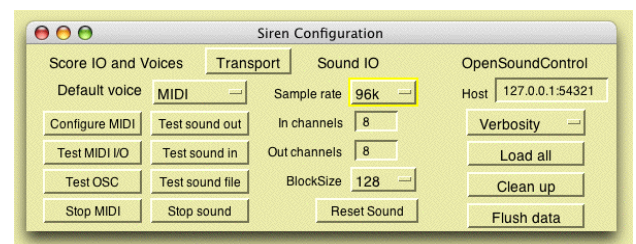
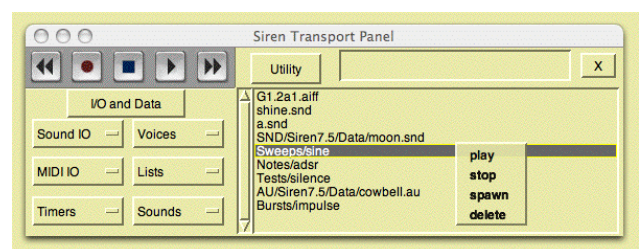
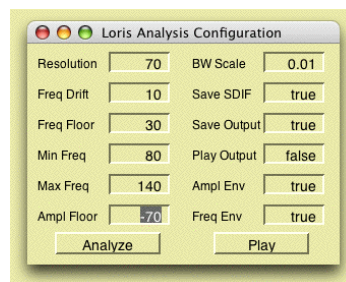
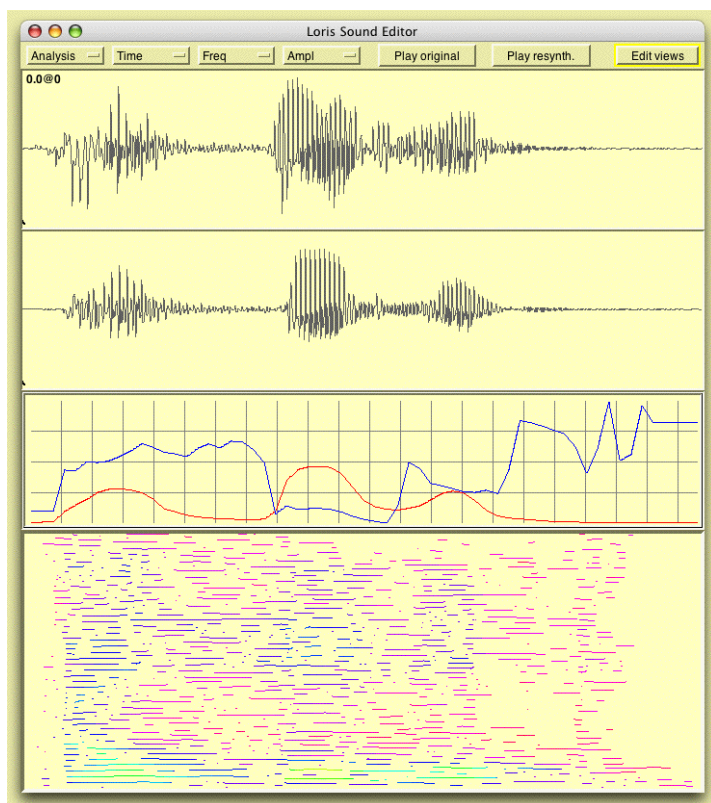


The Big MAT Book:

Courseware for Audio & Multimedia Engineering

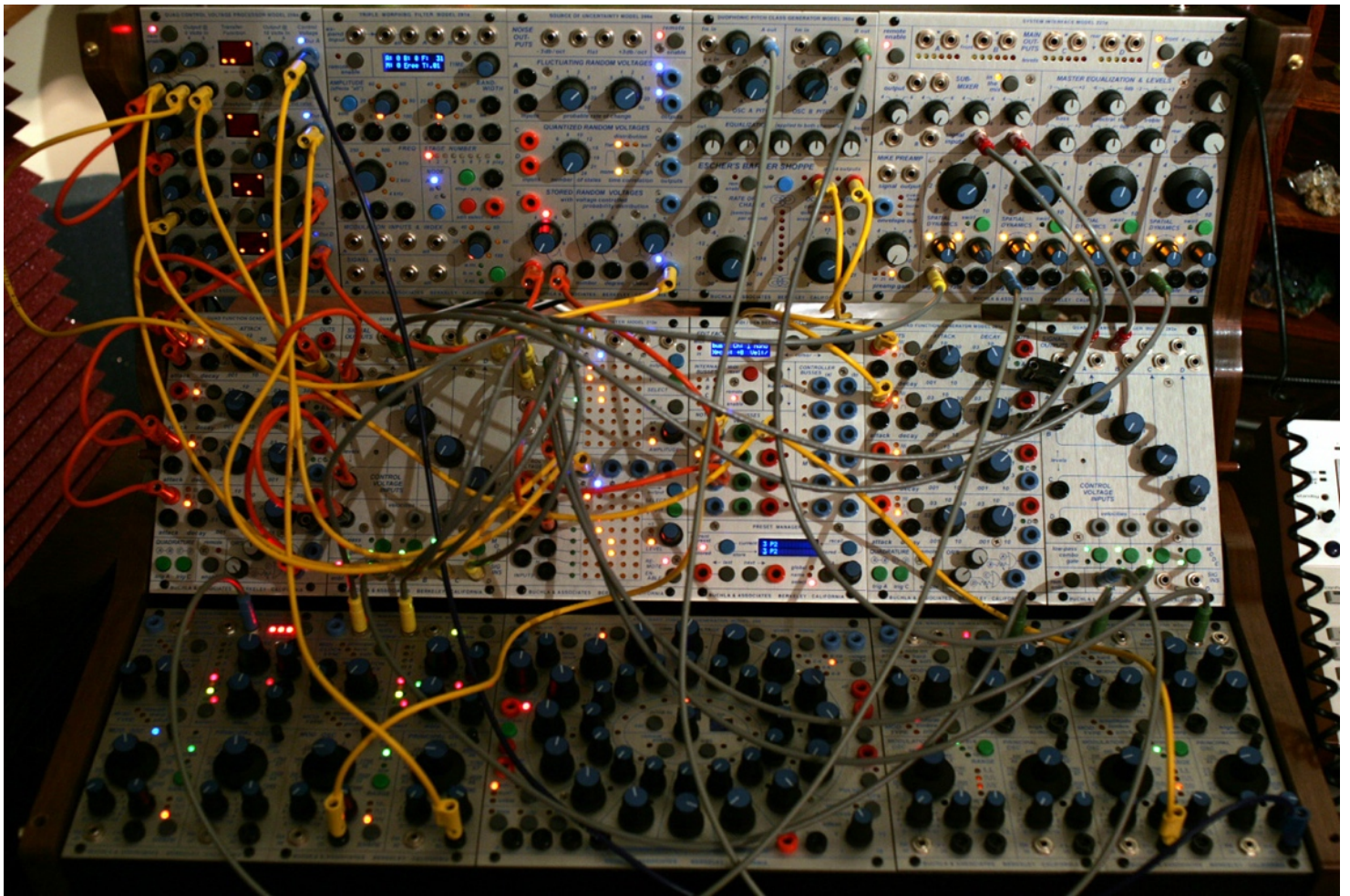
Volume 2: Audio Software

Stephen Travis Pope
Graduate Program in Media Arts & Technology &
Dept. of Music
University of California, Santa Barbara



MEDIA ARTS & TECHNOLOGY PROGRAM





Front cover: Sound processing tools from the Siren 7.5 package in Smalltalk

Inside flap: Buchla D200 synthesizer

Copyright © 2014. Stephen Travis Pope. Some rights reserved. *The Big MAT Book* is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 License.



The Big MAT Book: Courseware for Audio & Multimedia Engineering

Overview

Volume 1: Multimedia Engineering

Volume 2: Audio Software

Volume 3: Audio Hardware

Table of Contents

Volume 1: Multimedia Engineering

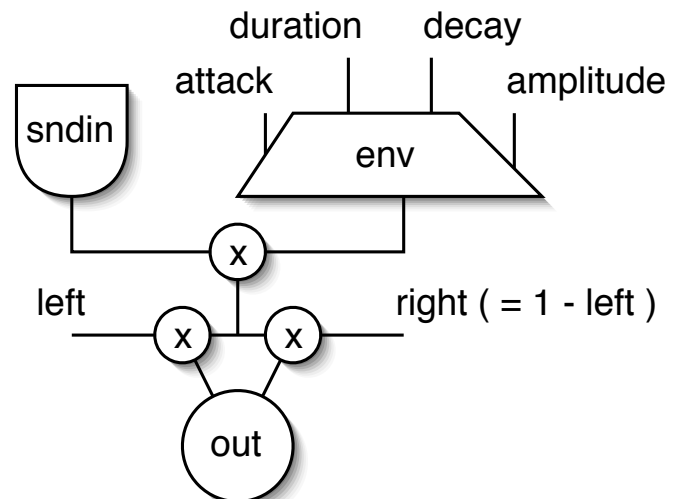
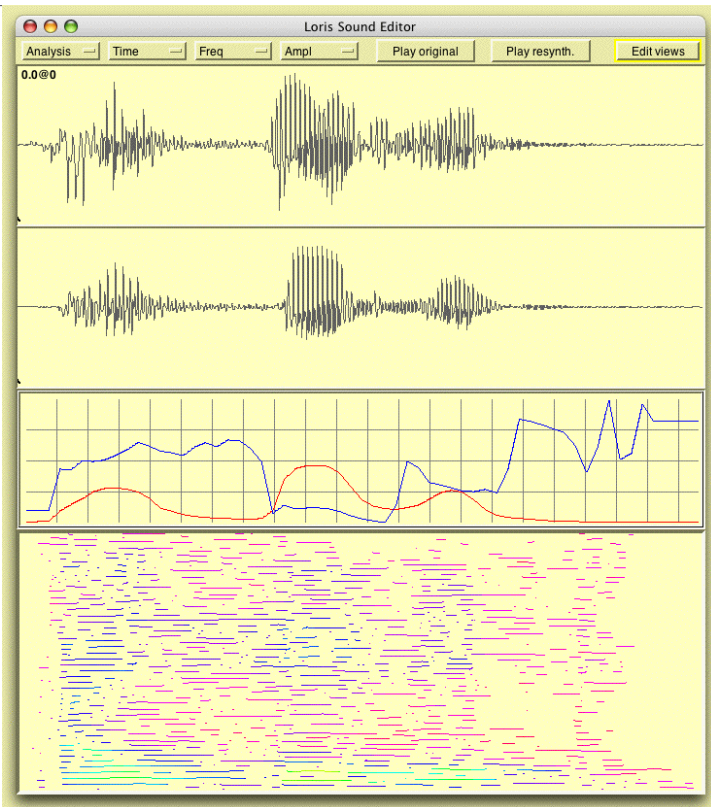
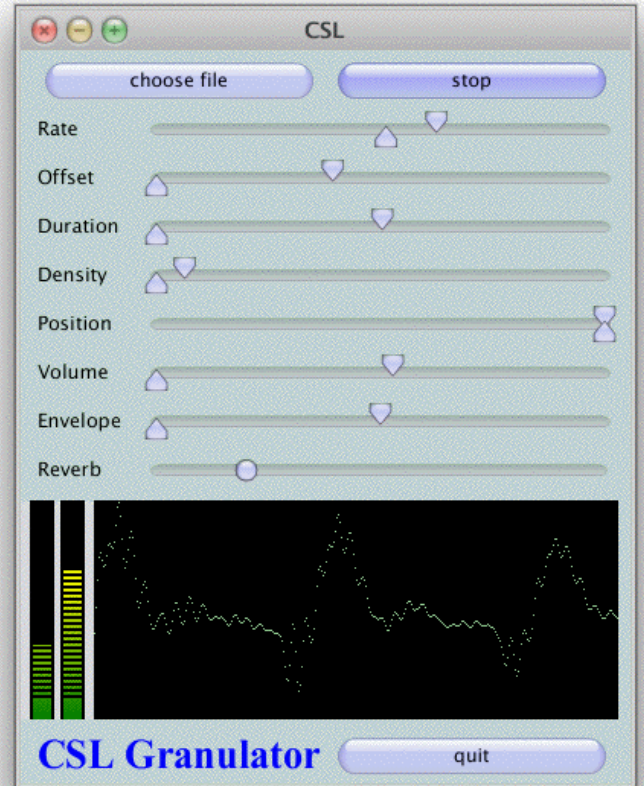
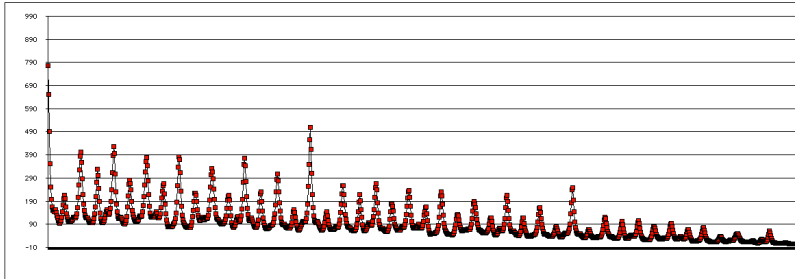
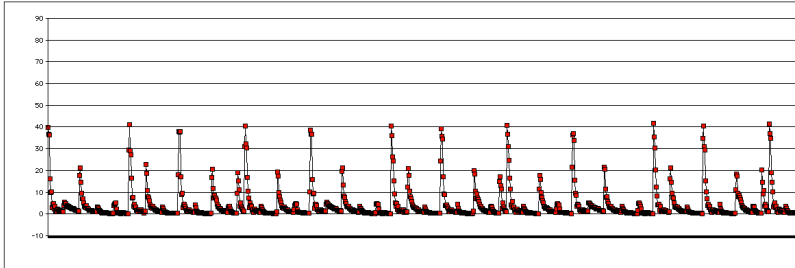
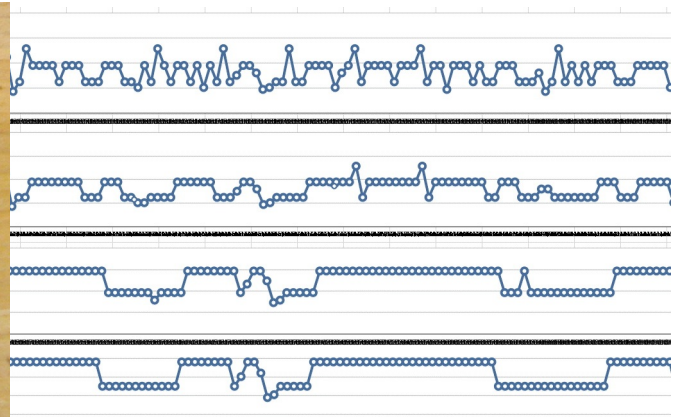
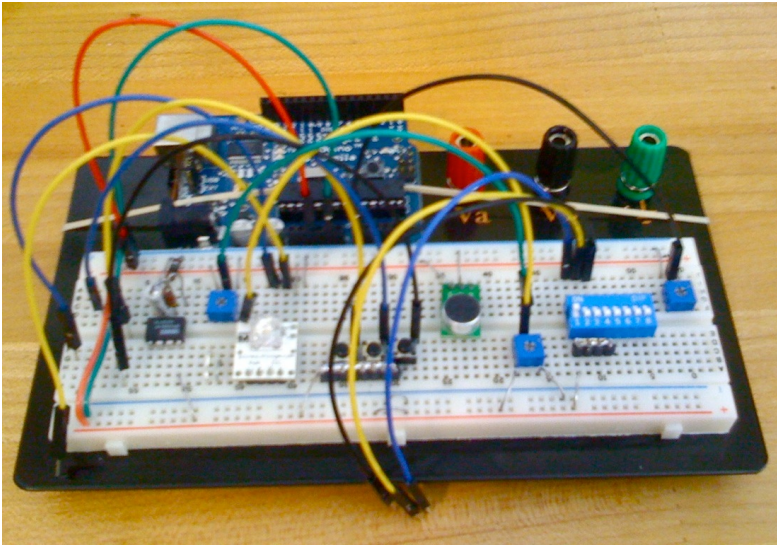
- 1. Survey of Media Engineering & Technology1
- 2. Media Signal Processing115
- 3. Computing with Media Data.....183
- 4. Sensors and Interfaces for Media Art293

