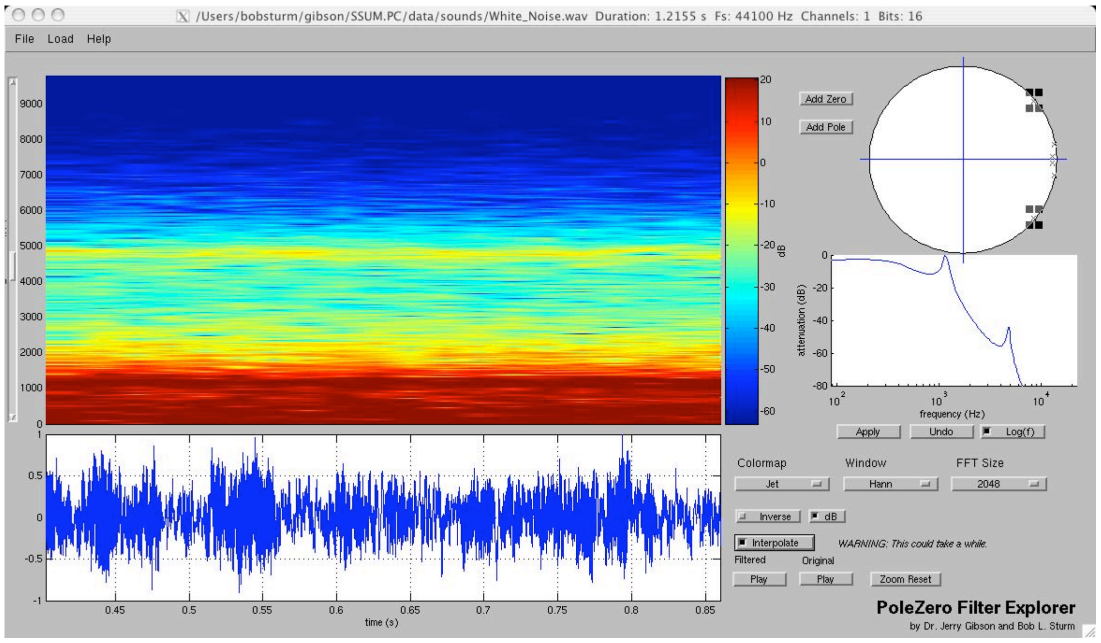
Within this environment you can see how the positions of poles and zeros affect the frequency and phase response of filters. You can start by either clicking on “Add Pole” or “Add Zero,” or by loading a pre-designed filter from the menu. The pre-designed filters include formant shapes. You can click on any pole or zero and move it around the plot. The frequency response and impulse response will change in real-time. The slider on the bottom scales the plot duration of the impulse response.

To apply this filter to a sound you can send it to the Pole-Zero Filter Explorer using the menu. You can send the impulse response to the Convolution Explorer.

Some code to make this demonstration was taken from the MAD program "polezero."

Pole-Zero Filter Explorer

This application allows interactive exploration of building filters using poles and zeros and applying it to a sound.



Within this environment you can apply filters constructed of poles and zeros to a sound. You can start by either clicking on “Add Pole” or “Add Zero,” or by loading a pre-designed filter from the menu. The pre-designed filters include formant shapes. You can click on any pole or zero and move it around the plot. The frequency response will change in real-time.

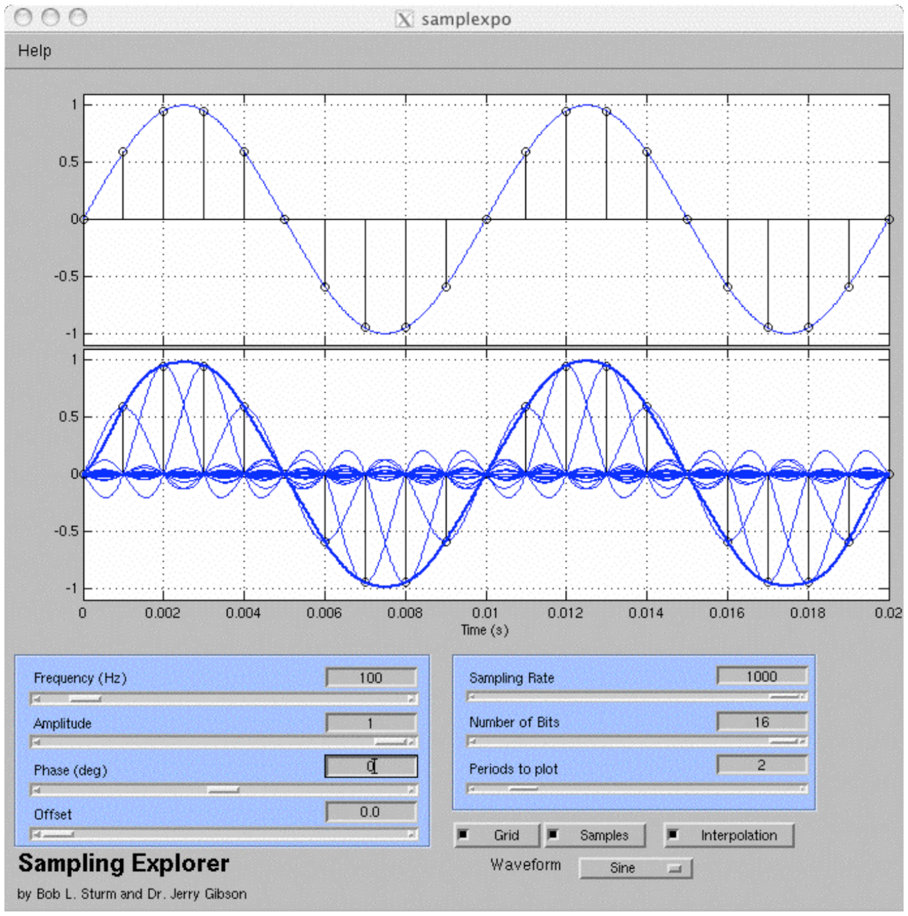
When you load a soundfile from the menu it will be analyzed and its sonogram will appear. Change the colormap and dB scale to see different features. Zooming in the waveform time-display will also zoom the sonogram. (If at any time this does not work, click on the “zoom reset” button to restore this functionality.) The "Interpolate" box will smooth the sonogram, but this will take a long time to complete for large portions showing—so use wisely. The play buttons will play only the waveform showing. The slider to the left of the sonogram will scale the frequency axis.

Once a sound is loaded it can be filtered. You can apply and undo these changes. Using the menu at the top, the filter can be sent to the Pole-Zero Explorer to show the phase response and impulse response. The resulting signal can be saved to a soundfile from the File menu.

Some code to make this demonstration was taken from the MAD program "polezero."

Sampling Explorer

This is a visual demonstration of sampling analog signals, including time sampling, quantization, and interpolation of the samples.



Frequency (Hz) 100
Amplitude 1
Phase (deg) 0
Offset 0.0
Sampling Rate 1000
Number of Bits 16
Periods to plot 2

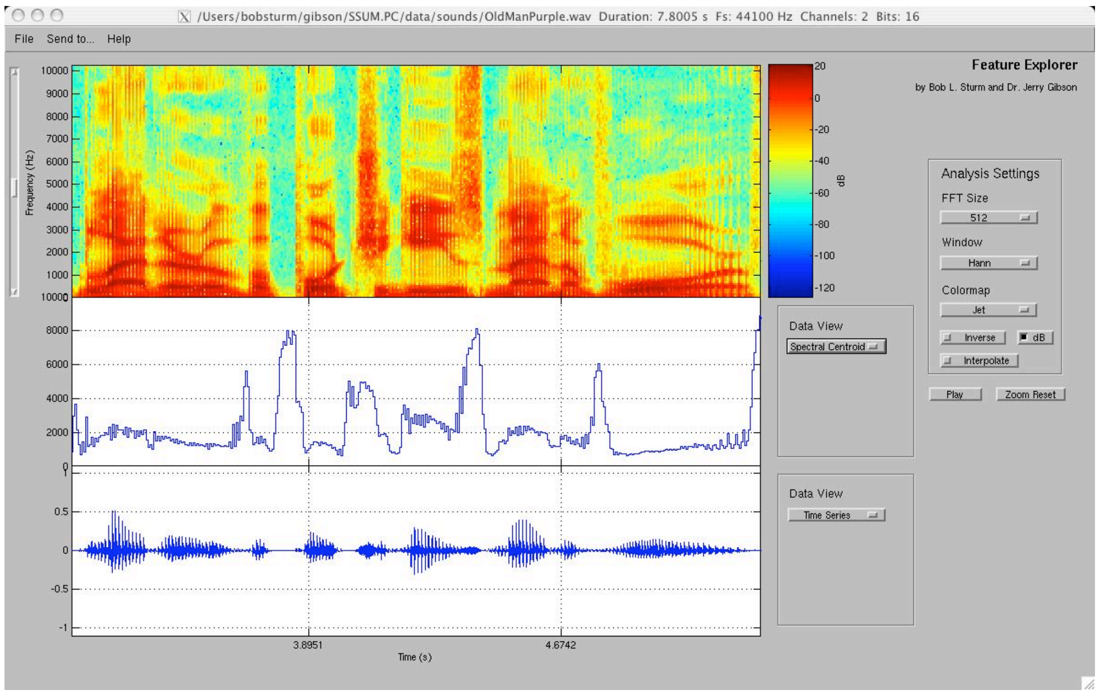
The original analog signal is the blue line in the top plot. The stems, or lollypops, denote the samples. The dark blue line in the bottom plot is the interpolation of the samples back to a continuous signal. You can choose different analog signals from the waveform menu at the bottom.

The frequency, amplitude, phase, and offset of the original signal can be changed using the sliders and text boxes on the left. The sampling rate and number of bits for each sample can be changed using the sliders and text boxes on the right. The number of plotted periods can also be changed to reduce the effect of edge effects.

Using the check boxes, the grids, samples, and sinc interpolation lines can be disabled.

Signal Feature Explorer

This demonstration shows a statistical analysis of sounds.



/Users/bobsturm/gibson/SSUM.PC/data/sounds/OldManPurple.wav Duration: 7.8005 s Fs: 44100 Hz Channels: 2 Bits: 16

Analysis Settings
FFT Size: 512
Window: Hann
Colormap: Jet

Data View: Spectral Centroid

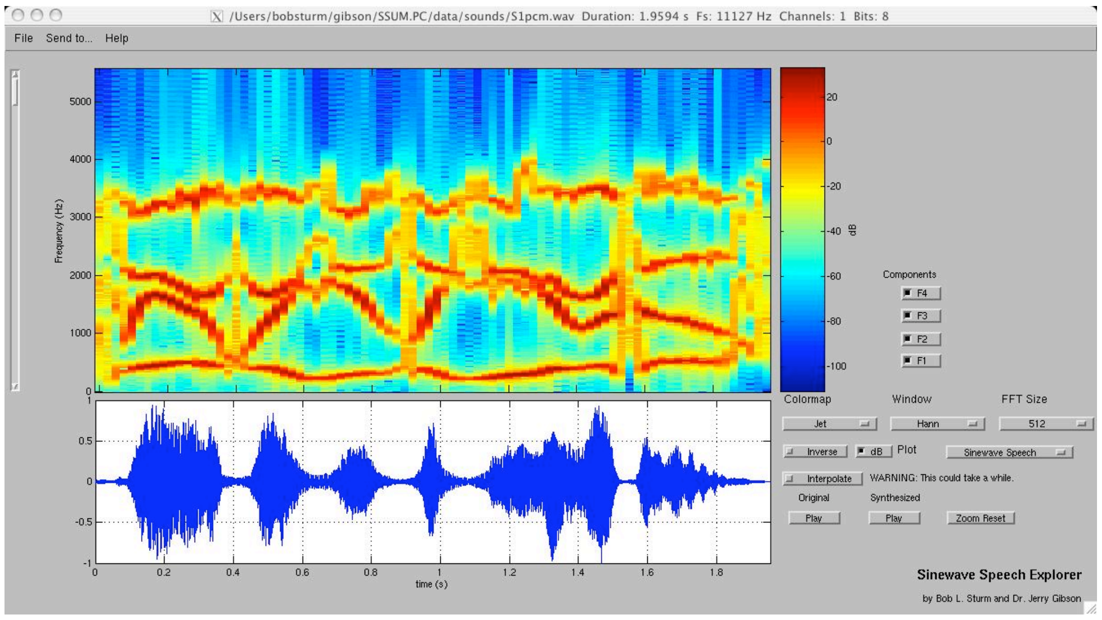
Data View: Time Series

Begin by loading a signal from the File menu. Once loaded the signal will be analyzed. The sonogram of the signal is displayed at top, and its time-domain waveform is at the bottom. The appearance of the sonogram can be changed with the options on the right. The vertical scroll bar changes the frequency limits of the sonogram.