Volume 2: Audio Software

- 5. Sound IO and Streaming APIs.....345
- 6. The Spectral Domain: Filter and the FFT393
- 7. Spatial/Surround Sound and Reverb.....423
- 8. Sound Synthesis Techniques.....475
- 9. Synthesis Control and Streaming.....507
- 10. Audio Analysis and Music Information Retrieval.....543

Volume 3: Audio Hardware

- 11. Audiophile Engineering593
- 12. Recording Studio Design and Engineering667



Introduction to the Series “Courseware for Audio & Multimedia Engineering”

Multimedia engineering is a broad and complex topic. It is also one of the fastest-growing and most valuable fields of research and development within electronic technology. The book before you is an anthology of curriculum materials developed over the space of 12 years at the University of California, Santa Barbara for students in UCSB’s *Graduate Program in Media Arts and Technology*.

The *BigMATBook* consists of the presentation slides for twelve ten-week courses, amounting to over 600 hours of presentation time. For each of the twelve courses, the presentation slides are accompanied by the tables of contents of the course readers, and an overview of the example code archives. These resources are available from the HeavenEverywhere web site; see

<http://HeavenEverywhere.com/TheBigMATBook>

The multimedia engineering courses included here cover theory and practice, hardware and software, visual and audio media, and arts as well as entertainment applications. Some of the courses (the first two chapters) are required of all MAT graduate students, and thus must target less-technical and also non-audio-centric students. The bulk of this material, though, consists of elective courses that have somewhat higher-level prerequisites and assume basic knowledge of acoustics and some (minimal) programming experience in mainstream programming languages.

The *BigMATBook* courses borrow liberally from R&D publications by my friends and colleagues, especially Roger Dannenberg, Julius O. Smith, D. Gareth Loy, F. R. Moore, Perry Cook, Adrian Freed, George Tzanetakis, Ross Bencina and Dan Overholt. I want also to express my deepest thanks to my MAT and Music Dept. colleagues JoAnn Kuchera-Morin, Curtis Roads, Clarence Barlow, Matthew Wright, and Matthew Turk, and to the many students who helped these courses evolve, either as course participants or teaching assistants (you know who you are).

Stephen Travis Pope (stephen@HeavenEverywhere.com)

Santa Barbara, California—September, 2009 (updated January, 2014)

Introduction to Volume 2: “Audio Software”

The materials in this volume represent a practical introduction to modern digital audio programming—the six-part MAT 240 *Digital Audio Programming* course sequence. We focus on learning the theory and application of state-of-the-art digital audio software libraries and development tools. Students read the seminal papers from the literature (in the course readers), as well as tutorials on recent industry standards, and will learn to use the current programming APIs and tools using several languages and development platforms. Programming tasks consist of using the C, C++, Java programming languages, and involve digital audio code development tasks on Linux, MS-Windows, the Macintosh platforms. Students are expected to know the basics of digital audio signal representation and processing, and to be proficient in C, C++, or Java (Smalltalk, SuperCollider, LISP or scripting languages are a plus).

The six parts of the *Digital Audio Programming* course sequence (and hence the six chapters in this volume) are:

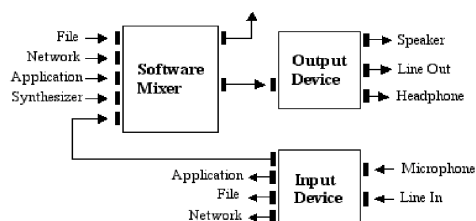
- 240A: Sound I/O APIs
- 240B: Spectral transformations
- 240C: Spatial and surround sound
- 240D: Sound synthesis techniques
- 240E: Control and multi-rate processing
- 240F: Databases & music information retrieval

MAT 240A - Digital Audio Programming: Sound I/O, Streaming and Plug-in APIs (Fall, 2008)

This course is a practical introduction to modern digital audio programming; it is the first part of the six-quarter MAT 240 *Digital Audio Programming* course sequence. We focus on applications development tools, libraries, and interfaces. The central topics are (1) sound I/O and LAN/WAN streaming, and (2) plug-in architectures for multimedia streaming and data processing.

Students will read the seminal papers from the literature, as well as tutorials on recent industry standards, and will learn to use the current programming APIs and tools using several languages and development platforms. The course reader is available at the UCSB book store, and the course web site includes references and code examples. Programming assignments will be given in the C, C++, Java programming languages, and will involve digital audio code development tasks on Linux, MS-Windows, the Macintosh.

Students are expected to know the basics of digital audio signal representation and processing, and to be proficient in C, C++, or Java (Smalltalk, SuperCollider, LISP, and/or XML are a plus). Grading will be on the basis of in-class participation and programming projects.



Course Outline

- Issues in sound and event programming
- Digital sound representations and quality
- File formats and I/O for sound and event data
- Buffers, latency, and jitter in I/O programming
- Real-time I/O and Internet streaming
- Compression formats, codecs, and players
- Browser and virtual machine plug-ins
- APIs for plug-ins to processing programs

Instructor

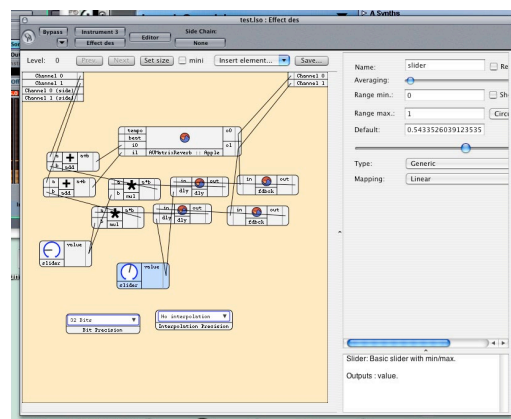
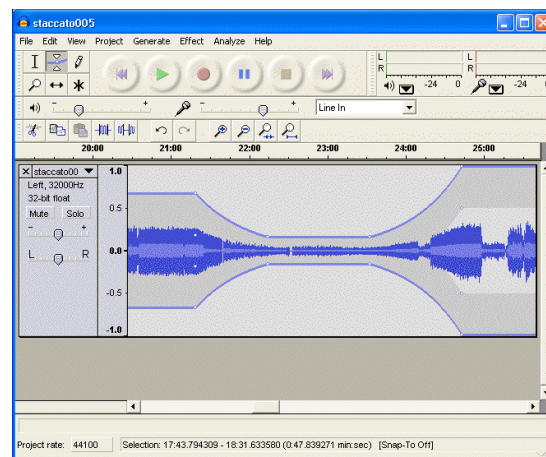
- Stephen T. Pope (stp@mat.ucsb.edu)

Meeting time and place

- Tu/Th 10:00 - 11:50 AM
- Music 2125 (CREATE class room)

Electronic Resources

- Course Web Site
See <http://www.mat.ucsb.edu/240>
- Email Mailing List
Post to 240@mat.ucsb.edu
See <http://www.mat.ucsb.edu/mailman/listinfo/240>



MAT 240A Reader Contents

Reader Sections

Introduction

- Digital audio background & StateOfTheArt
- Coding style and techniques for digital audio
- Software development tools and IDEs

Topic 1: Sound IO APIs

- I/O for sound and event data
- Dimensions of sound I/O APIs
- Using Sound I/O APIs on several platforms

Topic 2: Sound Storage Formats

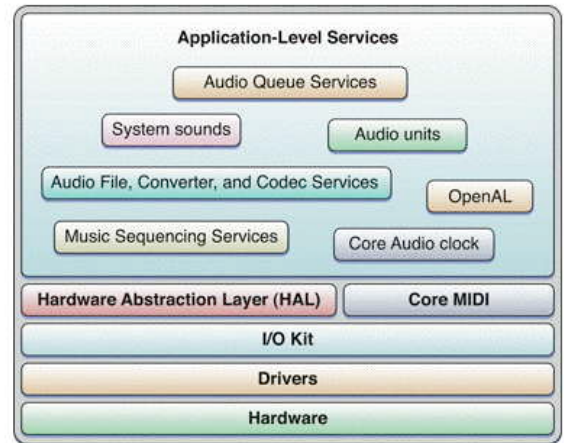
- Sound file formats intro and examples
- Real-time I/O and Internet streaming

Topic 3: Audio Compression Formats

- MPEG & AC-3 Documents

Topic 4: Plug-in Architectures

- Browser and OS Plug-in architectures
- Audio and media plug-in API examples



Readings

Introductory Readings

Digital Audio, James Beauchamp and Robert Maher, "Encyclopedia of Acoustics."
Digital Audio Breaks the Sound Barrier, EDN Magazine.
Beyond the Pale: High-Resolution Audio, Richard Elen, AudioMedia.
TOC of the Roadmap for Sound and Music Computing, S2S² Consortium.
List of Audio Engineering Society (AES) Tutorials.
Emerging Technology Trends in the Areas of the Technical Committees of the AES.
The Next Generation of Audio Communications, JAES.
Audio Networking Applications and Requirements, JAES.
The Software Studio in the Age of Audio Networking, JAES.
Using Game-Audio Tools to Build Audio Research Applications, JAES.
Clear, Efficient Audio Signal Processing in ANSI C, Adrian Freed.
Music Programming with the New Features of Standard C++, Adrian Freed and Amar Chaudhary.
Network Audio Links.
Wikipedia Comparison of Integrated Development Environments.
Freebyte's Guide to Free C++ (and C) Programming Tools.
Tools to Make Tools links.