The two plots below the sonogram can show two of seven things: samples, number of zero crossings, RMS, spectral centroid, spectral roll-off, harmonicity, and pitch. The resolution of the analysis is determined by the FFT size option on the right.

Sinewave Speech Explorer

This is a demonstration of sine wave speech.


/Users/bobsturm/gibson/SSUM.PC/data/sounds/S1pcm.wav Duration: 1.9594 s Fs: 11127 Hz Channels: 1 Bits: 8

Colormap: Jet
Window: Hann
FFT Size: 512
Plot: Sinewave Speech

WARNING: This could take a while.

Sinewave Speech Explorer

by Bob L. Sturm and Dr. Jerry Gibson

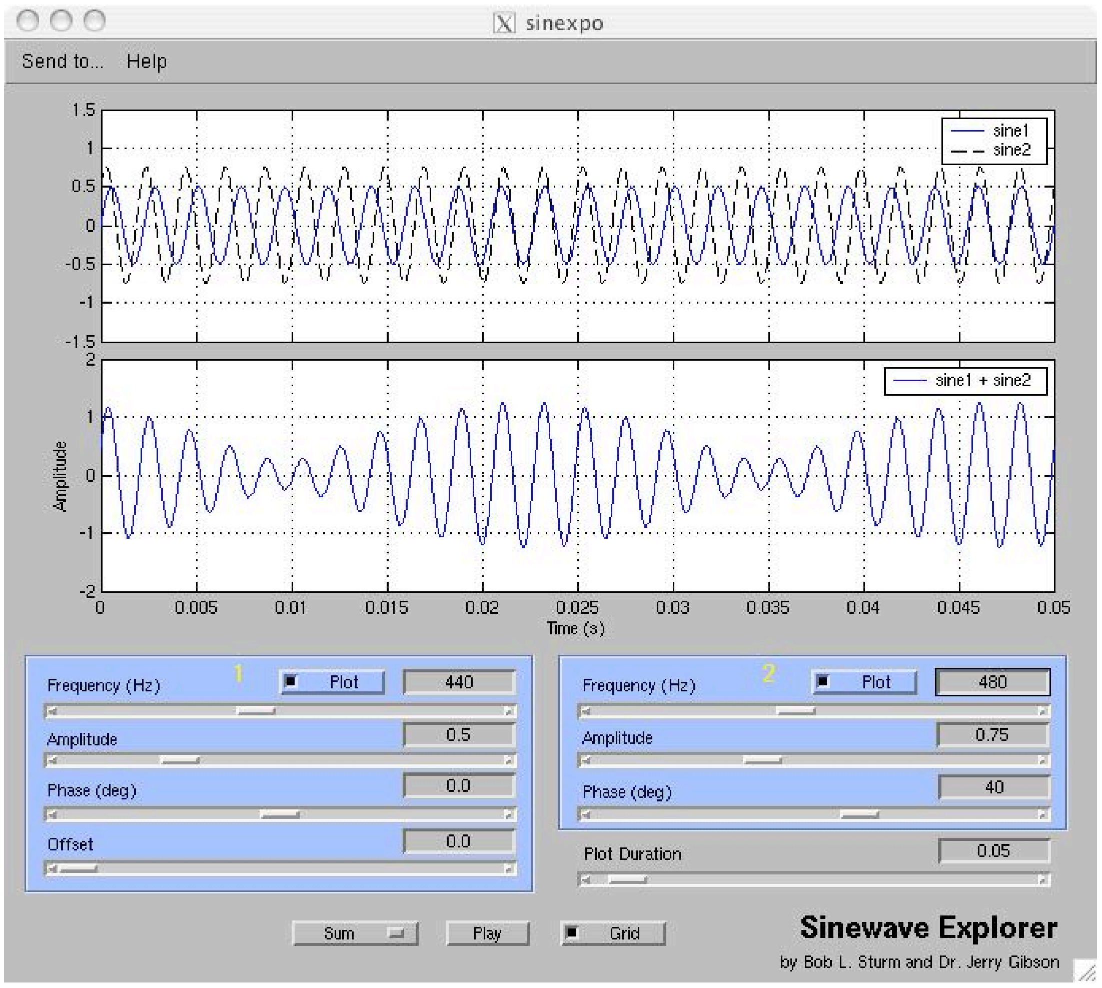
Begin by loading a sound, speech or not. An LPC analysis will be performed and reduced to four sinusoids with dynamic frequencies and amplitudes. The sonogram of the synthesis will be displayed, and its time-domain waveform will be shown below. The appearance of the sonogram can be changed using the options to the right. The sonogram of the original can also be displayed.

The synthesis and original can be played using the respective buttons. The four check boxes to the right of the sonogram enable the sine wave components in the synthesis.

Some code to make this demonstration was taken from the MAD program "sws." Code was also used from http://www.ee.columbia.edu/~dpwe/resources/matlab/sws/.

Sinusoidal Explorer

This is a demonstration of sinusoids and their parameters: frequency, phase, amplitude, and offset.



1 Plot
Frequency (Hz) 440
Amplitude 0.5
Phase (deg) 0.0
Offset 0.0

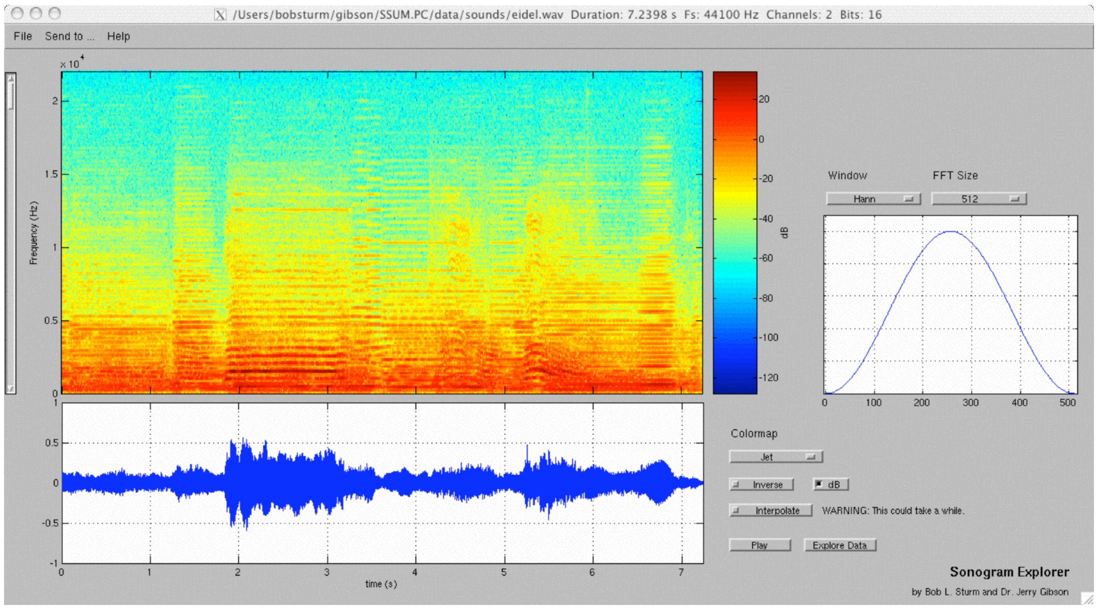
2 Plot
Frequency (Hz) 480
Amplitude 0.75
Phase (deg) 40

Plot Duration 0.05

You can sum or multiply two sine waves to see the different effects. The top plot displays the two sine waves. The “plot” check boxes enable or disable their waveform display. The resulting signal is shown in the lower plot. This can be sent to the Sonogram Explorer or the Fourier Explorer.

Sonogram Explorer

This application displays spectral information using a sonogram, or Short-Time Fourier Transform (STFT).



/Users/bobsturm/gibson/SSUM.PC/data/sounds/eidel.wav Duration: 7.2398 s Fs: 44100 Hz Channels: 2 Bits: 16

Window: Hann

FFT Size: 512

Colormap: Jet

WARNING: This could take a while.

Sonogram Explorer

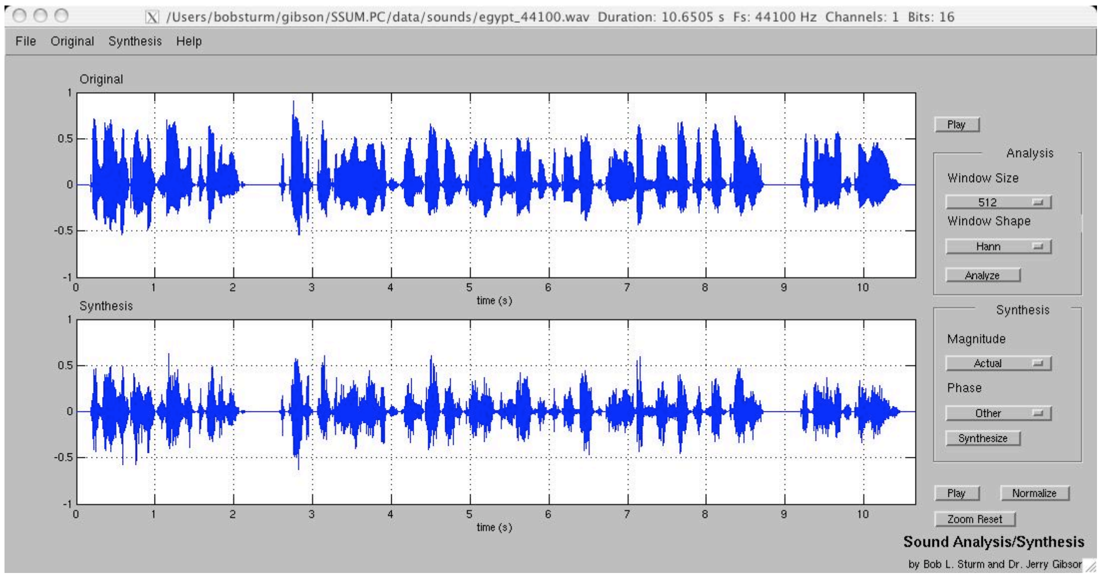
by Bob L. Sturm and Dr. Jerry Gibson

First load a soundfile from the menu. Once analyzed its sonogram will appear. Change the colormap and dB scale to see different features. Zooming in the waveform time-display will also zoom the sonogram. (If at any time this does not work, click on the “zoom reset” button to restore this functionality.) The "Interpolate" box will smooth the sonogram, but this will take a long time to complete for large portions showing—so use wisely. The play buttons will play only the waveform showing. The slider to the left of the sonogram will scale the frequency axis.

You can explore the effects of different window shapes and sizes on the resolution of the sonogram. The “Explore Data” button allows one to travel the partials in the sonogram to get detailed information about its frequency and amplitude envelopes, and the time information. After clicking the button, click on the sonogram several times. When finished hit “enter.” The frequencies, amplitudes, and times at each click will be printed in the MATLAB command window. These can be used with the additive synthesis program to attempt to synthesize the sound. This works especially well with birdsong (see Additive Synthesis Forest).

Sound Analysis/Resynthesis Explorer

This demonstration shows the spectral analysis of a sound, and its resynthesis using this data.