Topic 1: Sound IO APIs

Apple Core Audio Overview.

Sound Cards, Voice Management, and Driver Models: Ensuring that Your Game Audio Works Optimally and Correctly, Brian Schmidt, Microsoft

Microsoft DirectX Audio Overview.

DirectX Overview Presentation Slides.

Linux OSS API Basics

PortAudio Tutorial, P. Burke, <http://www.portaudio.com/trac/wiki/TutorialDir/TutorialStart>

JACK API FAQ & Overview.

Steinberg ASIO SDK Overview

SNDLib Doc, W. Schottstaedt.

Java Sound API Docs

Topic 2: Sound Storage and File I/O

Machine Tongues XVIII: A Child's Garden of Sound File Formats, S. T. Pope and G. Van Rossum, CMJ 19.1

LibSndFile API doc.

New Applications of the Sound Description Interchange Format, M. Wright et al.

Streaming Stored Audio and Video, James F. Kurose and Keith W. Ross .

UCDavis CS152 Lecture 23: Multimedia Applications, Demet Askoy

SHOUTCAST Streaming API

Code examples: lsf -- list sound files, EBICSF format header file

Topic 3: Audio Compression Formats

A Tutorial on MPEG/ Audio Compression, Davis Pan, IEEE MM.

Overview of MPEG Audio: Current and Future Standards for Low-Bit-Rate Audio Coding, K. Brandenburg & M. Bosi, JAES.

Design and Implementation of AC-3 Coders, Steve Vernon, Dolby

Topic 4: Plug-in Architectures

Wikipedia: Plug-in (computing)

Audio Unit Development fundamentals.

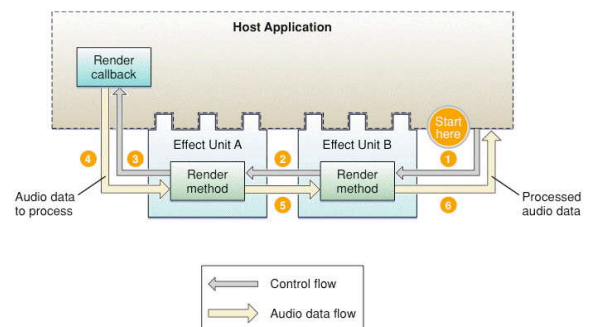
Steinberg VST Plug-in SDK Doc.

Linux Audio Plug-ins: LADSPA, Dave Phillips, O'Reilly.

Writing Browser Plug-ins in Java, Javasoft.

Additional References

See web site and course code ZIP archive.



MAT 240A

Code Archive

- IO APIs
- Documents
- Example Apps
- MIDI
- MP3
- Networking
- Plug-in APIs

▼ AudioAPIs	67.1 MB
▶ ASIOSDK2	1.4 MB
▶ jack-0.109.2	3.7 MB
▶ JavaSound	4.9 MB
▶ libsndfile-1.0.17	12.7 MB
▶ Linux	5.2 MB
▶ Mac Carbon	1 MB
▶ Mac_CoreAudio	7.5 MB
▶ MS-DirectSnd	477.7 KB
▶ nas-1.9.1	6.5 MB
▶ portaudio19	5 MB
▶ rtaudio-4.0.2	1.1 MB
▶ SDL	4.1 MB
▶ sndlib	6.1 MB
▶ JackOSX.0.77.pkg	6.7 MB
▶ Documents	82.8 MB
▶ ExampleApps	63.8 MB
▼ MIDI	2.8 MB
▶ portmidi	2.1 MB
▶ rtmidi-1.0.7	748 KB
▼ MP3	8 MB
▶ bladeenc-0.92.0	1.6 MB
▶ libid3tag-0.15.1b	2.5 MB
▶ libmad-0.15.1b	3.2 MB
▶ MP3.com-ios-api	144.3 KB
▶ lame3.83beta.tar.gz	276.7 KB
▶ mp3_stuff.html	10.6 KB
▶ mp3tools-1.1.tar.gz	34.8 KB
▶ mpg123-0.59r.tar.gz	155.3 KB
▼ Networking	10.8 MB
▶ jrtplib-3.7.0	7.2 MB
▶ librtsp-0.1	493.8 KB
▶ rtptools-1.18	804.5 KB
▶ XStreamRipper.app	2.4 MB
▼ Plug-ins	83.1 MB
▶ iTunes Visual SDK	509.9 KB
▶ Mac_MIDI-plugin	436.8 KB
▶ Netscape	4.4 MB
▶ SoundJam_MP_Dev_Kit	2.7 MB
▶ UglyVSTi X v0.2	2.7 MB
▶ vst_hosts	73.3 KB
▶ vstsdk2.4	21.1 MB
▶ vstsdk3	50.1 MB
▶ wa502_sdk Folder	829.4 KB
▶ Sounds	16.2 MB



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240A: DIGITAL AUDIO PROGRAMMING: THE SERIES, EPISODE 1, TAKE 5 SOUND I/O APIS FALL, 2008

PRESENTATION SLIDES BY STEPHEN TRAVIS POPE,
STP@MAT.UCSB.EDU

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

1

MAT 240A Course Outline

Sound I/O APIs and Audio Streaming

- Introduction
 - Digital audio basics
 - SW development tools
- Topic 1: Sound IO APIs
- Topic 2: Sound Storage/Streaming Formats
- Topic 3: Audio Compression Formats
- Topic 4: Plug-in DSP/Streaming Architectures

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

2

Digital Audio Programming Topics

- Sound/music software
 - (Composition), synthesis, processing, performance
 - Sources, filters, analysis/resynthesis
 - Feature extraction, databases
- Interactive systems and controllers
- Multimedia programming
- Multi-threaded & distributed systems
- Media data streaming and I/O

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

3

The MAT 240 Digital Audio Programming Series

- **240A: Sound I/O APIs**
- 240B: Spectral transformations
- 240C: Spatial and surround sound
- 240D: Sound synthesis techniques
- 240E: Control and multi-rate processing
- 240F: Databases & music information retrieval

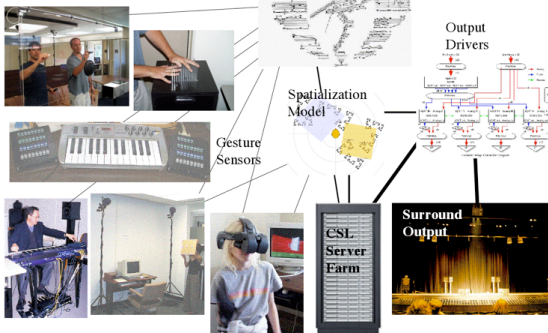
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

4

Why MAT 240?

OSC + CSL in
Pictures



CSB MAT 2

MEDIA ARTS & TECHNOLOGY PROGRAM

5

MAT 240A Overview

- Introduction to sound representation, storage, and interchange formats
- Practical introduction to digital audio programming on current platforms
- Focus on application development tools, libraries, and interfaces for sound I/O and distribution
- Topics
 - (1) Sound and event I/O and streaming
 - (2) Multimedia plug-in architectures and APIs

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

6

MAT 240A Activities

- Students read the seminal papers from the literature, as well as tutorials on recent industry standards, and learn to use the current programming APIs and tools in several languages on a number of development platforms.
- Hands-on C, C++, Java programming
- Digital audio code development tasks on Linux, MS-Windows, Macintosh, and mobile/embedded platforms.
- Students are expected to know the basics of digital audio, and to be proficient in C, C++, or Java (Smalltalk, SuperCollider, python, lua, ruby, LISP, or XML are a plus).
- Grading will be on the basis of in-class participation, topical presentations, and programming projects.

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

7

Course Logistics

- Meeting Times
 - Tues/Thurs 10:00-12:00
 - 2 Lecture/presentations & 1 work/lab session per week
- Materials
 - Readings book, Presentation slides
 - Web Site: <http://www.mat.ucsb.edu/240>
 - Code: http://www.mat.ucsb.edu/240/MAT240A_Code.zip
 - Course mailing list: <http://www.mat.ucsb.edu/mailman/listinfo/240>
- Grading: in-class participation
 - 2-3 topical in-class presentations
 - 2-3 SW projects

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

8

Student Survey

- Name, major,
- Audio/music background
- CS “depth”
- Language & IDE experience
- Other interests

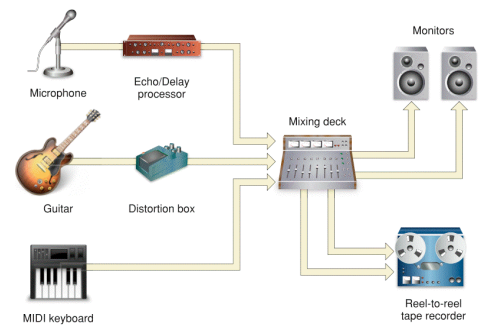
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

9

Why MAT 240? Before:

Figure 1-4 A simple recording studio



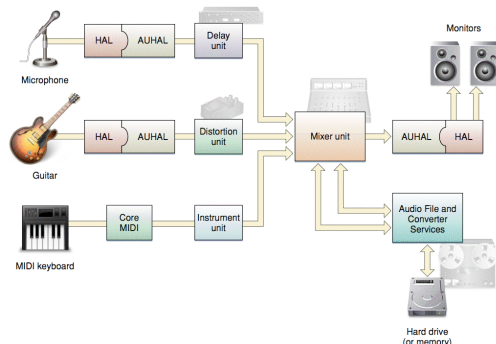
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

10

After: Digital Audio Workstation

Figure 1-5 A Core Audio “recording studio”



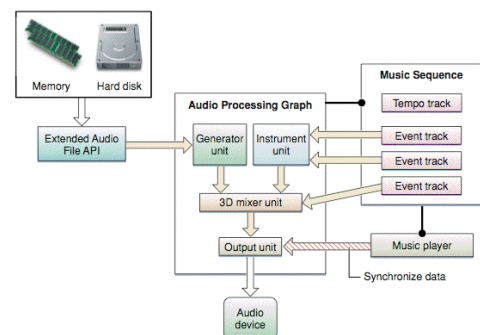
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

11

The Goal in CS Terms

Figure 3-10 Combining audio and MIDI data



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

12

MAT 240A Course Outline

Sound I/O APIs and Audio Streaming

- Introduction
 - Digital audio basics
 - SW development tools
- Topic 1: Sound IO APIs
- Topic 2: Sound Storage/Streaming Formats
- Topic 3: Audio Compression Formats
- Topic 4: Plug-in DSP/Streaming Architectures

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

13

Introduction

- Digital audio background
- State of the art & issues
- Digital sound formats and quality
- Hardware for digital audio
- Software and operating system issues

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

14

SW Development

- The SW development cycle
- Development Tools and IDEs
- SW architectures
- Kinds of APIs
- Tools for the MAT240 series

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

15

Media Programming

- Issues in sound and event programming
- Coding style and techniques for digital audio
- Coding for processing efficiency
- Coding for I/O performance
- Coding for portability

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

16

Topic 1: Sound Storage and Playback APIs

- Sound IO APIs
- I/O for sound and event data
- Dimensions of sound I/O APIs
- Using Sound I/O APIs on several platforms

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

17

Topic 2: Sound Storage & Streaming Formats

- Sound Storage Formats
- Sound file formats intro and examples
- Real-time I/O and Internet streaming

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

18

Topic 3: Compression Formats and Streaming

- Compression formats, codecs, and players
 - MP3, AAC
- Future formats
- Real-time I/O and Internet streaming

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

19

Topic 4: Plug-in Architectures

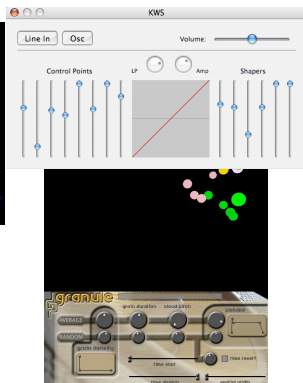
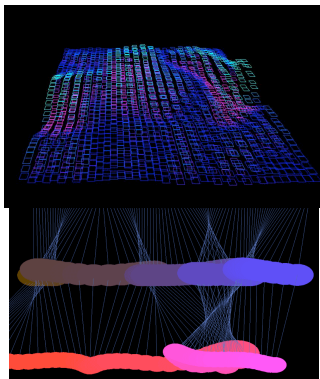
- Audio and media plug-in API examples
 - VST, ASIO, JACK, DirectX, AudioUnits
- Browser and OS Plug-in architectures
 - WWW, Java, VMs

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

20

Results: Student Projects



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

21

Logistics: Languages & IDEs

- **C, C++, Java**
- Smalltalk (Squeak, VisualWorks), LISP (clm, snd), SuperCollider, C#, Chuck, Max/MSP/Pd
- Lua, ruby, python, AppleScript
- **Mac:** XCode, Eclipse, gcc/shell
- **Linux:** gcc, Kdevelop, Eclipse, java sdk, jre
- **Windows:** VisualStudio, Eclipse CygWin/gcc, DevC++
- ***NIX:** Xemacs/gcc/gdb
- Mobile/embedded platforms

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

22

Code Archive

- IO APIs
- Documents
- Example Apps
- MIDI
- MP3
- Networking
- Plug-in APIs

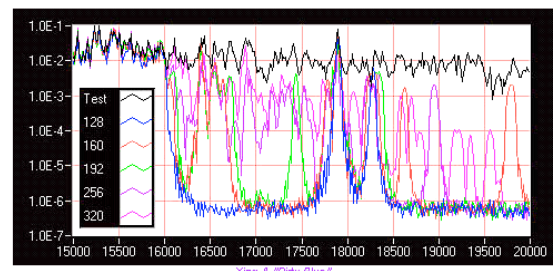
AudioAPIs	67.1 MB
ASIO SDK2	1.4 MB
Jack-0.109.2	3.7 MB
javaSound	4.9 MB
libsndfile-1.0.17	12.7 MB
Linux	5.2 MB
Mac Carbon	1 MB
Mac CoreAudio	7.5 MB
MS-DirectSnd	477.7 KB
mas-1.9.1	6.5 MB
portaudio19	5 MB
rtaudio-4.0.2	1.1 MB
SDL	4.1 MB
sndlib	6.1 MB
JackOSX-0.77.pkg	6.7 MB
Documents	82.8 MB
ExampleApps	63.8 MB
MIDI	2.8 MB
portmidi	2.1 MB
rtmidi-1.0.7	748 KB
MP3	8 MB
libmadenc-0.92.0	1.6 MB
libid3tag-0.15.1b	2.5 MB
libmad-0.15.1b	3.2 MB
MP3.com-iso-apl	144.3 KB
lame3.83beta.tar.gz	276.7 KB
mp3_stuff.html	10.6 KB
mp3tools-1.1.tar.gz	34.8 KB
mpg123-0.59r.tar.gz	155.3 KB
Networking	10.8 MB
grplib-3.2.0	7.2 MB
librip-0.1	493.8 KB
rtptools-1.18	804.5 KB
MSstreamlupperapp	2.4 MB
Plug-ins	83.1 MB
iTunes Visual SDK	509.9 KB
Mac MIDI-plugin	436.8 KB
Netscape	4.4 MB
SoundJam_MP_Dev_Kit	2.7 MB
uglyVST-X-v0.2	2.7 MB
vst_hosts	73.3 KB
vstsdk2.4	21.1 MB
vstsdk3	50.3 MB
wasO2_sdk Folder	829.4 KB
Sounds	16.2 MB

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

23

Topic 0: Digital Audio Programming Introduction



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

24

Introduction: Digital Audio & Computer Music Review

- Digital audio background & state of the art
- AES survey & tutorials
- Digital sound formats and quality
- Hardware for digital audio
- Operating system issues

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

25

Introductory Readings

- Digital Audio, James Beauchamp and Robert Maher, “Encyclopedia of Acoustics.”
- Digital Audio Breaks the Sound Barrier, EDN
- Beyond the Pale: High-Resolution Audio
- TOC of the Roadmap for Sound and Music Computing, S2S2 Consortium.

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

26

Introductory Readings 2

- List of Audio Engineering Society (AES) Tutorials
- Emerging Technology Trends in the Areas of the Technical Committees of the AES
- The Next Generation of Audio Communications
- Audio Networking Applications and Requirements
- The Software Studio in the Age of Audio Networking
- Using Game-Audio Tools to Build Audio Research Applications

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

27

Introductory Readings 3

- Clear, Efficient Audio Signal Processing in ANSI C, Adrian Freed
- Music Programming with the new Features of Standard C++, Freed and Chaudhary
- Network Audio & IDE Tool Links
- Wikipedia Comparison of IDEs
- Freebyte’s Guide to Free Programming Tools

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

28

Digital Audio Technology

- Introduction, history
- Formats and interchange
- DA/AD technology
- Projection technology
- Low-level support for audio

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

29

Digital Audio Background

- (Beauchamp & Maher chapter)
- 1950s-60s: Bell Labs “acoustical compiler”
- 1970s: MusicV and descendants (parallel analog synth developments), MILs, synth. techniques
- 1980s: HiFi DACs, Sony F-1, CD, DAT, comp. languages, RT SW, PCs & workstations
- 1990s: MD, MP3 (more analog), WWW, etc.

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

30

The State of the Art

- See AES *Emerging Trends* document
- Current: 24/96/5.1 formats
- Next-generation formats
- Software sound synthesis (SWSS)
- WAN audio streaming, VOIP
- DAW software systems
- FireWire/USB multichannel interfaces
- Control surface hardware

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

31

Encoding Sound

- Formats and convertors
- Time- and spectral-domains
 - FFT/IFFT-based codecs
 - LPC/filter-based codecs
 - Deterministic + stochastic representation
- Masking and data reduction
 - MP3
- Statistical methods
 - MLP

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

32

Software Sound Synthesis

- Instrument (procedure) + score (fcn-call list) model
- Various synthesis techniques
- Various language models
- Batch model
- Interactive/streaming model
- Relation to MIDI and control protocols

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

33

Digital Audio Formats and Quality

- Sampling and quantization
- SR: (Fs) 8-384 KHz (typ. 44100)
 - Sample rate and signal bandwidth
- Format: 4-128 bits (typ. 16-24)
 - Sample format and quantization noise
 - Dithering, noise-shaping, and over-sampling
 - Linear Int, "PCM"
 - DSD and SACD 1-bit

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

34

Sampling Rates

- Early: 10 kHz, 12-bit samples (determined by 1960s DAC HW)
- See table in Pope & van Rossum
- Modern rates: 5500 - 44100 - 192000 Hz
 - Common divisors of high frequencies
 - Common multiples of powers of two or other sampling frequencies

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

35

Sample Formats, Resolutions

- 8-bit compressed - 24-bit linear - 32- or 64-bit f-p
- Format: linear, μ -law, floating-point
- Resolution: 4-64-bits, relationship to word size
- Perception and limen testing (70s, 90s)
- Linearity, monotonicity, residual noise of contemporary playback systems
- 1-bit and "integrator" DACs

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

36

Sound Formats and Bandwidths

- Surround HiFi (8 @ 24/96) 18 M b/s
- “CD-quality” (2 @ 16/44) ~ 1.4 M b/s
 - Losses (rel. to 24/96) easily heard over good systems
- MP3 (DCT, masking) ~128 k b/s (VBR)
 - Debatable losses relative to “CD-quality” over “mid-fi” systems for ≥ 128 k b/s bandwidth
 - Obvious losses for lower bit-rates
- RealAudio (CELP) ~ 14 k b/s
 - Monophonic, “telephone-quality”
- Best speech compression ~ 1.5 k b/s
- Per-generation, audio is 50-100 times smaller than video

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

37

The NeXT/Sun File Header (1986)

```
/* .snd file header */
typedef struct {
    int magic;           /* magic number SND_MAGIC */
    int dataLocation;    /* offset/pointer to the data */
                        /* (header can be >28 bytes) */
    int dataSize;        /* number of bytes of data */
    int dataFormat;      /* the data format code */
    int samplingRate;    /* the sampling rate */
    int channelCount;    /* the number of channels */
    char info[4];        /* optional text information */
} SndSoundStruct;

(dataFormat is a #define'd macro)
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

38

Multi-channel Formats

- Stereo
- Quad (1970s)
- CM formats
- ADAT (multiples of 8, octophonic circle, corners-of-a-cube)
- Home-theater and 5.1
- Extended 5.1 = 7.1, 10.2
- Pluriphonic systems, AlloSphere

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

39

Hardware for Digital Audio

- ADC and DAC structure & construction
- Convertor quality issues
 - Digital signal massaging (jitter)
 - Convertor properties (linearity)
 - Anti-aliasing (or reconstruction) filter
 - Analog driver amp. stage
- DACs for computers
- Sound cards
 - Convertor, synth. chip, interfaces

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

40

Hardware Interfaces

- DAC/ADC
 - 2/4/5.1/7.1/8/n*8 channels
 - 1/8", 1/4" TS/TRS, RCA, XLR
- Digital I/O
 - S/P-DIF, AES/EBU, TOSLINK, AT&T
 - ADAT, n*ADAT, S-Mux
 - USB-Audio
 - FireWire/IEEE1394-Audio
 - CORBA A/V streaming
- Web references, SndCard reading



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

41

Operating System Issues

- SWSS support (RT facilities, lwp, IO, protocol stacks, etc.)
- Sound file storage (then and now)
- Sound I/O streaming
 - Play & record programs
- Network streaming
- Utility programs
 - Isf, sndinfo, cpsf, etc.

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

42

OS Overview

- CPU & I/O management
- Memory/cache management
- Interrupt-handling, timers
- I/O drivers, DMA
- APIs for OS interfaces

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

43

UNIX Driver Model

- Character & block device tables with function pointers for open(), close(), read(), write() (possibly async), and ioctl()
- ioctl() takes command flags and var-arguments for device-specific control
- Error codes returned as errno; use perror() to print (no out-of-thread exceptions)
- Utility programs manipulate device “files” and read/write pointer data

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

44

UNIX Driver Special Files