/Users/bobsturm/gibson/SSUM.PC/data/sounds/egypt_44100.wav Duration: 10.6505 s Fs: 44100 Hz Channels: 1 Bits: 16

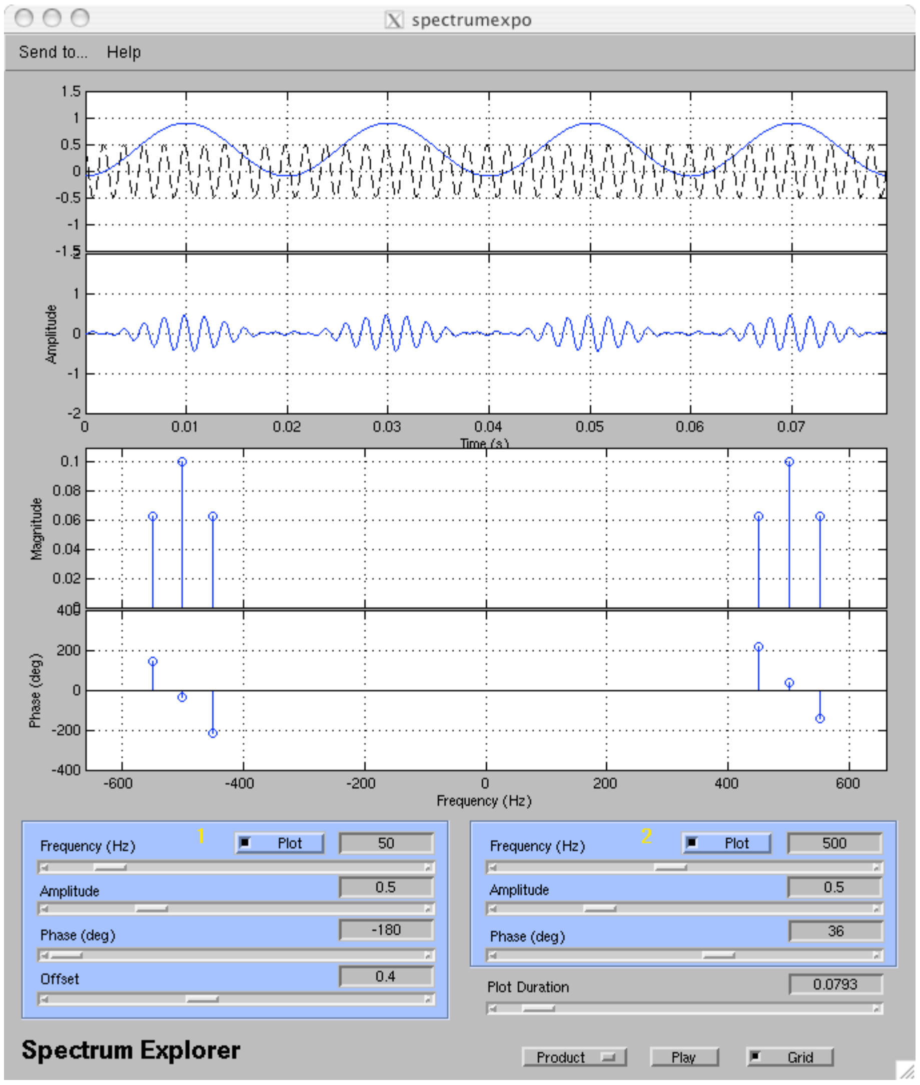
Begin by loading a sound. Its time-domain waveform is shown in the top plot. Select the analysis parameters and click “Analyze.” Now select the synthesis parameters and click “Synthesize.” Once finished the synthesis will appear in the bottom plot. Either sound can be played using the respective buttons.

As in the Image Analysis/Resynthesis Explorer, the magnitudes and phases of the analysis can be altered. The magnitudes can be set to the original, randomized, or from the analysis of another sound file. The phases can be set to the original, zeros, randomized, or from the analysis of another sound.

You can send either the original or the synthesized signal to the Fourier Explorer or Sonogram Explorer. The result can be saved to a sound file from the File menu.

Spectrum Explorer

This application demonstrates the frequency domain interpretation of pure signals.



Frequency (Hz) 1: 50
Amplitude: 0.5
Phase (deg): -180
Offset: 0.4

Frequency (Hz) 2: 500
Amplitude: 0.5
Phase (deg): 36

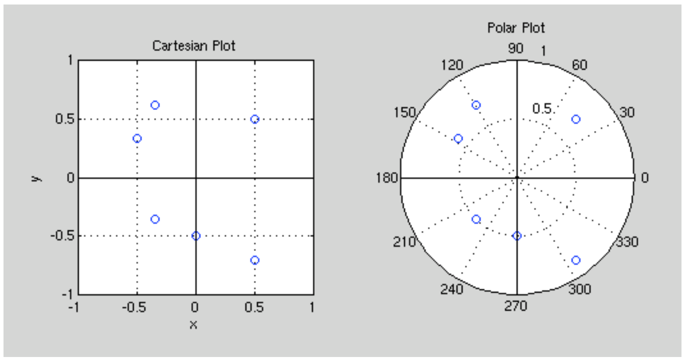
Plot Duration: 0.0793

You can sum or multiply two sine waves to see the different effects, and the representation of them in the frequency domain. The top plot displays the two sine waves. The "plot" check boxes enable or disable their waveform display. The resulting signal is shown in the plot below this. The two plots below these show the magnitude and phase spectrum of the resulting signal. The resulting signal can be sent to the Sonogram Explorer or the Fourier Explorer.

Appendix B: Assignments for 201A

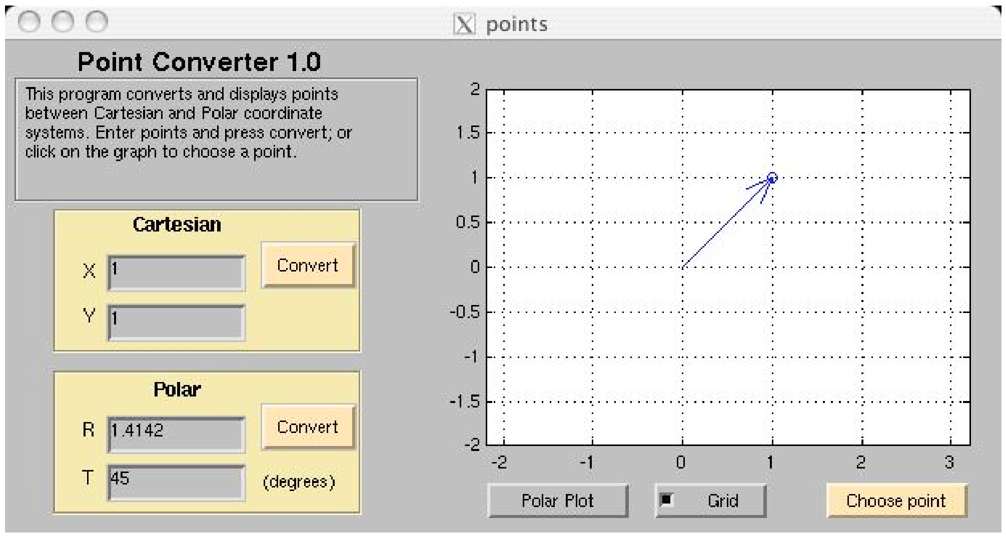
Homework #1

1. Consider the complex number: z = x + j*y. This number can be represented as the complex polar number: z = r*exp(j*theta). Using MATLAB create a **function** named *carpol.m* that will take as arguments x, y and return r, theta (in radians). Test it with the following numbers:
   a. z = 0.5 + j*0.5
   b. z = 0.5 – j*sqrt(2)/2
   c. z = -0.5 + j*0.33
2. Create another **function** named *polcar.m* that will convert polar to Cartesian. It should take as arguments r, theta (in radians) and return x, y. Test it with the following complex numbers:
   a. z = 0.5*exp(-j*pi/2)
   b. z = 1/2*exp(-j*3*pi/4)
   c. z = sqrt(2)/2*exp(j*2*pi/3)
3. Plot these points on two graphs in the same figure using “subplot.” Use the command “plot” for the first plot and the command “polar” for the other plot. Use the marker ‘o’ for each point. If your functions are working correctly the two graphs will be exactly the same, i.e. the points will be in the same locations. (Make the limits of the Cartesian graph centered on 0, and set the axis to be square, i.e. “axis square.”) It should look like the following:



When completed and corrected e-mail the two functions, *carpol.m* and *polcar.m*, and the m-file used to create the plots in part 3, to the T.A. b.sturm@mat.ucsb.edu, by 2 p.m. on 20040407. Late homework will not be considered.

Homework #2



This program converts and displays points between Cartesian and Polar coordinate systems. Enter points and press convert; or click on the graph to choose a point.

X 1
Y 1
R 1.4142
T 45 (degrees)

Point Converter 1.0

1. (50 points) Make *pointsgui.m* fully functional by programming its callbacks to do all of the following:
    a. When the user enters a number in the X, Y, R, or T text boxes and presses return, the point should plot on the graph to the right, **and** the relevant conversion should happen and be displayed in the appropriate boxes. For instance, if I type a "1" in X and press return, an arrow should appear in the graph pointing straight right, and the R text box should show "1" while T shows “0”. T should be given in degrees. Use the MATLAB functions *pol2cart* and *cart2pol*. To make the arrow use the command *compass*.
    b. When the user enters two values for either Cartesian or Polar and presses the corresponding convert button, the point should plot in the graph to the right **and** the conversion shown in the corresponding text boxes. T should be given in degrees.
    c. The "Choose Point" button, when pressed, should allow the user to click on the graph to the right, draw an arrow from the origin to the point, **and** show the coordinates in the both the Cartesian and Polar text-boxes. (HINT: look at the function *ginput*.)
    d. When the "Polar Plot" button is pressed, the graph on the right should become a polar plot. (Use the function *compass* or *polar* to create this plot.) The text field of this button should change to "X-Y Plot." When that is pressed, the graph on the right should

be changed to a rectangular plot, and the text field of the button changed to "Polar Plot."

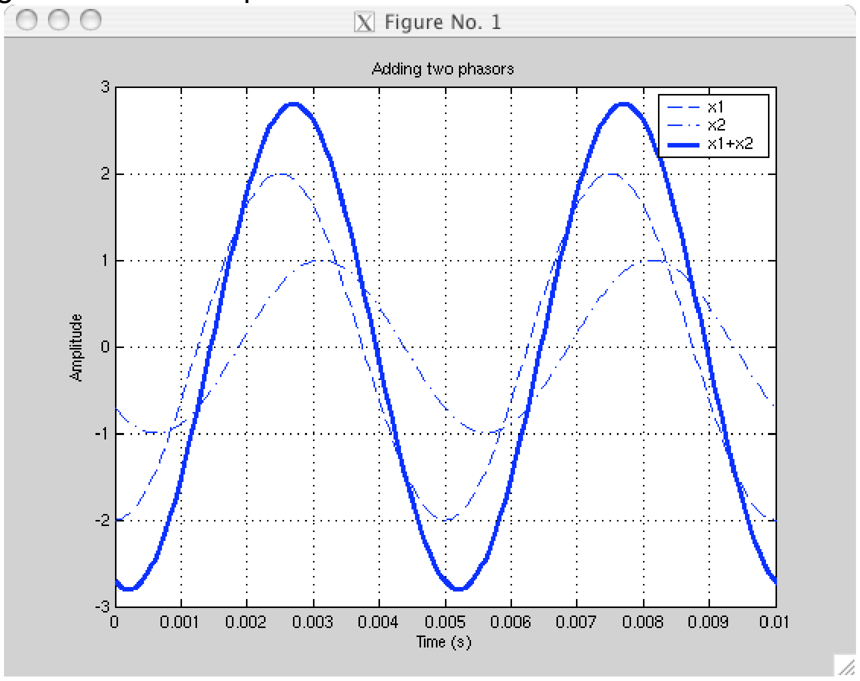
   e. When the "Grid" checkbox is selected, the graph on the right should contain a grid.

2. (25 points) Using the Sampling Explorer in SSUM answer the following:
   a. What frequency (Hz) sinewave is produced at the output when sampling a 900 Hz sinewave at Fs = 1000 Hz? (Remember frequency is the inverse of period.)
   b. What frequency sinewave is produced at the output when sampling a 700 Hz sinewave at Fs = 1000 Hz?
   c. Derive a formula for the frequency of the output given the input sinewave frequency f, and sampling rate Fs, where f > Fs/2.
   d. Why is it dangerous to sample a signal at exactly twice its highest frequency component?
   e. Why it is better to have more bits per sample?
3. (25 points) Write a short synopsis of the algorithm for the Additive Synthesis Forest. Start with *birdsgui.m*. How is the GUI created? Trace what happens when the user selects "Robin" and presses play. (Note: The function *synth.m* is located in the library directory of SSUM.)

When completed and corrected e-mail your file *pointsgui.m*, your answers to part 2, and the discussion of the Additive Synthesis Forest, to the T.A. b.sturm@mat.ucsb.edu, by 2 p.m. on 20040414. Late homework will not be accepted.