```
[stp:/dev] ll *disk*
brw-r----- 1 root operator 14,  0 Sep  9 00:18 disk0
brw-r----- 1 root operator 14,  1 Sep  9 00:18 disk0s1
brw-r----- 1 root operator 14,  2 Sep  9 00:18 disk0s2
brw-r----- 1 root operator 14,  3 Sep  9 00:18 disk0s3
crw-r----- 1 root operator 14,  0 Sep  9 00:18 rdisk0
crw-r----- 1 root operator 14,  1 Sep  9 00:18 rdisk0s1
crw-r----- 1 root operator 14,  2 Sep  9 00:18 rdisk0s2
crw-r----- 1 root operator 14,  3 Sep  9 00:18 rdisk0s3
```

- 2 tables: character and block devices
- Entry 14 is a list of the ptrs to the disk's open/close/read/write/ioctl functions
- C: fopen("/dev/disk0s1", "r") => b_table[14].open(...)

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

45

Other OS/Driver Models

- Macintosh toolbox “managers”
- Object-oriented models
 - Design: buffer, sound, stream, device
 - Java, Siren, STK, cIm, Jsynth, MusicKit, ...
 - BeOS I/O objects
- Plug-in architectures
 - Browser plug-ins
 - Streaming app. Plug-ins

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

46

SW Development

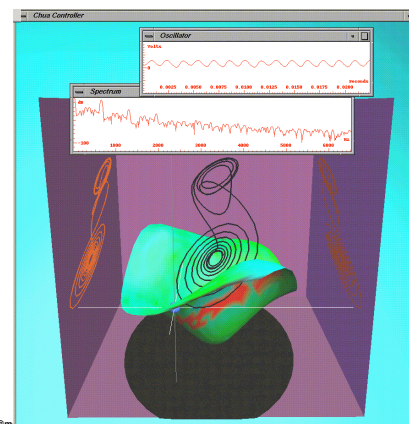
- The SW development process
- Compilers and interpreters
- Development tools
 - Editor (text, code) (file-, module-, object-based)
 - Compiler (file, module, class structure)
 - Make (compiler scripting and dependency)
 - Debugger (execution, breakpoints, variables)
 - Source code control, versioning, sharing, configuration management
- Integrated Development Environments (IDE)
- Details to follow... (see also MAT 201B)

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

47

Topic 0: Media Programming



CSB MAT 240A stephen@m

MEDIA ARTS & TECHNOLOGY PROGRAM

48

Topic 0: Media Programming

- File formats and I/O for sound and event data
- Using Sound I/O APIs
- Processor and language issues
- Programming guidelines

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

49

File Formats and I/O

- (CMJ sound file format compendium)
- Sound sample and file format
- Sound file headers
- Tagged file types
- Standards:
 - SND (NeXT, Sun, Linux, CARL, EBICSF)
 - AIFF (SGI, Apple, Linux)
 - RIFF/WAV (MS)
 - AU (sun, speech, net)
 - MP3/AAC (compressed formats)

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

50

Using Sound I/O APIs

- Sound file APIs
 - Formats, annotation, streaming, interchange
- Sound I/O APIs
 - Play/record programs, I/O buffering
- Codec libraries and utilities
- Content streaming APIs
 - IP, HTTP
- Plug-in APIs for WWW browsers, OS
 - Window management, event call-backs

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

51

Example 0: CARL (UCSD, 1980)

- Platform: DEC VAX-11/780, BSD UNIX
- Low-level support (for C)
 - File I/O libraries (CCSS = Cylinder-Contiguous Soundfile Storage system)
 - API: float streaming over UNIX pipes with procom headers
- Utilities
 - CCSS/CARL sound file system + utilities
 - Cmusic SWSS compiler
 - Filters, reverberators, space, etc.
 - E.g., `wave -args | gain 0.5 | spect | sndin file.snd`
- UI Framework

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

52

Later CARL progress

- Mid-1980s
 - PiDet, MIDI
 - Sound-in
 - Port to Sun workstations
 - Use of BSD file system
 - X-based GUIs
- 1990s
 - Pcmusic
 - Descendants

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

53

CARL Descendants

- Sndlib (CCRMA)
- EBICSF (Berkeley, IRCAM, Princeton, UCSB)
- PortAudio (Mills/AUS, CMU)
- Composer's Desktop Project
- MPEG7
- MusicXML
- MP3 ID3

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

54

Media Programming Issues

- Issues in sound and event programming
- Coding style and techniques for digital audio
- Coding for processing efficiency
- Coding for I/O performance
- Coding for portability

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

55

Media Programming

- (Freed readings)
- General programming issues
 - Data structs. and I/O
- Vector and block numerical processing
- Double-buffered I/O
 - Call-backs, async/blocking I/O
- Event- and I/O-driven programming
 - Control, GUIs
- Streams and DSP pipes

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

56

Processor Issues

- Integer vs. f-p performance and formats (endianness)
- CISC vs. RISC vs. DSP
 - Separate DSP issue
 - Active sound cards
- Special-purpose processors
 - Synth. Techniques
 - Convolvotron
 - FFT/IFFT

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

57

C, C++ Language Issues

- Data types: 8-64-bit ints, 32/64-bit floats
- Data structures: sample buffer [short *samps], header, ...
- Operations: math, vector ops, storage
- Libraries and APIs
- Versions of C/C++/ObjC/Java/C++
- Platform portability

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

58

Adrian's Advice

- Use floats if feasible for sample math
- Use longs or doubles for time values
- Need > 16 bits for frequency values
- Use accuracy-preserving math
- Watch out for processors with slow integer performance (i.e., using f-p ALU)
- Watch out for processors with slow f-p performance (e.g., Pentium)
- Know your compiler!

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

59

Adrian's Advice 2

- Test the built-in library functions
- Use casts and parentheses
- Watch malloc()
- Use appropriate language features
 - Exceptions
 - Threads
- Know your compiler optimizations!
 - Register variables
 - for() {} vs. while() {}

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

60

OSS Programming Guidelines

- Use API macros
- Check return error codes
- Don't assume initial conditions, set device parameters
- Be careful in interrupt call-backs
- Cast pointers and ioctl() arguments

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

61

Programming for Portability

- Several options
 - #ifdefs for all platforms
 - Macros for platform-specific operations
 - Use portability APIs (e.g., CygWin)
 - Obviously choose least-common denominator functionality and formats
- Examples
 - PortAudio, Sndlib, JUCE, OpenAL, JACK, libSndFile, RTaudio, CSL

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

62

Sound Storage and I/O

- Sound file formats
 - 3 standards: snd/next, aiff/aifc, wav/riff
 - Many obscure: MOD, IRCAM/BICSF, au, sphere
- Sound file I/O programming
 - Example: snd-format lsfc for UNIX
 - Extended EBICSF version headers
 - Sndinfo from sndlib
 - Other examples -- play/record

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

63

Reading APIs

- Data structures, class structure
- Methods and signatures
- Exceptions, error codes, limits, ranges, boundary conditions, legal input, etc.
- Doc and examples
- Protocols and p-APIs
- IDL descriptions
- String example from MAT 201B

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

64

Software Development Tools

- Basic steps
 - Compile, assemble, link, run-time start-up
 - Support for processes
 - Manage code, build & versioning, debugger, GUIDE, code-sharing
- Integrated Development Environments (IDEs)
 - Project/target concepts
 - Setting-up & building a target
 - Debugging & breakpoints
 - GUI development
 - SCCS links

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

65

Learning a new IDE

- Target language
- Host/target processor/OS
- Tool GUI style
 - Editor
 - Compiler
 - Debugger
- Package discipline
 - Project
 - Makefile

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

66

IDE families

- vi/emacs + shell/make/cc + adb (1970s)
 - Linux, CygWin, OSX shell
- Project + “integrated” (turbo) tools
 - VisualStudio, XCode, KDevelop, Eclipse, ...
- Incremental, rapid-turn-around IDE
 - Smalltalk, VisualAge/Java, CommonLISP, MATLAB, scripting languages

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

67

How did Grandpa do it? Make and Makefiles

```
# Declare an object file list (this is a comment)
OBJECT_FILES = CSL_Core.o Gestalt.o Variable.o

# Declare some compiler flags
CCFLAGS = -O2 -I../Kernel -I../CSL/Sources

# Add a rule or dependency (target, source, command)
CSL_Core.o: ../Kernel/CSL_Core.cpp
    $(CC) -c $(CCFLAGS) ../Kernel/CSL_Core.cpp

# Add a “target” or executable
beep: $(OBJECT_FILES) Beep.o
    $(CC) $(OBJECT_FILES) Beep.o $(LD_FLAGS) -o beep
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