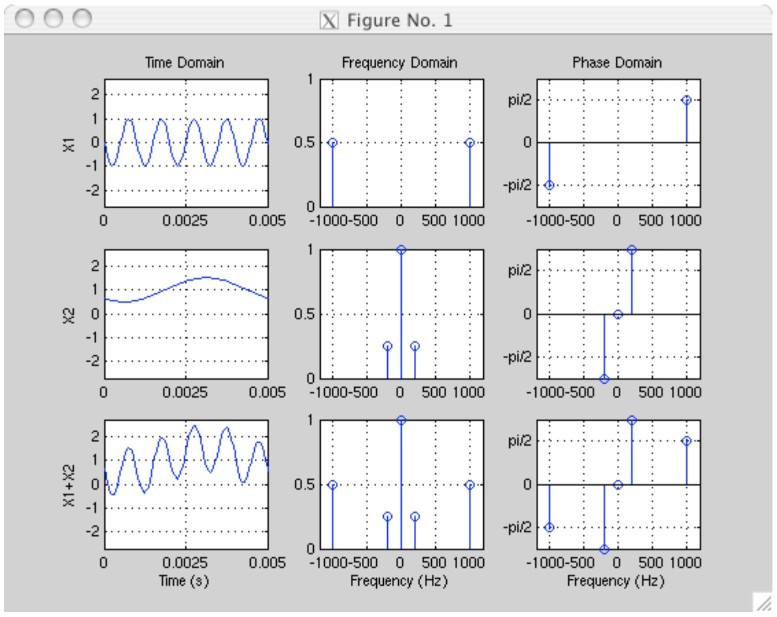
Homework #3

1. (30 points) Reprogram *pointsgui.m* so that the GUI code is separate from the functional code, *pointsfn.m*. The function *pointsfn.m* has already been started and one example of it is shown. Use any SSUM program as an example of how this is done.

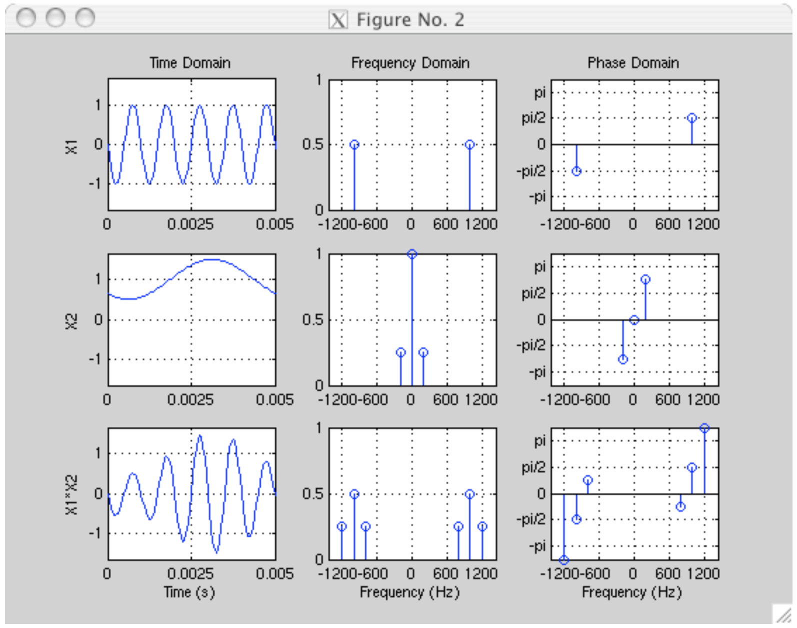


2. (20 points) Add the following phasors using MATLAB, and plot two periods of each waveform (x1, x2, x1+x2) on **one** plot. Use different linestyles for the plot and use *legend*. Set the linewidth of the sum to 3 (Hint: h = plot(…); get(h)). Check your result using the SSUM applications *Complex Number Explorer*, and *Sine Explorer*. Figure 1 shows how your plot should look.
    a. x1 = 2 cos(2*pi*200*t – pi), x2 = cos(2*pi*200*t + 3pi/4)

3. (50 points) Add and multiply the following phasors using MATLAB and plot each waveform (x1, x2, x1+x2; x1, x2, x1*x2) over the longest period of the three. In same figure plot the spectrum of each real waveform. Your plots should look like figures 2 and 3, but not exactly. (Note: use *stem*, not *fft* for the frequency and phase plots.) Use SSUM *Sine Explorer* to verify your results. *LOADS* of extra credits for those who make a function to do this automatically for any two sinusoids (HINT: use structures).
    a. x1= cos(2*pi*500*t – pi/2), x2=1+0.5 cos(2*pi*100*t + pi/4)



Adding two phasors, one with an offset

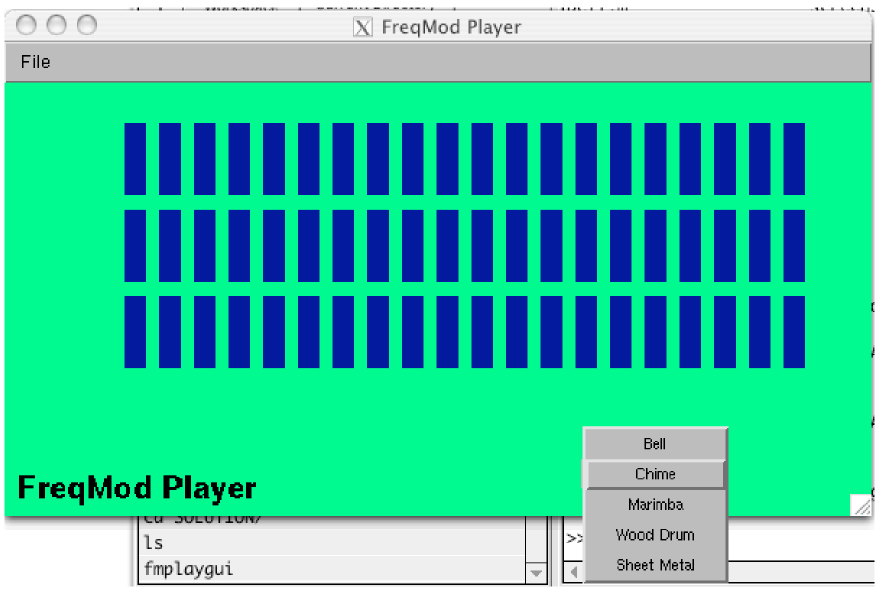


Multiplying two phasors, one with an offset

When completed and corrected e-mail your files *pointsgui.m*, *pointsfn.m*, and the scripts used to answer questions 2 and 3, to the T.A. b.sturm@mat.ucsb.edu, by 2 p.m. on 20040421. Late homework will not be accepted.

Homework #4

In this assignment you will create your own GUI using GUIDE that interfaces the frequency modulation synthesizer. The FreqMod Player is included as an example. Additional help can be found in the MATLAB manual.



```
ls
fmplaygui
```

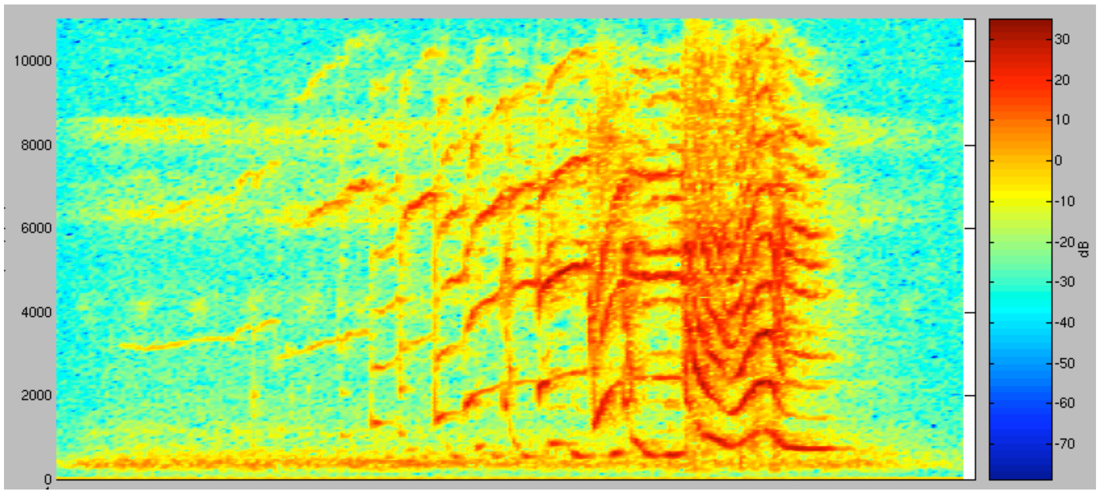
Example interface; the blue rectangles trigger the sounds when clicked.

1. (100 points) Using GUIDE, create a GUI that interfaces the frequency modulation synthesizer found in the SSUM library in the *synth.m* and *freqmod.m* functions.
    a. Make a menu to select an instrument to play: {bell, chime, wood drum, marimba, or sheet metal}. Use the special values in the provided *fmplayfn.m* for each instrument.
    b. Make at least five buttons to serve as the "keyboard." When you press a button the selected instrument sound is synthesized and played through the speakers. The pitch of the note should be different for each button. Feel free to use any scale you want. Extra credit if you make a menu to select particular scales.
    c. Create a volume slider that scales the amplitude of the synthesized sound.
    d. Extra credit if you find and code other instruments, or include an interface to modify frequency modulation parameters.

When completed and corrected e-mail all files necessary to run your GUI (including the .fig file) to the T.A. b.sturm@mat.ucsb.edu, by 2 p.m. on 20040503. Late homework will not be accepted.

Homework #5, Part I

Choose either Part I or Part II (extra credit if you do both). In Part I you will perform an analysis of bird song, synthesize the sound using additive synthesis, and compare the spectra of the original with the synthesis.



Sonogram of Costa Rican bird Montezuma Oropendolas.

1. (50 points) From the birds.zip collection of bird song, you will synthesize the bird of your choice.
    a. Use the SSUM Sonogram Explorer to trace the spectral components (i.e. use the “Explore” button).
    b. Using the data you collected above, resynthesize the sound with the additive synthesis code in SSUM. (Hint: look at the SSUM Additive Synthesis Forest to see how to do this.)
    c. Write this sound to a sound file.
2. (50 points) Compare the original sound to the synthesized sound.
    a. Create a figure with two plots, one showing a sonogram of the original sound, and the other showing a sonogram of the synthesized version. Make sure the axis limits on the two plots are the same. (Extra credit if you make a GUI to load two sound files which will then display the sonogram automatically.)
    b. Put play buttons next to each plot to play the corresponding sound. You don’t need to use GUIDE to do this; see uicontrol. (Hint: Look at the SSUM Catastochastic Additive Synthesis code if you are unsure of how to do this.)

When completed and corrected e-mail all m-files necessary for parts 1 and 2 to the T.A. b.sturm@mat.ucsb.edu, by 2 p.m. on 20040517. Late homework will not be accepted.

Homework #5, Part II

In this assignment you will look at how windowing affects the Fourier transform of two signals.

1. Create a 0.5 second1000 Hz sinewave with a Fs of 11127 and save this to a file.
2. For this sound and S5pcm.wav (in the SSUM data directory) do the following:
    a. (40 points) For a fixed window size of 512 samples, look at the effects of the rectangular, Hann, and Hamming windows on the Fourier transform. Discuss the differences between each for each signal. (Take 512 samples of the speech file starting at sample 6231.)
    b. (40 points) Vary the window length for each window for each sound and discuss the effects of the length of each window on the Fourier transform.
    c. (19 points) For each signal, which is the best window and window length and why?

When completed and corrected e-mail answers to these questions and your programs created to answer them to the T.A. b.sturm@mat.ucsb.edu, by 2 p.m. on 20040512. Late homework will not be accepted.

Final Project Details

The final project will consist of two parts: a program written using MATLAB, and a paper discussing the program and the principles behind it. Ideally the project should be something that is creative and interesting to you. It must make use of media signal processing concepts, such as filtering and spectral analysis. If your program is good, it will be integrated with SSUM. The program should consist of more than 2000 lines of code and should be presented within a GUI.

The paper accompanying your project should document the idea, execution, and refinement of the program, and should address the principles behind it. If your project consists of creating a multiband vocoder then the paper should present that subject with several references. The paper should be no less than 1000 words, and contain at least three references. The projects will be presented in class during the final week (**June 7, 9, 2004**).

The “Catastochastic Additive Synthesis Machine” included in SSUM, for instance, is a good example for a project. It involves extensive function writing, GUI work, and creates interesting output; it also demonstrates an understanding of windowing and combining signals. Furthermore it has use outside of its demonstrative purpose—one can compose with it. You are encouraged to pursue projects that are of interest to your artistic sensibilities, but they must relate to the class.

Whether you are a visual or sound artist you will find MATLAB presents a wealth of interesting possibilities. The great thing about it is its use in prototyping algorithms and ideas. Because of this Sturm has used MATLAB to compose music since 1998. Once interesting opportunities are discovered the algorithms can be translated to faster languages like C++.