68

Executing Make in a Shell

```
#make (type in shell command)
gcc -c -O2 -I../Kernel ../Kernel/CSL_Core.cpp
...
gcc -c -O2 -I../Kernel ../Tests/Beep_demo.cpp
gcc CSL_Core.o CGestalt.o [...] Beep_demo.o
-L/usr/local/lib -lm -lportaudio
-L/usr/lib/gcc/darwin/3.3 -lstdc++
-o beep_demo
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

69

Debugging

- Running an executable within a debugger allows you to:
 - control execution (breakpoints, single-stepping)
 - Examine system state just before/after an error (stack, variables)
- Examples: gdb, ddd, etc.
- Debuggers in IDEs
 - Issues, differences...

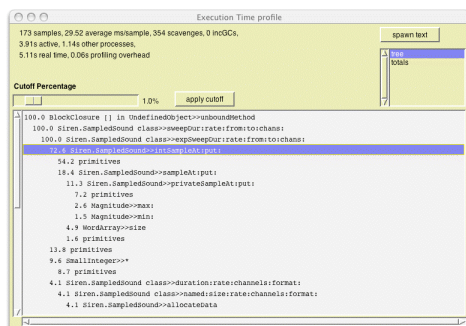
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

70

Profiling

- Run the program and “observe” where it spends its time; print out some meaningful representation of this



CSB MAT 240A step

MEDIA ARTS & TECHNOLOGY PROGRAM

71

Integrated Development Environments (IDEs)

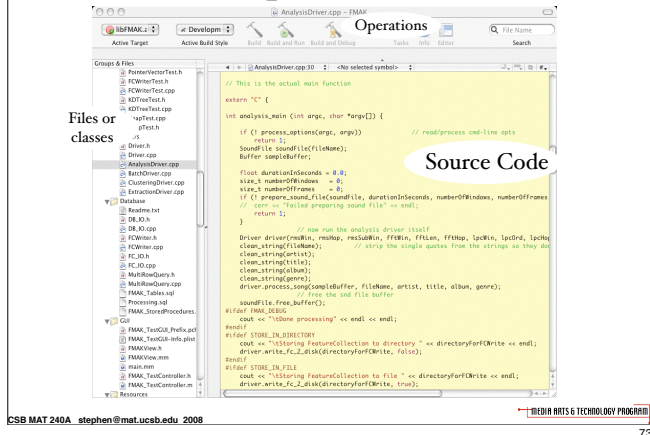
- Project/file management
 - File trees, libraries and frameworks
- Source code editing
 - Syntax-aware code editor
 - File/class browser and cross-referencing
- Compilation/linking
 - Rapid-turn-around compiler (?)
- Execution/debugging
 - Execution environment and I/O
 - Breakpoints and single-stepping
- Packaging/deployment
 - Source code management system interface
 - Development vs. deployment compiler options

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

72

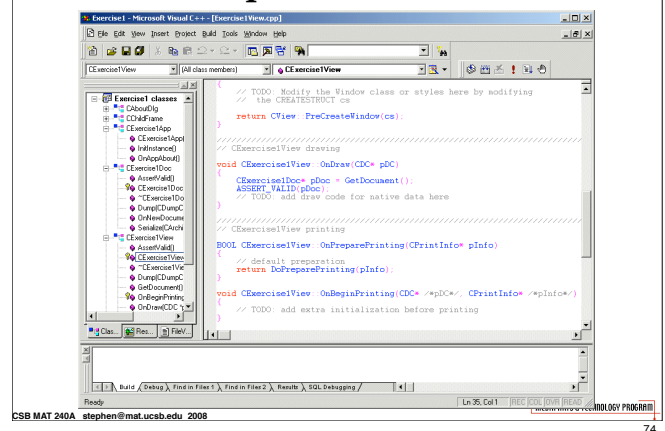
IDE Example: Xcode Editor



CSB MAT 240A stephen@mat.ucsb.edu 2008

73

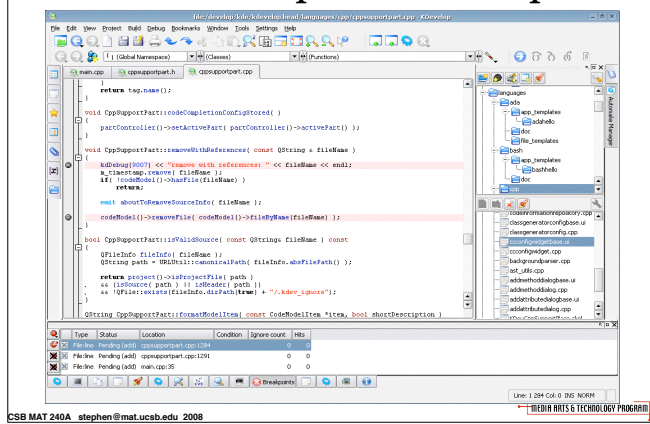
IDE Example: MS Visual C++



CSB MAT 240A stephen@mat.ucsb.edu 2008

74

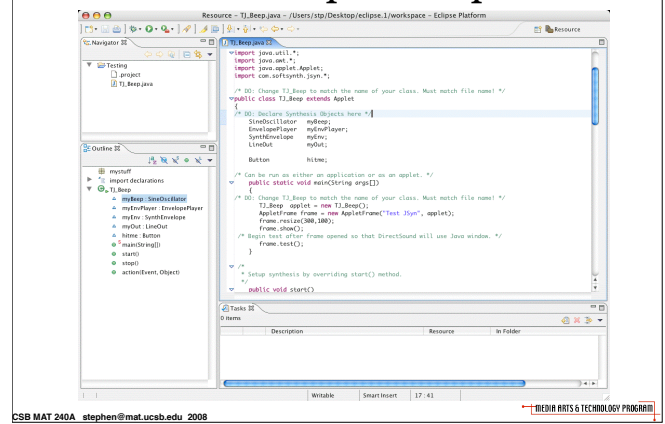
IDE Example: KDevelop



CSB MAT 240A stephen@mat.ucsb.edu 2008

75

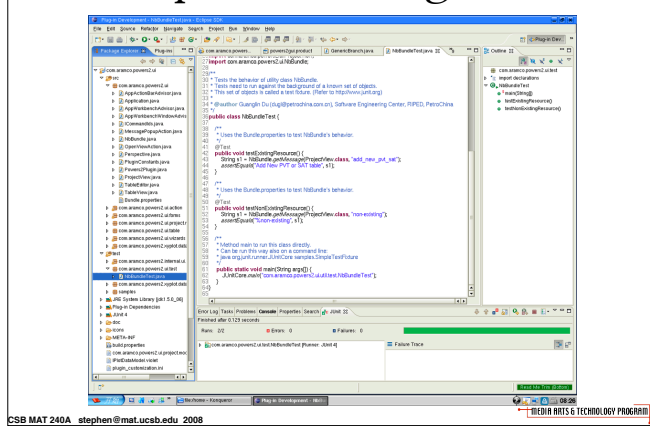
IDE Example: Eclipse



CSB MAT 240A stephen@mat.ucsb.edu 2008

76

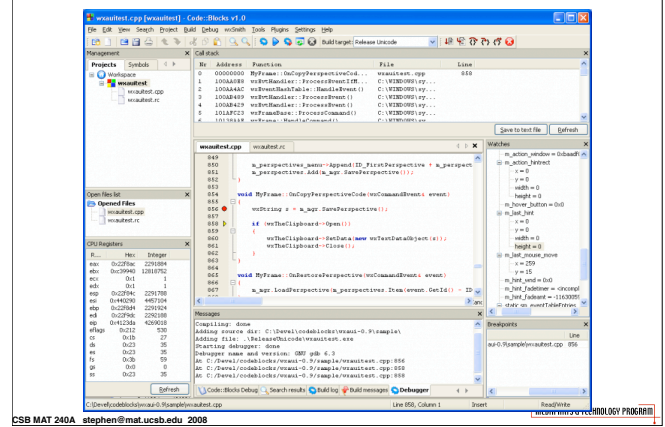
Eclipse on RedFlagLinux



CSB MAT 240A stephen@mat.ucsb.edu 2008

77

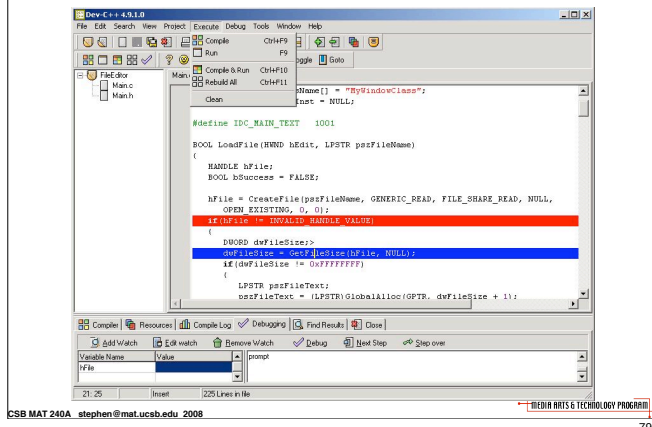
CodeBlocks IDE



CSB MAT 240A stephen@mat.ucsb.edu 2008

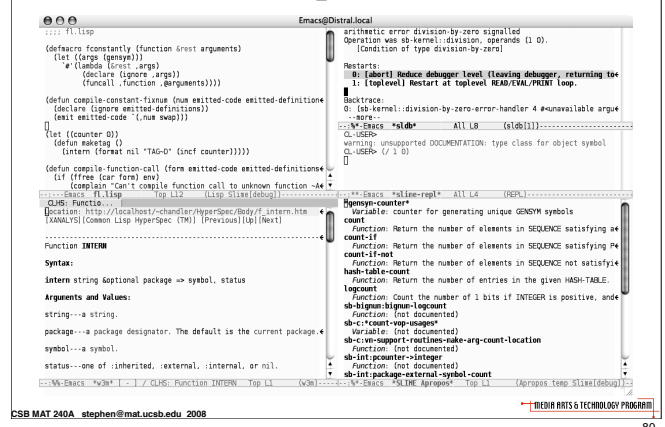
78

Dev-C++ IDE



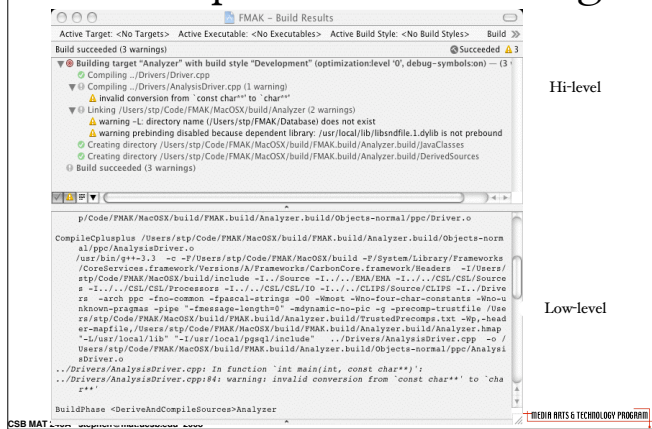
79

IDE Example: XEmacs



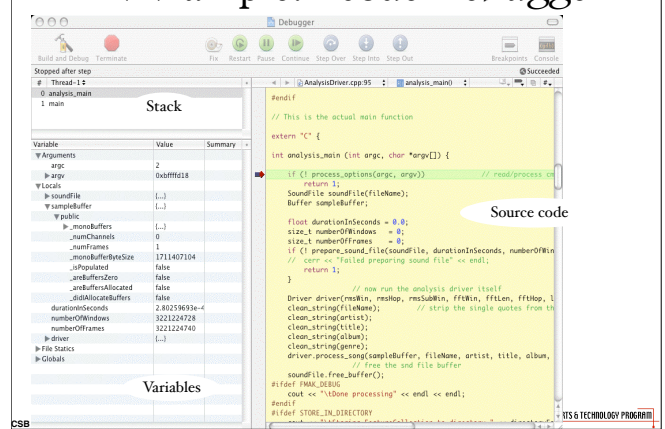
80

IDE Example: Xcode Build Stages



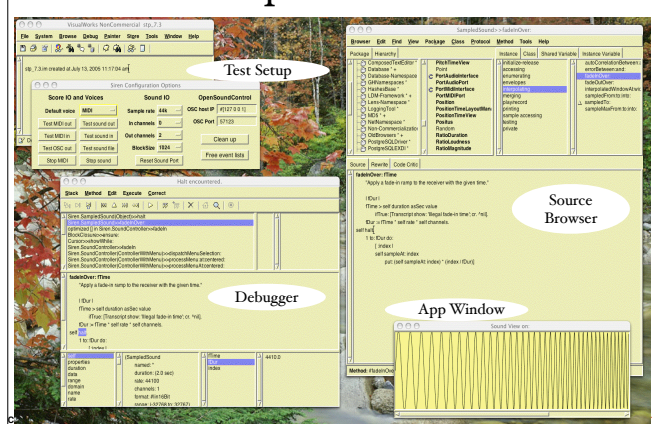
81

IDE Example: Xcode Debugger



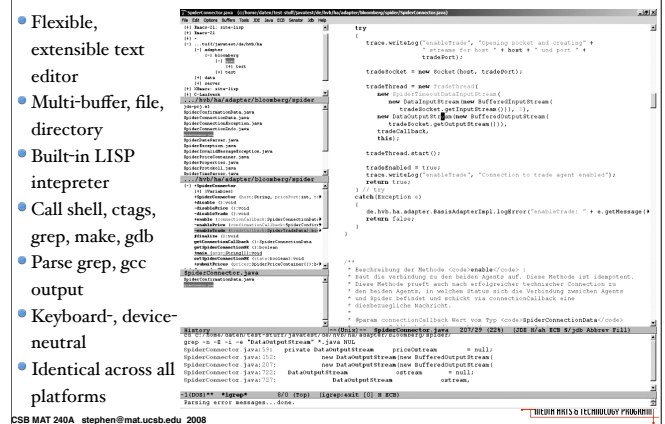
82

IDE Example: Smalltalk-80



83

In Praise of Emacs



84

Installing a Linux Package

- Down-load a “tar-ball” (gzip-compressed tar archive); unpack it
 - `tar -xvfz tarball.tgz`
- Run configuration shell script
 - `./configure`
- Build target
 - `make`
- Install target
 - `sudo make install`
- Building distributions: `conf/make/package`

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

85

Review

- DASP Intro and state of the art
- AES tutorials
- Coding idioms and guidelines
- Survey of tools and IDEs
 - IDE project set-up, build/debug
 - IDEs & frameworks

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

86

Coding Exercises

- Hello world on several platforms
- Hello world with GUI on several platforms
- C/C++ APIs
 - File I/O, threads
 - GUI libraries & tools, sockets & networking
- Frameworks
 - JUCE, .NET, Cocoa
- Efficient, clean signal processing style
- Writing call-back functions
 - Events
 - Audio/MIDI

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

87

What's next?

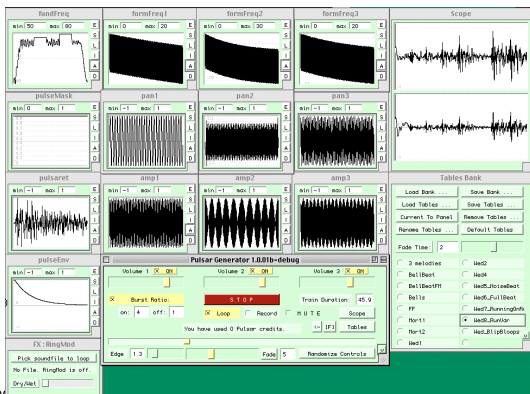
- Topic 1 Readings
 - MAC OSX CoreAudio Introduction
 - Microsoft DirectX Overview
 - Sound Cards, Voice Mgmt, and Drivers
 - Linux OSS & JACK API Basics
 - Portaudio, ASIO, SndLib
 - Java Sound API

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

88

Topic 1: Sound APIs



CSB MAT 240A

MEDIA ARTS & TECHNOLOGY PROGRAM

89

Topic 1: Sound IO APIs

- CoreAudio, AudioUnits
- DirectX, DirectSound
- OSS, JACK
- PortAudio
- RTAudio
- SndLib
- ASIO
- JavaSound

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

90

Sound API Readings

- MAC OSX CoreAudio Introduction
- Microsoft DirectX Overview and Plug-in Spec.
- Sound Cards, Voice Management, and Driver Models, Brian Schmidt, example: playsound
- Portaudio tutorial
- Linux OSS & JACK API Basics
- Steinberg ASIO SDK
- Java Sound API

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

91

3 Dimensions of Sound APIs

- How sample buffers are handled
 - Sample blocks (short * samples[])
 - Opaque pointers (void * data)
 - I/O Streams
- I/O Model
 - UNIX model, others
 - Sync/async I/O
 - Write vs. call-back
- API structure
 - Functional
 - Object-oriented

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

92

Call-back & Event APIs

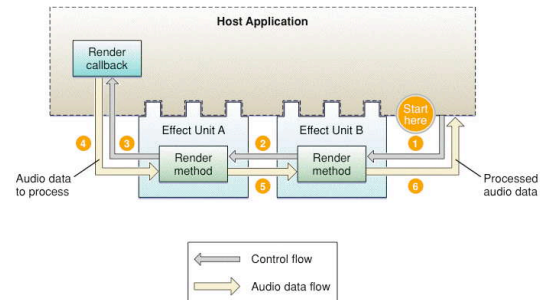
- App supplies some start-up code
 - Open special device
 - Call API initialize fcn
- Register a function pointer for OS call-backs
 - Exact signature defined by API
 - Takes time-stamp and mem ptrs from OS
 - May write to OS data
 - Has a free void* argument you can fill in

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

93

Host App & Plug-in Call-backs



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

94

Mac OSX CoreAudio

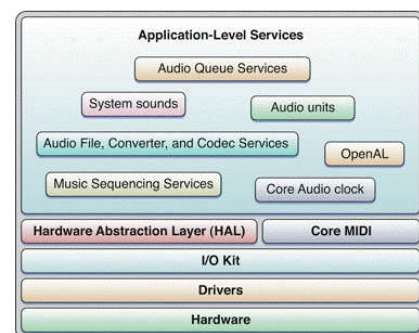
- Old Mac World(s)
 - SoundManager, MIDIManager
 - ASIO, OMS, FreeMidi
- Mac OSX (Darwin)
 - Mach/BSD base
 - Apple Frameworks
 - CoreAudio, CoreMIDI
 - QuickTime
 - OpenGL

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

95

Core Mac OSX Services



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

96

CoreAudio

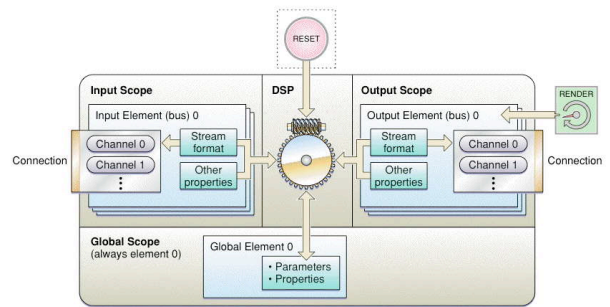
- Plug-in interfaces for audio synthesis/processing
- Support for a wide variety of audio file and data formats
- Plug-in interfaces for custom file and data formats
- A modular approach for constructing signal chains
- Scalable multi-channel input and output
- Easy synchronization of audio and MIDI data
- Interface to all built-in and external hardware devices, regardless of connection
- Straight-C API, ObjectiveC, C++, Java wrapper objects

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

97

OS-level DSP Processing



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

98

CoreAudio Structure

- Hardware Abstraction Layer
- AudioUnits
- Audio Toolbox
- QuickTime
- JavaMF API
- SoundManager/Classic/Carbon

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

99

APIs in CoreAudio

- AudioUnit.h
- AUComponent.h
- AudioOutputUnit.h
- AudioUnitParameters.h
- AudioUnitProperties.h
- AudioUnitCarbonView.h
- AUCocoaUIView.h
- MusicDevice.h
- AudioUnitUtilities/AudioToolbox

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

100

CoreAudio HAL

- AudioHardware calls
 - List audio IO devices
 - Get/set device properties (property names are #define'd)
 - AudioDevice structures and API call-back functions
 - Device property listener functions
 - AudioStreams and IO

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

101

Example: Get the Device List