You are required to write a project proposal due **May 5, 2004**. The proposal should address the following points:

1. Statement of purpose, problem; importance of topic
2. Work done by others on this topic
3. Your approach and how it may be unique
4. Your interest in the topic and its relevance to class

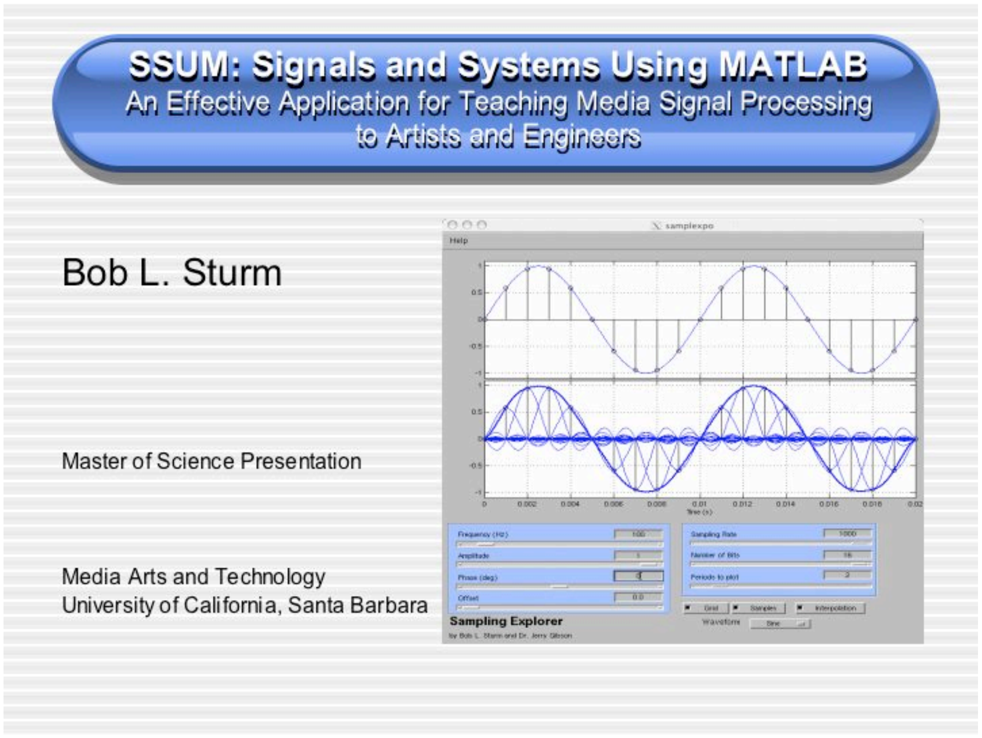
The proposal should be no more than 500 words, and must include at least two references. After reviewing your proposal the professor will meet with you to discuss it.

WARNING*: There are many interesting projects that have been created in MATLAB and are completely available on-line. If your program is found to be similar to any of these you will lose 25% from your final grade.*

Ideas for final projects

1. Algorithmic Composition
    a. Cellular automata, genetic algorithms, neural networks, markov chains, expert systems
2. Sound synthesis/modification techniques
    a. Wavetable/wave-terrain/wave-shaping
    b. Subtractive synthesis, Frequency/Phase modulation, Granular synthesis
    c. Physical modeling using digital waveguides
    d. Speech synthesis/LPC
    e. Spectral morphing, Phase Vocoder
    f. Create a “Metasynth” for MATLAB
3. Spatialization
    a. HRIRs and HRTFs, Ambisonics, Vector Based Amplitude Panning
4. Acoustics
    a. Modeling room acoustics, Reverberation
5. Sonification
    a. Seismic, ocean, astronomical, stock market, fractals
6. Graphics manipulation
    a. Warping techniques
7. Video analysis
    a. Motion detection, Scene analysis/recognition
8. Content retrieval
    a. timbre recognition, automatic classification, score following
    b. Speech recognition, Face/object recognition
    c. Optical Character recognition, Visual scene changes
9. Sound analysis
    a. Phase Vocoder, Pitch detection
    b. Foot-tappers, tempo/rhythm derivation;
    c. Transient/onset detection, Classifying sounds/timbres
    d. Speech Analysis
10. Sound Modeling
    a. Spectral Modeling Synthesis, Phase Vocoder
    b. Physical modeling using digital waveguides
    c. Linear Prediction Coding
    d. Neural Nets
11. Encryption
    a. Hiding data/text within an image or sound
12. Signal compression
    a. Sound: MPEG-2 layer 3, mu-law
    b. Images: TIFF, GIF, JPEG
    c. Movies: MPEG-2, AVI

Appendix C: MS Presentation

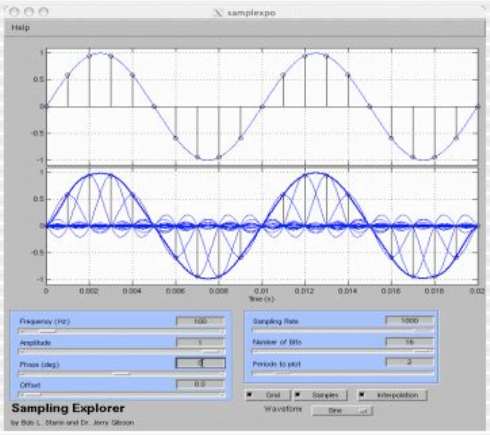


SSUM: Signals and Systems Using MATLAB
An Effective Application for Teaching Media Signal Processing to Artists and Engineers

Bob L. Sturm

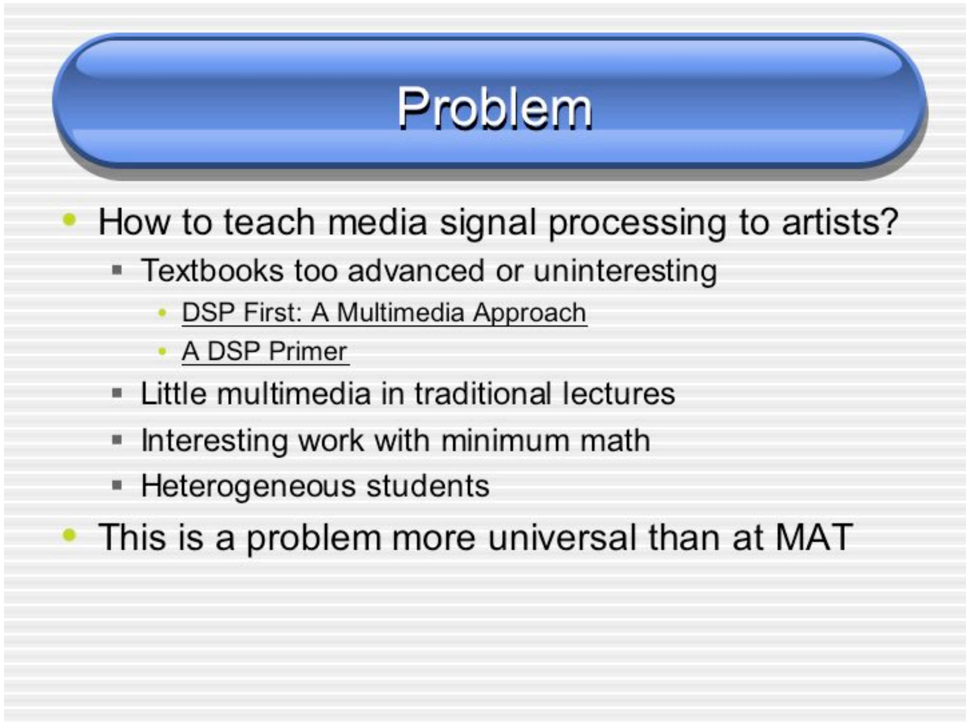
Master of Science Presentation

Media Arts and Technology
University of California, Santa Barbara



Frequency (Hz) 100
Amplitude 1
Phase (deg) 0
Offset 0.0
Sampling Rate 1000
Number of Bits 16
Periods to plot 2
Grid
Samples
Interpolation
Waveform Sine

Sampling Explorer
by Bob L. Sturm and Dr. Jerry Gibson



Problem

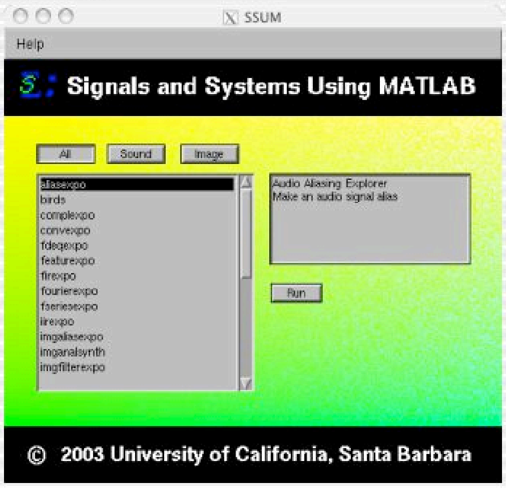
- How to teach media signal processing to artists?
  - Textbooks too advanced or uninteresting
    - DSP First: A Multimedia Approach
    - A DSP Primer
  - Little multimedia in traditional lectures
  - Interesting work with minimum math
  - Heterogeneous students
- This is a problem more universal than at MAT

A Solution

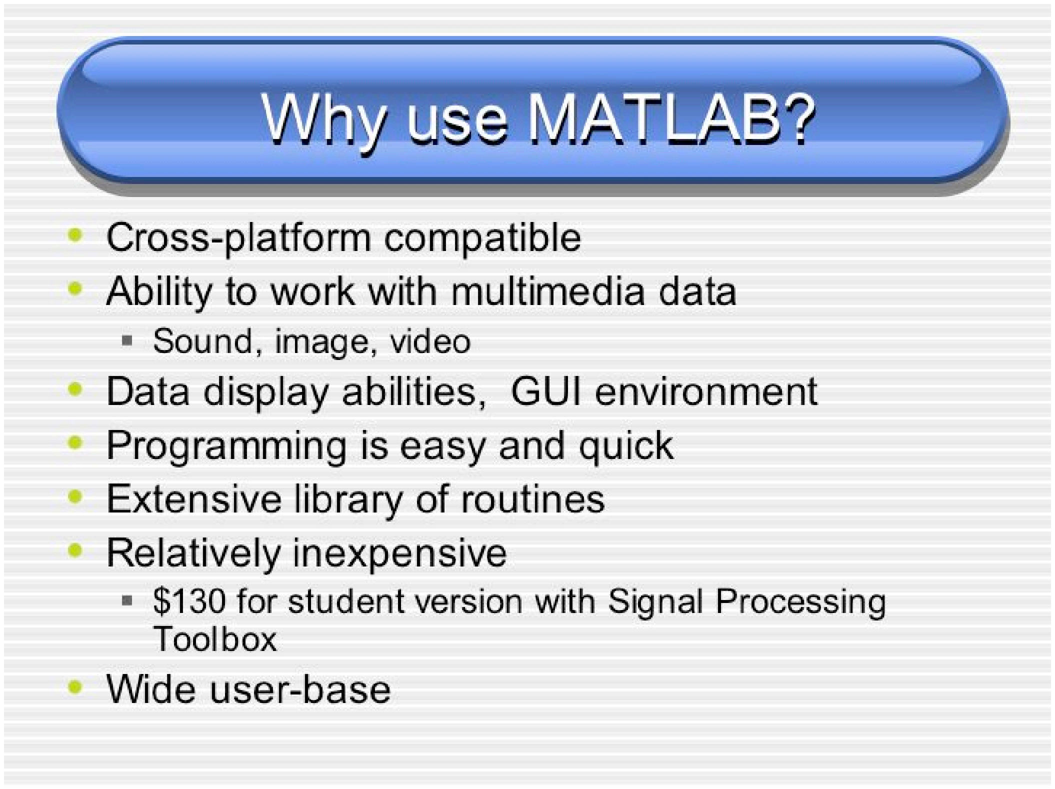
- Lecture with demonstrations
- Discuss real signals like sound and image
- Inspire and motivate using examples from art and music
- Focus on programming algorithms rather than math
- Reader with DSP articles and notes of interest to artists

SSUM

- Suite of over 25 exploratory applications programmed in MATLAB
  - Multimedia
  - Creative
  - All wrapped in GUIs
  - Perfect for lectures, student's own time