```

UInt32 theSize;
theStatus = AudioHardwareGetPropertyInfo
    (kAudioHardwarePropertyDevices, &theSize, NULL);
theNumberDevices = theSize / sizeof(AudioDeviceID);
theDeviceList = (AudioDeviceID*) malloc
    (theNumberDevices * sizeof(AudioDeviceID));
UInt32 theSize = theNumberDevices * sizeof(AudioDeviceID);
theStatus = AudioHardwareGetProperty
    (kAudioHardwarePropertyDevices,
     &theSize, theDeviceList)

```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

102

CA IO Process Call-back

```
(*AudioDeviceIOProc) (AudioDeviceID inDevice,
    const AudioTimeStamp* inNow,
    const AudioBufferList* inInputData,
    const AudioTimeStamp* inInputTime,
    AudioBufferList* outOutputData,
    const AudioTimeStamp* inOutputTime,
    void* inClientData);
```

Register an IO Process on a device

```
AudioDeviceAddIOProc(AudioDeviceID inDevice,
    AudioDeviceIOProc inProc, void* inClientData);
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

103

Example: JMC's Sine

```
OSStatus appIOProc (AudioDeviceID inDevice, const AudioTimeStamp* inNow,
    const AudioBufferList* inInputData, const AudioTimeStamp* inInputTime,
    AudioBufferList* outOutputData, const AudioTimeStamp* inOutputTime, void*
    appGlobals) {
    appGlobalsPtr globals = appGlobals;
    int i;
    double phase = gAppGlobals.phase;
    double amp = gAppGlobals.amp;
    double pan = gAppGlobals.pan;
    double freq = gAppGlobals.freq * 2. * 3.14159265359
        / globals->deviceFormat.mSampleRate;
    int numSamples = globals->deviceBufferSize
        / globals->deviceFormat.mBytesPerFrame;
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

104

Sine Demo, part 2

```
// assume floats for now....
float *out = outOutputData->mBuffers[0].mData;
for (i = 0; i < numSamples; ++i) {
    float wave = sin (phase) * amp;
    phase = phase + freq;
    *out++ = wave * (1.0-pan);
    *out++ = wave * pan;
}
gAppGlobals.phase = phase;
return (kAudioHardwareNoError);
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

105

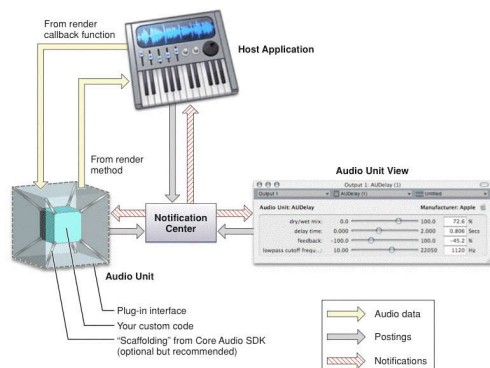
AudioUnits

- Components that have sources and destinations
- AUs can be connected into AUGraphs
- AUs have properties and a property API
- AUs have IO call-backs
- Call-backs and the “pull” model

CSB MAT 240A stephen@mat.ucsb.edu 2008

106

App & Plug-in



CSB MAT 240A stephen@mat.ucsb.edu 2008

107

AU AudioStreams

```
struct AudioStreamBasicDescription {
    Float64 mSampleRate; // the native sample rate
    UInt32 mFormatID; // the specific encoding type
    UInt32 mFormatFlags; // flags specific to each format
    UInt32 mBytesPerPacket; // the number of bytes in a packet
    UInt32 mFramesPerPacket; // the number of frames in packet
    UInt32 mBytesPerFrame; // the number of bytes in a frame
    UInt32 mChannelsPerFrame; // the number of channels per frame
    UInt32 mBitsPerChannel; // the number of bits in each channel
};
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

108

Other MacOSX APIs

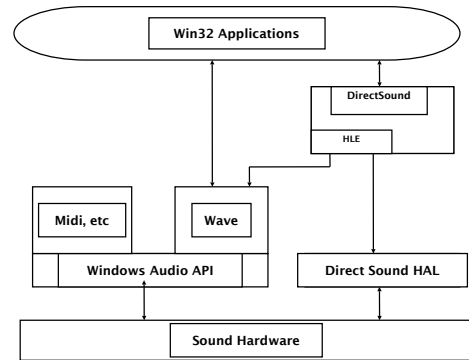
- AudioUnits
- QuickTime
- Quartz
- AppleScript
- Cocoa (NeXTSTEP)
- CoreAnimation, CoreVideo
- OpenGL

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

109

Windows & DirectX



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

110

MS DirectX, DirectSound

- Merge of DirectMusic (+ Producer, Composer), DirectSound, Direct3D
- Downloadable Sounds (DLS)
- DirectX Media Objects, Direct3D
- File and streaming API
- DSP pipe-line and plug-ins
- DirectSound buffers, synthesizers
- Segments and containers
- MS Windows 98.2, 2000, NT, XP, Vista

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

111

MS MM APIs

- DirectDraw (2D graphics)
- Direct3D (3D)
- DirectSound/3D (sampled sound IO)
- DirectPlay (old) UDP streaming
- DirectInput/XInput/WMInput
- DirectShow (MM playback & plug-ins)

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

112

DirectSound Objects

- IDirectSound, IDirectSoundBuffer, IDirectSound3DBuffer, and IDirectSound3DListener.
- IDirectSoundCapture and IDirectSoundCaptureBuffer COM
- IKSPropertySet COM
- IDirectSoundNotify

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

113

DirectSound Examples: PlaySound.cpp

Functions & Globals

```

INT_PTR CALLBACK MainDlgProc( HWND hDlg, UINT msg,
    WPARAM wParam, LPARAM lParam );
VOID OnInitDialog( HWND hDlg );
VOID OnOpenSoundFile( HWND hDlg );
HRESULT OnPlaySound( HWND hDlg );
HRESULT PlayBuffer( BOOL bLooped );
VOID OnTimer( HWND hDlg );
VOID EnablePlayUI( HWND hDlg, BOOL bEnable );

CSoundManager* g_pSoundManager = NULL;
CSound* g_pSound = NULL;
BOOL g_bBufferPaused;
  
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

114

PlaySound.cpp: OnOpenSoundFile()

```
// Setup the OPENFILENAME structure
OPENFILENAME ofn = { sizeof(OPENFILENAME), hDlg, NULL,
    TEXT("Wave Files\\0*.wav\\All Files\\0*.\\0"), NULL,
    0, 1, strFileName, MAX_PATH, NULL, 0, strPath,
    TEXT("Open Sound File"),
    OFN_FILEMUSTEXIST|OFN_HIDEREADONLY, 0, 0,
    TEXT(".wav"), 0, NULL, NULL }; // struct init
...
// Load the wave file into a DirectSound buffer
if( FAILED( hr = g_pSoundManager->Create( &g_pSound, strFileName,
    0, GUID_NULL ) ) )
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

115

DirectSound Examples

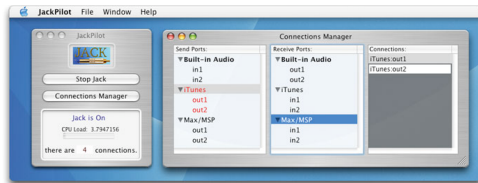
- 3D sound
- soundFX
- Desires and control
- Producer, Composer

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

116

Audio on Linux



- OSS, nas on *NIX
- OSS, ALSA, LADSPA
- JACK Kit
- ASIO

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

117

Linux Audio APIs

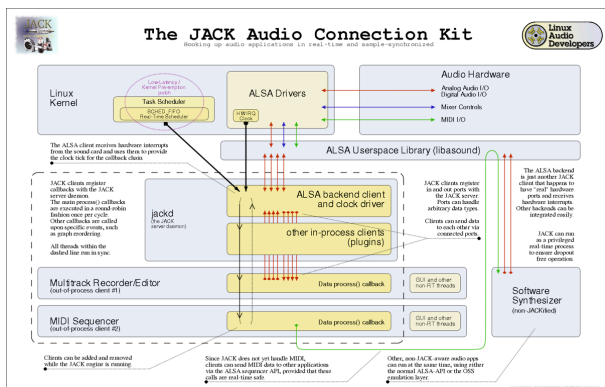
- OSS
 - Special files
 - /dev/audio, /dev/dsp, /dev/midi, /dev/mixer, /dev/synth, ..
 - Open(), close(), read(), write(), ioctl()
 - Ioctl() calls for param setting, reset, queries
- ALSA
 - HAL + client API in C structs
- LADSPA
 - Plug-in architecture, not inter-app
- JACK Connection Kit
 - Server daemon & client API

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

118

JACK: The Big Picture



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

119

JACK Headers

- <jack/jack.h> is the main JACK interface
- <jack/statistics.h> monitor the performance of a server
- <jack/intclient.h> load and unload JACK internal clients
- <jack/ringbuffer.h> lock-free ringbuffers
- <jack/transport.h> simple transport control for clients
- <jack/types.h> main JACK data types
- <jack/thread.h> thread creation for JACK and clients
- <jack/midiport.h> read and write MIDI

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

120

JACK in a Nut-shell

```
jack_client_t * jack_client_open(char * client_name,
jack_options_t options, jack_status_t * status, ... )

int jack_connect(jack_client_t * , char * source_port,
char * destination_port)

jack_port_t * jack_port_register(jack_client_t *
client, char * port_name, char * port_type, unsigned
long flags, unsigned long buffer_size)

int jack_set_process_callback(jack_client_t * client,
JackProcessCallback process_callback, void * arg)

typedef int(* JackProcessCallback)(jack_nframes_t
nframes, void * arg)
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

121

JACK in CSL

```
char * server_name = "CSL_SERVER";
char * client_name = "CSL_CLIENT";
mClient = jack_client_open (client_name, options, &status, server_name);
if (status & JackServerStarted) logMsg("\tJACK server started");
jack_set_process_callback (mClient, JackCallback, 0);
jack_on_shutdown (mClient, jack_shutdown, 0);
mOutput_port = jack_port_register (mClient, "output",
JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0);
ports = jack_get_ports (mClient, JackPortIsPhysicalJackPortIsOutput);
jack_connect (mClient, jack_port_name (mOutput_port), ports[0]);

Then,

int JackCallback (jack_nframes_t nframes, void *arg)...
```

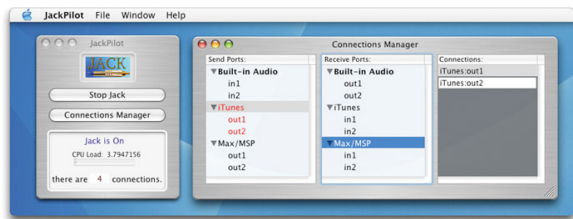
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

122

JACK Daemon

- Start/stop/monitor server
- List/connect clients



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

123

JACK Clients

- Ardour
- Audity
- Score editors
- Sample editors
- Player/streamers
- VOIP
- MP3 encode/decode
- MIDI sync