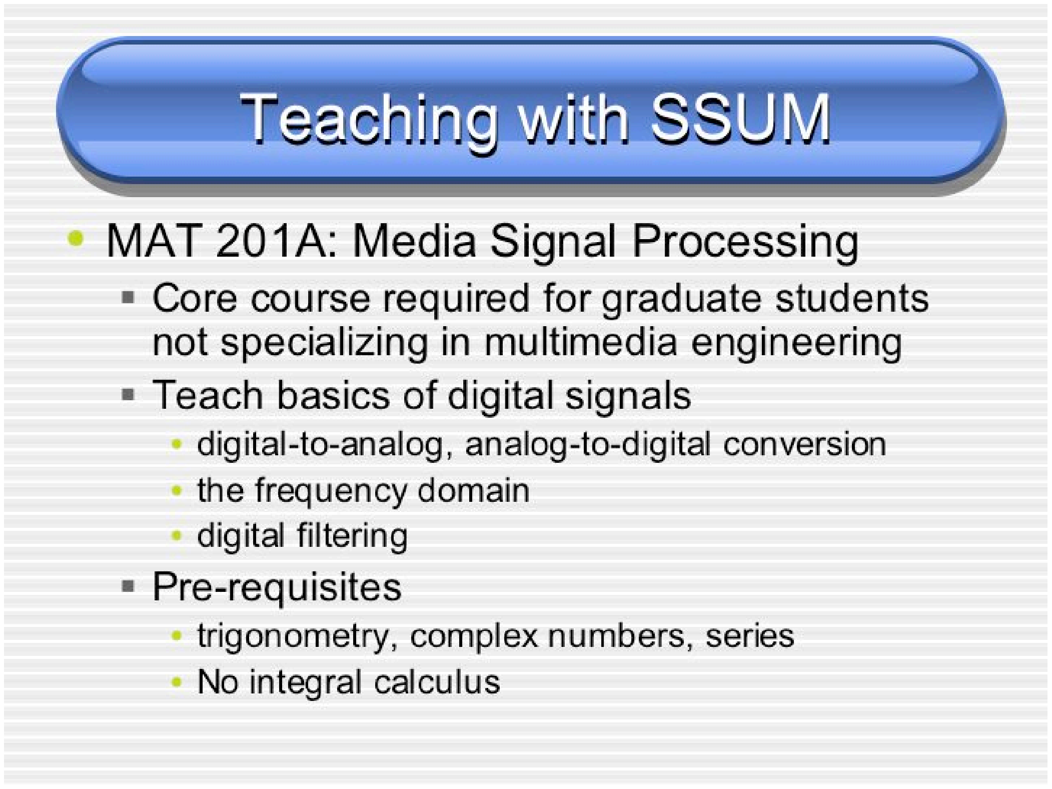


Audio Aliasing Explorer
Make an audio signal alias



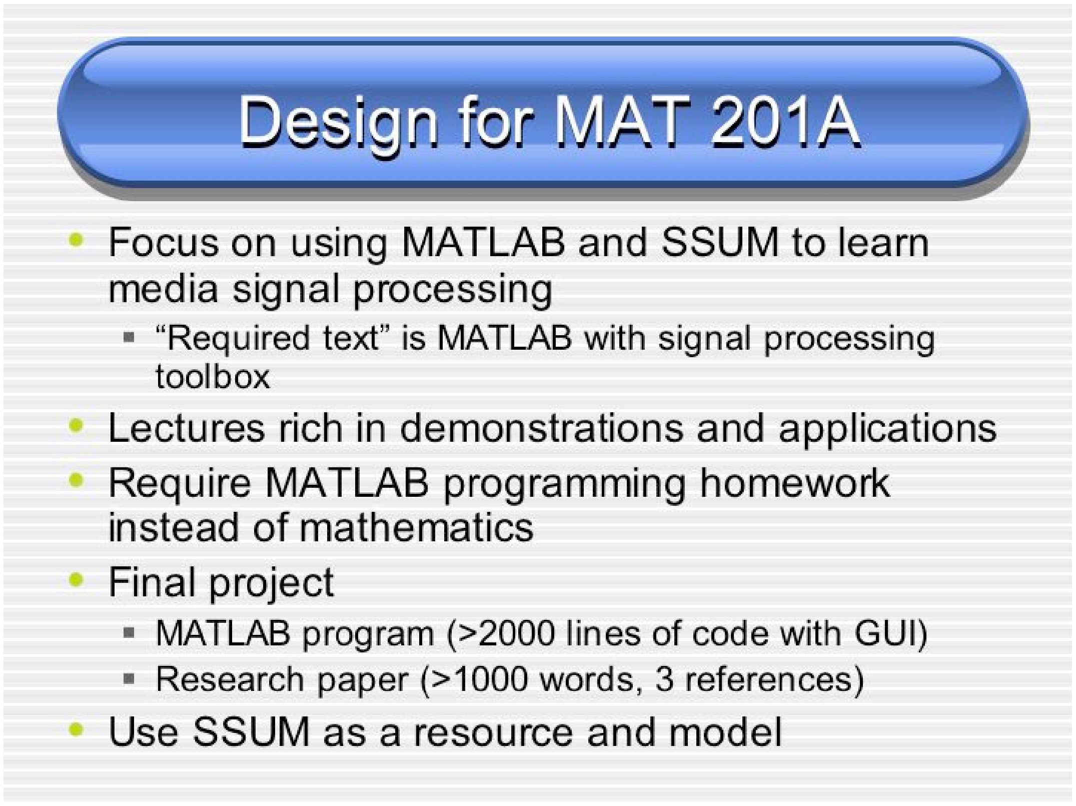
Why use MATLAB?

- Cross-platform compatible
- Ability to work with multimedia data
  - Sound, image, video
- Data display abilities, GUI environment
- Programming is easy and quick
- Extensive library of routines
- Relatively inexpensive
  - $130 for student version with Signal Processing Toolbox
- Wide user-base



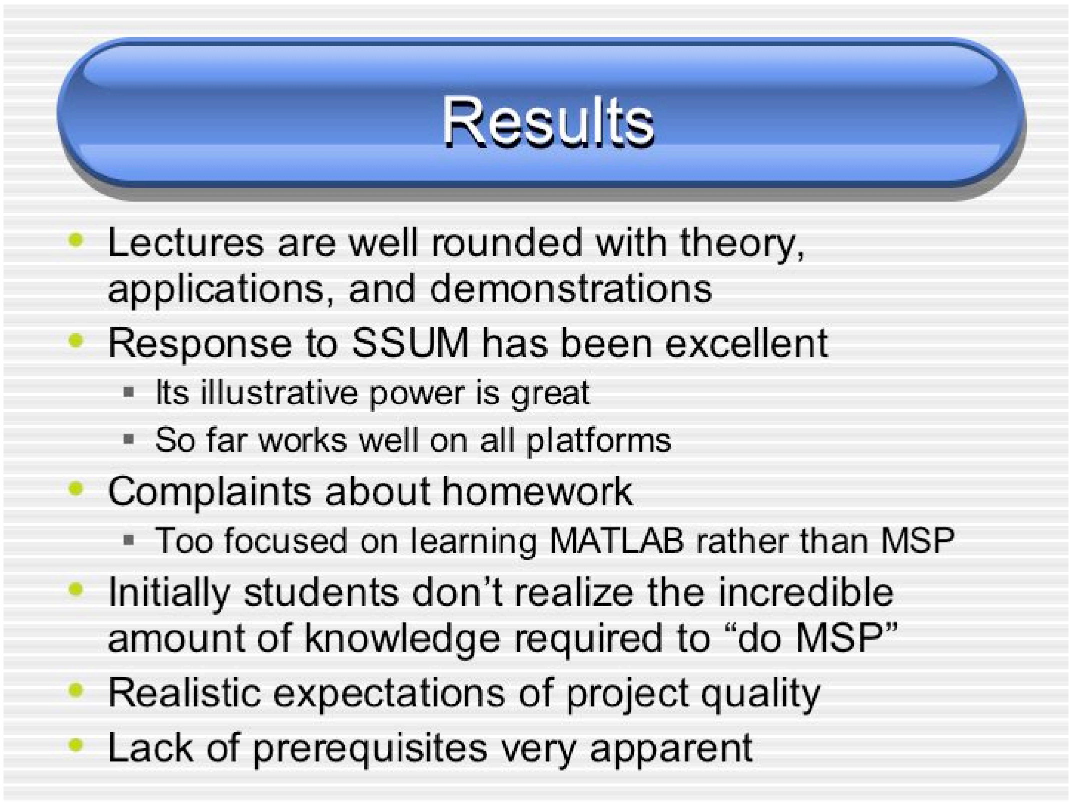
Teaching with SSUM

- MAT 201A: Media Signal Processing
  - Core course required for graduate students not specializing in multimedia engineering
  - Teach basics of digital signals
    - digital-to-analog, analog-to-digital conversion
    - the frequency domain
    - digital filtering
  - Pre-requisites
    - trigonometry, complex numbers, series
    - No integral calculus



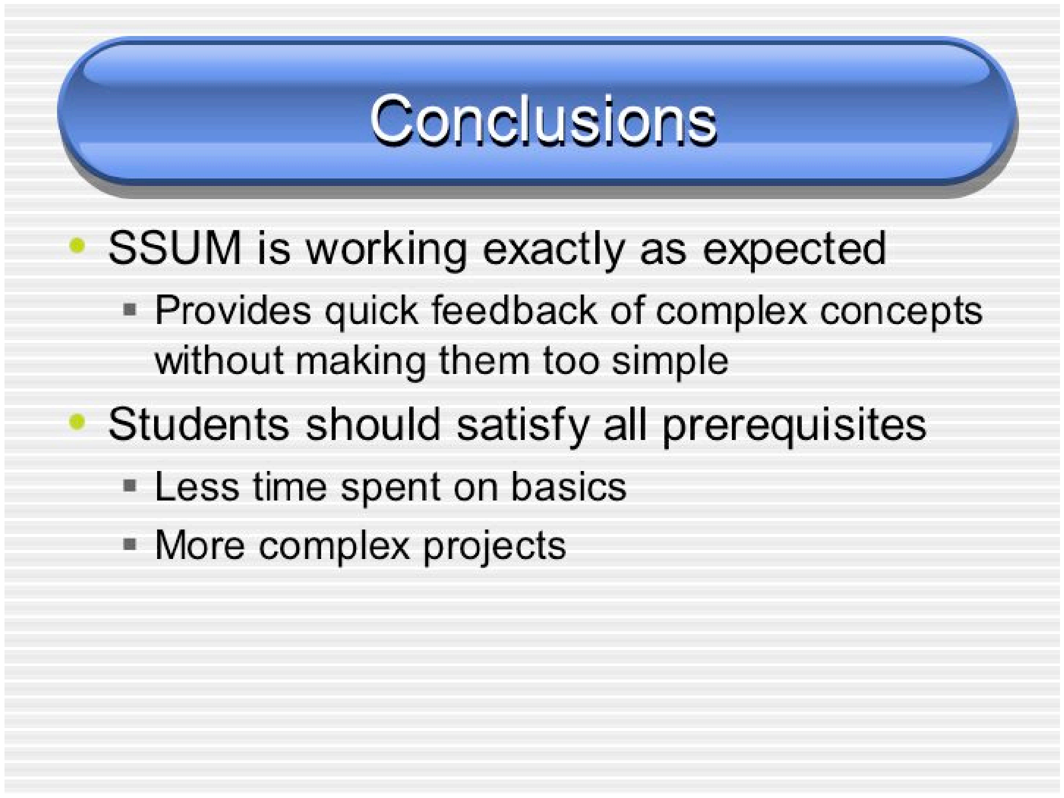
Design for MAT 201A

- Focus on using MATLAB and SSUM to learn media signal processing
  - “Required text” is MATLAB with signal processing toolbox
- Lectures rich in demonstrations and applications
- Require MATLAB programming homework instead of mathematics
- Final project
  - MATLAB program (>2000 lines of code with GUI)
  - Research paper (>1000 words, 3 references)
- Use SSUM as a resource and model



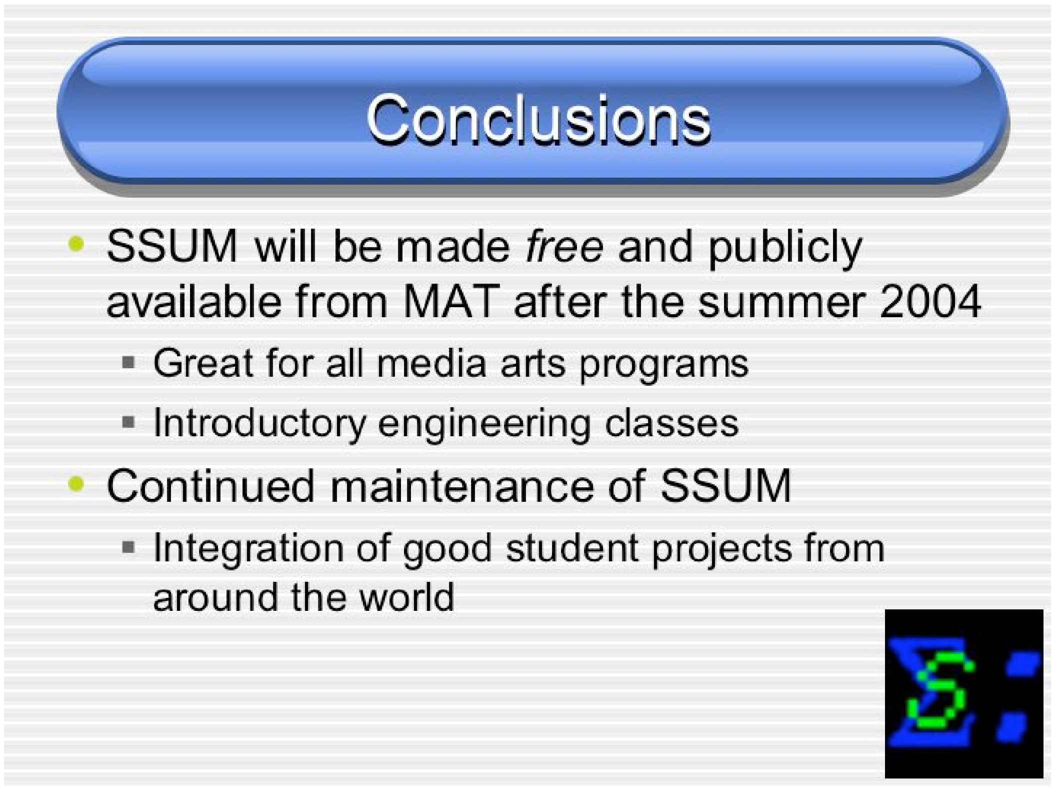
Results

- Lectures are well rounded with theory, applications, and demonstrations
- Response to SSUM has been excellent
  - Its illustrative power is great
  - So far works well on all platforms
- Complaints about homework
  - Too focused on learning MATLAB rather than MSP
- Initially students don’t realize the incredible amount of knowledge required to “do MSP”
- Realistic expectations of project quality
- Lack of prerequisites very apparent



Conclusions

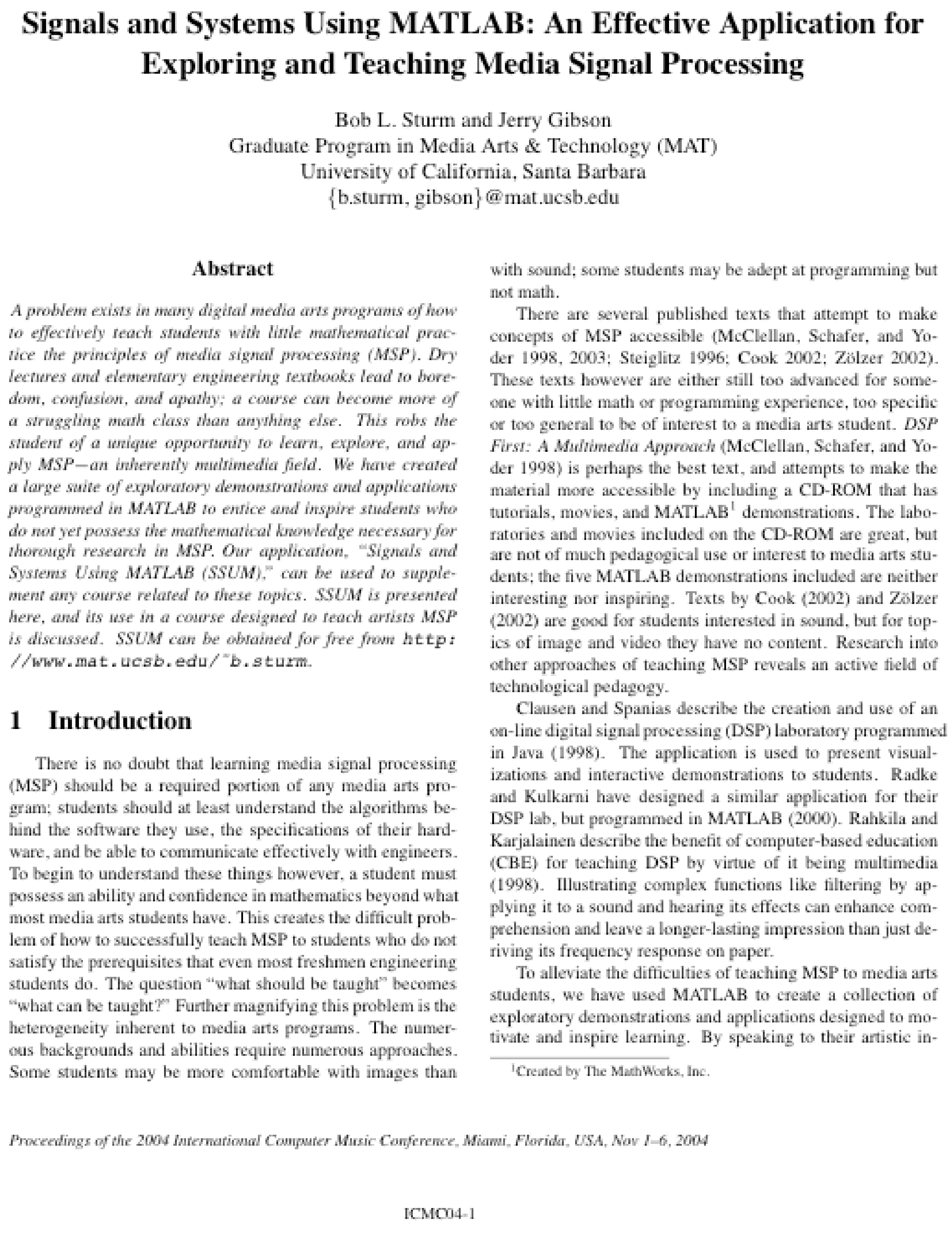
- SSUM is working exactly as expected
  - Provides quick feedback of complex concepts without making them too simple
- Students should satisfy all prerequisites
  - Less time spent on basics
  - More complex projects



Conclusions

- SSUM will be made *free* and publicly available from MAT after the summer 2004
  - Great for all media arts programs
  - Introductory engineering classes
- Continued maintenance of SSUM
  - Integration of good student projects from around the world

Appendix D: Publication of SSUM at ICMC2004



Signals and Systems Using MATLAB: An Effective Application for Exploring and Teaching Media Signal Processing

Bob L. Sturm and Jerry Gibson
Graduate Program in Media Arts & Technology (MAT)
University of California, Santa Barbara
{b.sturm, gibson}@mat.ucsb.edu

Abstract

A problem exists in many digital media arts programs of how to effectively teach students with little mathematical practice the principles of media signal processing (MSP). Dry lectures and elementary engineering textbooks lead to boredom, confusion, and apathy; a course can become more of a struggling math class than anything else. This robs the student of a unique opportunity to learn, explore, and apply MSP—an inherently multimedia field. We have created a large suite of exploratory demonstrations and applications programmed in MATLAB to entice and inspire students who do not yet possess the mathematical knowledge necessary for thorough research in MSP. Our application, "Signals and Systems Using MATLAB (SSUM)," can be used to supplement any course related to these topics. SSUM is presented here, and its use in a course designed to teach artists MSP is discussed. SSUM can be obtained for free from `http://www.mat.ucsb.edu/~b.sturm`.

1 Introduction

There is no doubt that learning media signal processing (MSP) should be a required portion of any media arts program; students should at least understand the algorithms behind the software they use, the specifications of their hardware, and be able to communicate effectively with engineers. To begin to understand these things however, a student must possess an ability and confidence in mathematics beyond what most media arts students have. This creates the difficult problem of how to successfully teach MSP to students who do not satisfy the prerequisites that even most freshmen engineering students do. The question "what should be taught" becomes "what can be taught?" Further magnifying this problem is the heterogeneity inherent to media arts programs. The numerous backgrounds and abilities require numerous approaches. Some students may be more comfortable with images than with sound; some students may be adept at programming but not math.

There are several published texts that attempt to make concepts of MSP accessible (McClellan, Schafer, and Yoder 1998, 2003; Steiglitz 1996; Cook 2002; Zölzer 2002). These texts however are either still too advanced for someone with little math or programming experience, too specific or too general to be of interest to a media arts student. *DSP First: A Multimedia Approach* (McClellan, Schafer, and Yoder 1998) is perhaps the best text, and attempts to make the material more accessible by including a CD-ROM that has tutorials, movies, and MATLAB[1] demonstrations. The laboratories and movies included on the CD-ROM are great, but are not of much pedagogical use or interest to media arts students; the five MATLAB demonstrations included are neither interesting nor inspiring. Texts by Cook (2002) and Zölzer (2002) are good for students interested in sound, but for topics of image and video they have no content. Research into other approaches of teaching MSP reveals an active field of technological pedagogy.

Clausen and Spanias describe the creation and use of an on-line digital signal processing (DSP) laboratory programmed in Java (1998). The application is used to present visualizations and interactive demonstrations to students. Radke and Kulkarni have designed a similar application for their DSP lab, but programmed in MATLAB (2000). Rahkila and Karjalainen describe the benefit of computer-based education (CBE) for teaching DSP by virtue of it being multimedia (1998). Illustrating complex functions like filtering by applying it to a sound and hearing its effects can enhance comprehension and leave a longer-lasting impression than just deriving its frequency response on paper.

To alleviate the difficulties of teaching MSP to media arts students, we have used MATLAB to create a collection of exploratory demonstrations and applications designed to motivate and inspire learning. By speaking to their artistic in-

[1] Created by The MathWorks, Inc.

terests and showing the creative usefulness of the concepts the students are more likely to accept the learning curve and apply themselves. Within this paper we present this suite of applications and review its use in a class teaching MSP.

1.1 Why Use MATLAB?

There are several goals for the development of a suite of effective demonstrations and applications for teaching MSP. First, the concepts must be presented clearly with little interfering information. Second the applications should be direct, flexible, and fast. Third, both sound and image must be used to demonstrate concepts. Fourth, the demonstrations should leave plenty to explore. Fifth, a student should be able to look into the code to understand how it works. Sixth, the demonstrations should be compatible with as many computer platforms possible. And finally, the cost to students to be able to run the applications on their own computer should be minimal.

There are a several low-level programming languages that can be used to demonstrate the application of DSP, such as C++, and Java. But these require a high proficiency in programming, not to mention the tangle of cross-platform issues. Sound processing languages SuperCollider(McCartney 1996) or the graphical programming applications Max/MSP[2] or pd (Puckette 1996), can also be used to create interesting demonstrations, but only for sound. Though these have excellent real-time capability, they have marginal abilities for visual data display.

There are several high-level software packages that can be used to teach signal processing, such as Mathematica,[3] Octave,[4] and MATLAB. A good overview of these and other packages in terms of engineering education can be found in (Nagrial 2002). Mathematica is meant more for symbolic mathematics than creating applications, and it cannot easily produce sound. Octave, a free open-source mathematics software application, is quite compatible with MATLAB code, but it lacks much of the rich library of functions in MATLAB. In addition there is no easy way to create graphical user interfaces (GUIs).

MATLAB provides a flexible integrated programming environment that is easy to use and understand, and cheap for students.[5] MATLAB is platform independent, has superior graphics handling and visualization capabilities, and has a great GUI development environment for wrapping applications. In addition it offers unparalleled functionality with many different data formats. It has an extensive library of routines, and "toolboxes" can be purchased to add more specialized functionality, such as advanced signal and image processing routines. Applications written in MATLAB are open; any user can look at the code. Furthermore, countless institutions, both academic and corporate, as well as many independent users worldwide, use MATLAB for algorithm development, quick prototyping, and complex problem solving. A drawback to using MATLAB, however, is its lack of real-time functions, like tracking a sound as it plays, or visualizing a spectrogram straight from the sound input. Though there should be some familiarity with vectors and matrices, the MATLAB programming language is easy to learn and intuitive. For these reasons it is clear that MATLAB is the best choice for developing applications that satisfy the several criteria above.

There are excellent examples of multimedia pedagogical applications written using MATLAB. The "MATLAB Auditory Demonstrations" (MAD) is perhaps the best and most relevant to signal processing (Cooke, Parker, Brown, and Wrigley 1999a). MAD provides a large suite of interactive demonstrations for exploring psychoacoustics (Cooke, Parker, Brown, and Wrigley 1999b). Using MAD as a model, we developed "Signals and Systems Using MATLAB" (SSUM) to aid in the teaching of MSP to media arts students at the Media Arts and Technology (MAT) graduate program at the University of California, Santa Barbara (UCSB).

2 SSUM: Signals and Systems Using MATLAB

SSUM is a suite of exploratory demonstrations and applications programmed in MATLAB. To use SSUM, MATLAB must be installed,[6] as well as the signal processing toolkit. SSUM demonstrates the essential principles and concepts of MSP without requiring rigorous mathematics; exploration and learning is done first using software. SSUM currently has 31 programs illustrating concepts of waveforms, modulation, sampling and interpolation, aliasing, the frequency domain, convolution and filtering, pole-zero diagrams, analysis and synthesis, statistical signal features, and many others. Many of these are applied to sound and images.

There are also applications that demonstrate curious topics such as sound cross-synthesis, additive synthesis of birdsong, and sine wave speech synthesis. SSUM is perfect for use in lectures, labs, homework, and creative work. All SSUM programs are wrapped in GUIs, so there is no need for typing commands at the prompt. Many of the applications are integrated as well. For instance, if one is creating a waveform in an application, it can be sent to another application

[2]Distributed by Cycling74: `http://www.cycling74.com`
[3]Created by Wolfram Research Inc.
[4]Available at `http://www.octave.org`
[5]Currently, the student version of MATLAB costs $99 US.

[6]Preferably MATLAB version 6.5 or greater.

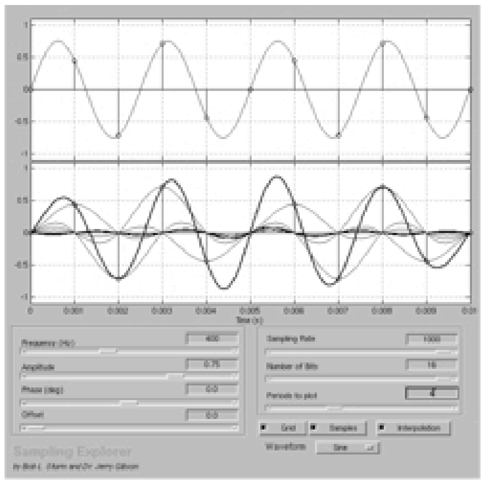


Figure 1: Sampling Explorer

for filtering, and to another to see its frequency content.

Figure 1 shows Sampling Explorer, which demonstrates how continuous signals are digitized. Using Sampling Explorer one can investigate the cause and effect of aliasing, the effects of quantization, and the process of making digital signals continuous using ideal lowpass filtering.

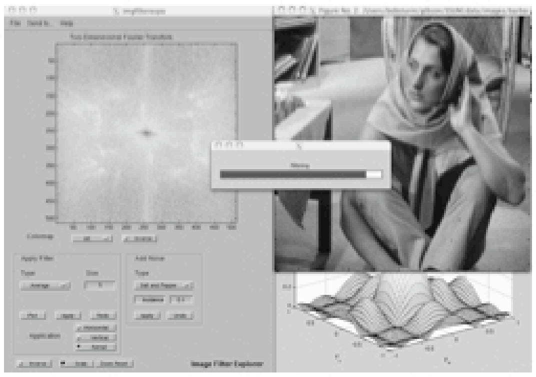


Figure 2: Image Filter Explorer

Filtering images can be explored using Image Filter Explorer (Figure 2). Once an image has been loaded, its two-dimensional Fourier transform is displayed. Several filters are available including the moving average, Gaussian, and median filter. The spatial frequency response of each filter can be plotted, except for the non-linear filters. The filters can be applied to only the horizontal or vertical directions, or over square blocks. Noise can be added to an image and its effects on filtering seen. Students find the ability of the median filter to remove speckle noise startling.

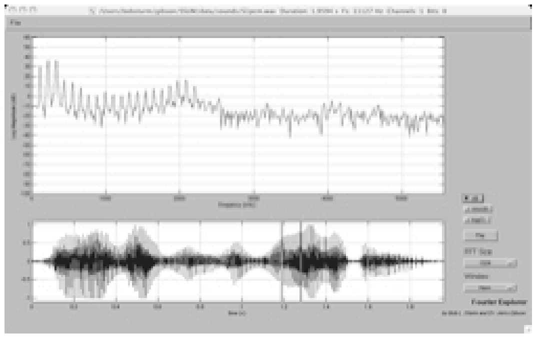


Figure 3: Fourier Spectrum Explorer

Fourier Spectrum Explorer, shown in Figure 3, enables one to look at the spectrum of a sound. As the user drags a window across the time-domain representation of a signal, the spectrum changes. The window size and shape can be changed. It would be ideal to have the window sweep as the sound was played, but currently MATLAB cannot handle such tasks. A similar application is Sonogram Explorer, which presents the user with the short-time Fourier transform (STFT) of a sound.

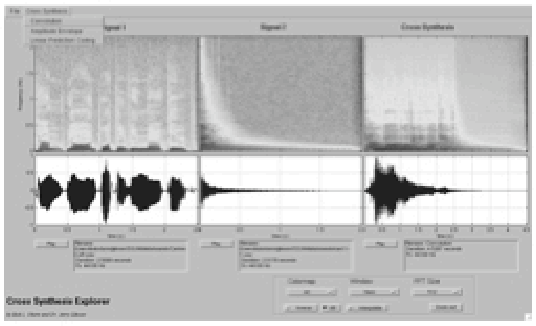


Figure 4: Cross-Synthesis Explorer

Figure 4 shows Cross-Synthesis Explorer. This application allows three different methods for cross-synthesizing sounds: convolution of the sounds, amplitude enveloping of one signal by the other, and linear prediction—using one sound as a model and the other as a source. Students really enjoy this demonstration and begin to realize what convolution does; suddenly the mystery of digital reverberation disappears.

In addition to these applications, SSUM contains demonstrations of MATLAB programming for making sound, music, and images. Using these applications as models, students can easily and quickly construct their own composition programs.

3 Conclusion

SSUM has been integrated into a course teaching MSP to media arts students in the MAT graduate program. The syllabus is not intended to be exhaustive, but the students should finish with at least an understanding of digital signals (e.g. samples, and sampling), the frequency domain (e.g. spectra), conversion between analogue and digital systems (e.g. interpolation), filtering (FIR, IIR, z-transform), and time-frequency analysis (e.g. DFT, DTFT). Weekly assignments involve MATLAB programming and using SSUM applications. For a final project each student is required to write a short research paper and MATLAB program exploring an aspect of MSP.

It might be stated that by focusing on MATLAB in a syllabus one is replacing the difficulty of learning mathematics with programming. Thus the class will become more a class of programming MATLAB than learning MSP. However, due to the multimedia nature of MSP, it makes more sense to concentrate on practicing building applications to learn the theory rather than plugging away with abstract mathematics.

The use of SSUM in our classroom has proved to be indispensable for quickly and effectively illustrating concepts. SSUM is designed first for practical demonstrations, second to provide an interactive experience to enhance comprehension of MSP, and third to serve as a repository of algorithms and code. SSUM nicely satisfies these three goals, and creates a fruitful multimedia experience for teaching and learning MSP. All the applications are quick to compute and display results, so there is little worry for the learning process to come to a halt. As SSUM is used more in the classroom, its collection of demonstrations and applications will grow by incorporating good projects by students.

By catering to the creative motivations of the media arts student, the difficult concepts of MSP can be approached with enthusiasm rather than dread. SSUM provides a practical experience with great examples. "If the teachers can create an enduring fascination for the subject-matter, the job's almost over: the more the students love the subject, the less help they need in their studies" (Koumi 1994).

Much more information on SSUM can be found in (Sturm 2004). SSUM can be downloaded for free from `http://www.mat.ucsb.edu/~b.sturm`.

3.1 Acknowledgments

The MathWorks, Inc., the makers of MATLAB, has supported this research by providing the authors with full multiplatform licenses to MATLAB.

References

Clausen, A. and A. Spanias (1998). An internet-based computer laboratory for DSP courses. In *Proc. of 28th ASEE/IEEE Frontiers in Education*.

Cook, P. R. (2002). *Real Sound Synthesis for Interactive Applications*. Massachusetts: A. K. Peters.

Cooke, M., H. Parker, G. J. Brown, and S. N. Wrigley (1999a). The interactive auditory demonstrations project. In *Eurospeech Conference*.

Cooke, M., H. Parker, G. J. Brown, and S. N. Wrigley (1999b). Mad: MATLAB auditory demonstrations. `http://www.dcs.shef.ac.uk/~martin/MAD/docs/mad.htm`.

Koumi, J. (1994). Designing for learning—effectiveness with efficiency. In R. Hoey (Ed.), *Effective Screenwriting for Educational Television*, pp. 230–239. U.K.: Kogan Page Ltd.

McCartney, J. (1996). Supercollider: A new real-time sound synthesis language. In *Proc. of the Int. Computer Music Conference*.

McClellan, J. H., R. Schafer, and M. A. Yoder (1998). *DSP First: A Multimedia Approach*. New Jersey: Prentice Hall.

McClellan, J. H., R. Schafer, and M. A. Yoder (2003). *Signal Processing First*. New Jersey: Prentice Hall.

Nagrial, M. (2002). Education and training in engineering software and applications. In *Int. Conference on Engineering Education*.

Puckette, M. (1996). Pure data. In *Proc. of the Int. Computer Music Conference*.

Radke, R. J. and S. Kulkarni (2000). An integrated MATLAB suite for introductory DSP education. In *Proc. of the First Signal Processing Education Workshop*.

Rahkila, M. and M. Karjalainen (1998). Considerations of computer based education in acoustics and signal processing. In *Proc. of 28th ASEE/IEEE Frontiers in Education*.

Steiglitz, K. (1996). *A DSP Primer: with Applications to Digital Audio and Computer Music*. Menlo Park: Addison Wesley.

Sturm, B. L. (2004). *SSUM: Signal and Systems Using MATLAB; Creating an Effective Application for Teaching Media Signal Processing to Artists and Engineers*. (M.S. Project) University of California, Santa Barbara, Graduate Program in Media Arts and Technology, USA.

Zölzer, U. (Ed.) (2002). *DAFx: Digital Audio Effects*. New York: Wiley.

4 Appendix

This is a list of exploratory demonstrations and applications currently implemented in SSUM.

- *Additive Synthesis Bird Song*
  Bird song synthesized using additive synthesis.
- *Catastochastic Additive Synthesis Composition Machine*
  Random music generator with variable partials and envelopes.
- *Complex Number Explorer*
  Visualize complex numbers; add/subtract vectors.
- *Concatenative Synthesis Explorer*
  Synthesize sounds from other sounds using feature extraction and matching criteria.
- *Convolution Explorer*
  Visualize linear and circular convolution with different signals.
- *Cross-Synthesis Explorer*
  Cross-synthesize two sounds using convolution, amplitude enveloping, or LPC.
- *Finite Difference Equation Explorer*
  Enter finite difference equations and see their frequency and impulse response.
- *FIR Filter Explorer*
  Create FIR filters and apply to a sound.
- *Formant Explorer*
  Drag window across sound and watch the spectrum and formants change; also displays autocorrelation, cepstrum.
- *Fourier Explorer*
  Drag window across sound and watch the spectrum change.
- *Fourier Series Explorer*
  Inspect the Fourier series of a periodic step function.
- *IIR Filter Explorer*
  Create IIR filters and apply to sound.
- *Image Aliasing Explorer*
  Explore aliasing for images; use anti-aliasing filter for downsampling.
- *Image Analysis/Reconstruction*
  Spectral analysis of image and reconstruction. Ability to swap magnitudes and phases of other images.
- *Image Filter Explorer*
  Apply different filters to images; see linear filter frequency response.
- *Image Sampling Explorer*
  Sample images at different resolutions.
- *Image Spectrum Explorer*
  Explore the spatial frequencies in images.
- *LPC Explorer*
  Explore linear prediction for audio. Resynthesize with residual, noise, pulses, or another sound.
- *Model Explorer*
  See the effects of different models for simulating communication.
- *Modulation Explorer*
  Modulate one signal by another and see changes in spectrum and waveform.
- *Pole-Zero Explorer*
  Drag poles and zeros around a unit circle to watch frequency and impulse response change.
- *Pole-Zero Filter Explorer*
  Create filter using pole-zero plot and apply it to sound.
- *Sampling Explorer*
  Demonstrate sampling, quantization, and interpolation.
- *Signal Feature Explorer*
  Explore the statistics of a signal, such as RMS, spectral centroid, and pitch.
- *Sinewave Speech Synthesis Explorer*
  Use LPC to reduce sounds to four sine waves.
- *Sinusoidal Explorer*
  Parameters of sine waves; add and multiply two sine waves.
- *Sonogram Explorer*
  Explore the STFT of a signal; trace partials with mouse clicks.
- *Sound Aliasing Explorer*
  Explore aliasing and folding for sound signals; use anti-aliasing filter for downsampling.
- *Sound Analysis/Synthesis*
  Spectral analysis of sound and resynthesis. Ability to swap magnitudes and phases of other sounds.
- *Waveform Explorer*
  Generate waveforms using 15 sinusoids.
- *Window Explorer*
  Explore the frequency and phase response of several analysis windows.