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

124

Ardour OpenSource DAW



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

125

ALSA SndFile Playback

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

main (int argc, char *argv[]) {
    int i;
    short buf[128];
    snd_pcm_t *playback_handle;
    snd_pcm_hw_params_t *hw_params;
    snd_pcm_open (&playback_handle, argv[1],
        SND_PCM_STREAM_PLAYBACK, 0);
    snd_pcm_hw_params_malloc (&hw_params);
    snd_pcm_hw_params_any (playback_handle, hw_params);
    snd_pcm_hw_params_set_format (playback_handle,
        hw_params, SND_PCM_FORMAT_S16_LE);
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

126

ALSA Playback (2)

```
snd_pcm_hw_params_set_rate_near (playback_handle,
    hw_params, 44100, 0);
snd_pcm_hw_params_set_channels (playback_handle,
    hw_params, 2);
snd_pcm_hw_params (playback_handle, hw_params);
snd_pcm_hw_params_free (hw_params);
snd_pcm_prepare (playback_handle);

for (i = 0; i < 10; ++i)
    snd_pcm_writet (playback_handle, buf, 128);

snd_pcm_close (playback_handle);
}
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

127

Cross-Platform APIs

- PortAudio - C, call-back or blocking
- RTAudio - C++, simple objects, call-back
- SndLib - C, UNIX-style file/stream API
- ASIO - C, HAL & API, fast
- JUCE - C++ models for audio, midi, files, sockets, threads, GUI, GUI builder, ...
- OpenFrameworks - coming soon
- Java APIs

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

128

PortAudio

- Phil Burk (SoftSynth) and Ross Benecia, 2000 (based on open design process)
- Music-dsp mailing list (<http://shoko.calarts.edu/~glmrboy/musicdsp/music-dsp.html>)
- Media_api mailing list (http://create.ucsb.edu/mailman/listinfo/media_api)
- PortAudio:
 - <http://www.portaudio.com/>
 - <http://www.cs.cmu.edu/~music/portmusic/>
- Supports Mac OS-9, -X, and MS Windows now, *NIX underway (...anybody want to help?)
- Relation to PortMIDI effort

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

129

PortAudio

Call-back functions and streams

```
#include "portaudio.h" // this has...
typedef int PaStreamCallback(const void *input, void *output,
    unsigned long frameCount, const PaStreamCallbackTimeInfo* timeInfo,
    PaStreamCallbackFlags statusFlags, void *userData );

// in main()
err = Pa_Initialize();
if( err != paNoError ) goto error;
printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

130

PortAudio Streams

```
err = Pa_OpenDefaultStream( PaStream** stream,
    int numInputChannels,
    int numOutputChannels,
    PaSampleFormat sampleFormat,
    double sampleRate,
    unsigned long framesPerBuffer,
    PaStreamCallback *streamCallback,
    void *userData );

err = Pa_StartStream( stream );
if( err != paNoError ) goto error;
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

131

PortAudio Details

- Stream control
- Device and CPU queries
- Timestamps
- Blocking I/O option
- PortAudio Tutorial

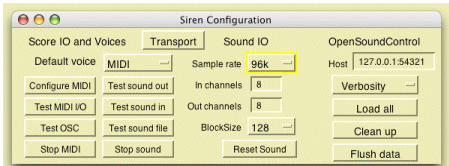
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

132

PortAudio Examples

- PA tools
 - `pa_tests/pa_devs.c` = print a list of devices
 - `pa_tests/pa_minlat.c` = test latency
- PortAudio ring modulator source
- PA transport example: Siren



CSB MAT 240A stephen@mat.ucsb.edu 2008

133

Excerpt from `patest_record.c`

```
static int recordCallback(void *inputBuffer, void *outputBuffer,
    unsigned long framesPerBuffer, PaTimestamp outTime, void *userData) {
    SAMPLE *rptr = (SAMPLE*)inputBuffer;
    for(i=0; i<framesToCalc; i++) {
        *wptr++ = *rptr++;
        *wptr++ = *rptr++;
    }
    // ...
}
```

In `main()`

```
err = Pa_OpenStream(&stream,
    Pa_GetDefaultInputDeviceID(),
    data.samplesPerFrame, /* stereo input */
    PA_SAMPLE_TYPE, NULL, paNoDevice, 0,
    PA_SAMPLE_TYPE, NULL, S_RATE, 1024, // frames / buf
    0, paClipOff, recordCallback, &data );
while(Pa_StreamActive(stream))
    Pa_Sleep(100);
err = Pa_CloseStream( stream );
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

134

Building and Extending PortAudio

- Making the library (see tutorial)
- Making the tests
- Integration with a program (see CO)
- Porting to a new platform (see advanced notes)

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

135

RTAudio

- Object-oriented C++ design
- Simple, common API across all supported platforms
- Only one source and two header files for easy inclusion in programming projects
- Allow simultaneous multi-API support
- Support dynamic connection of devices
- Provide extensive audio device parameter control
- Allow audio device capability probing
- Automatic internal conversion for data format, channel number compensation, (de)interleaving, and byte-swapping

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

136

RTAudio Call-back

```
// Two-channel sawtooth wave generator.

int call_back_fcn(void *outputBuffer, void *inputBuffer,
    unsigned int nBufferFrames, double streamTime,
    RtAudioStreamStatus status, void *userData) {
    double * buffer = (double *) outputBuffer;
    double * lastValues = (double *) userData;

    for (unsigned i = 0; i < nBufferFrames; i++) {
        for (unsigned j = 0; j < 2; j++) {
            *buffer++ = lastValues[j];
            lastValues[j] += 0.005 * (j+1+(j*0.1));
            if (lastValues[j] >= 1.0) lastValues[j] -= 2.0;
        }
    }
    return 0;
}
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

137

in RTAudio `main()`

```
RtAudio dac;
RtAudio::StreamParameters parameters;
parameters.deviceId = dac.getDefaultOutputDevice();
parameters.nChannels = 2;

try {
    dac.openStream( &parameters, NULL, RTAUDIO_FLOAT32,
        sampleRate, &bufferFrames,
        &call_back_fcn, (void *)&user_data );
    dac.startStream();
} ..
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

138

RTAudio Compilation

OS:	Audio API:	C++ Class:	Preprocessor Definition:	Library or Framework:	Example Compiler Statement:
Linux	ALSA	RtApiAlsa	__LINUX_ALSA__	asound, pthread	g++ -Wall -D __LINUX_ALSA__ -o probe probe.cpp RtAudio.cpp -lasound -lpthread
Linux	OSS	RtApiOss	__LINUX_OSS__	pthread	g++ -Wall -D __LINUX_OSS__ -o probe probe.cpp RtAudio.cpp -lpthread
Linux or Macintosh OS-X	Jack Audio Server	RtApiJack	__UNIX_JACK__	jack, pthread	g++ -Wall -D __UNIX_JACK__ -o probe probe.cpp RtAudio.cpp `pkg-config --cflags --libs jack` -lpthread
Macintosh OS-X	CoreAudio	RtApiCore	__MACOSX_CORE__	pthread, CoreAudio	g++ -Wall -D __MACOSX_CORE__ -o probe probe.cpp RtAudio.cpp -framework CoreAudio -lpthread
Windows	Direct Sound	RtApiDs	__WINDOWS_DS__	dsound.lib (ver. 5.0 or higher), multithreaded	compiler specific
Windows	ASIO	RtApiAsio	__WINDOWS_ASIO__	various ASIO header and source files	compiler specific

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

139

Sndlib Introduction

- CCRMA/Stanford, Bill Schottstaedt, 1990-present
- C API for clm and other packages (CREATE auralizer)
- Supports SGI (either audio library), NeXT, Sun, Be, OSS or ALSA (Linux and others), Mac, OS-9, -X, HPUX, LinuxPPC, and MS Windows
- Blocking I/O model

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

140

Sndlib Ports and Functionality

System	SndSine	SndInfo	Audinfo	SndPlay	SndRecord	CLM
NeXT 68k	ok	ok	ok	ok	ok	ok
NeXT Intel	ok	ok	ok	interruptions runs (*)	ok	ok
SGI old and new AL	ok	ok	ok	ok	ok	ok
OSS (Linux et al)	ok	ok	ok	ok	ok	ok
Be	ok	ok	not written	not written	not written	ok
Mac	ok	ok	ok	ok	ok	ok
Windoze	ok	ok	ok	ok	not written	ok
Sun	ok	ok	ok	ok	interruptions ok	ok
HPUX	untested	untested	untested	untested	untested	untried
LinuxPPC	ok	ok	ok	ok	untested (**)	ok
ALSA	ok	ok	ok	ok	ok	ok
Mac OS-X	ok	ok	not written	not written	not written	ok

(*) I can't find a microphone.

(**) Last I looked, recording was still not supported in this OS.

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

141

Sndlib Source Files

- io.c (read and write sound file data)
- headers.c (read and write sound file headers)
- audio.c (read and write sound hardware ports)
- sound.c (provide slightly higher level access to the preceding files)
- sndlib.h (header for the preceding files)
- sndlib2scm.c and sndlib-strings.h (tie preceding into Guile)
- clm.c and clm.h (Music V implementation)
- clm2scm.c, vct.c and vct.h (tie clm.c into Guile)
- old-sndlib.h (old names)

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

142

Sndlib Sound Access Functions

```

int mus_sound_samples(const char *arg) /* samples according to header */
int mus_sound_frames(const char *arg) /* samples per channel */
float mus_sound_duration(const char *arg)
int mus_sound_datum_size(const char *arg) /* bytes per sample */
int mus_sound_data_location(const char *arg) /* location of first sample (bytes) */
int mus_sound_chans(const char *arg) /* number of channels (interleaved) */
int mus_sound_srate(const char *arg) /* sampling rate */
int mus_sound_header_type(const char *arg) /* header type (aiff etc) */
int mus_sound_data_format(const char *arg) /* data format (alaw etc) */
char *mus_sound_comment(const char *arg) /* comment if any */
int mus_sound_length(const char *arg) /* true file length (for error checks) */
int mus_sound_distributed(const char *arg) /* is header scattered in sound file */
int mus_sound_write_date(const char *arg) /* uninterpreted file write date */
int mus_sound_type_specifier(const char *arg) /* original header type identifier */
int mus_sound_align(const char *arg) /* more compression data */
int mus_sound_bits_per_sample(const char *arg) /* bits per sample */
int mus_data_format_to_bytes_per_sample(int format) /* bytes per sample */

```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

143

Sndlib Utility Functions

```

int mus_sound_max_amp(const char *arg, int *vals) /* list of max sample pairs */
int mus_sound_aiff_p(const char *arg) /* old-style AIFF file (not AIFC) */
int *mus_sound_loop_info(const char *arg) /* 6 loop vals (mode,start,end, detune, base-note) */
int mus_sound_initialize(void) /* initialize everything */

char *mus_header_type_name(int type) /* "AIFF" etc */
char *mus_data_format_name(int format) /* "16-bit big endian linear" etc */

mus_error_handler_t *mus_error_set_handler(mus_error_handler_t *handler);
mus_print_handler_t *mus_print_set_handler(mus_print_handler_t *handler);

char *mus_error_to_string(int err);

```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

144

Sndlib Data I/O Functions

- `int mus_sound_open_input (const char *arg)`
- `int mus_sound_open_output (const char *arg, int srates, int chans,`
`int data_format, int header_type, const char *comment)`
- `int mus_sound_reopen_output (const char *arg, int type,`
`int format, int data_loc)`
- `int mus_sound_close_input (int fd)`
- `int mus_sound_close_output (int fd, int bytes_of_data)`
- `int mus_sound_read (int fd, int beg, int end, int chans,`
`MUS_SAMPLE_TYPE **bufs)`
- `int mus_sound_write (int fd, int beg, int end, int chans,`
`MUS_SAMPLE_TYPE **bufs)`
- `int mus_sound_seek (int fd, long offset, int origin)`
- `int mus_sound_seek_frame (int fd, int frame)`

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

145

Sndlib Hardware, Systems, Devices

- `int mus_audio_initialize(void)`
- `void mus_audio_save/restore(void)`
- `void mus_audio_describe(void)`
- `char *mus_audio_report(void)`
- `int mus_audio_open_output/input(int dev, int srates, int chans,`
`int format, int size)`
- `int mus_audio_close(int line)`
- `int mus_audio_read/write(int line, char *buf, int bytes)`
- `int mus_audio_mixer_read/write(int dev, int field, int chan, float *val)`
- `int mus_audio_systems(void)`
- `char *mus_audio_system_name(int system)`
- `void mus_audio_set_dsp_devices(int cards, int *dsps, int *mixers)`

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

146

Sndlib Examples

- `sndinfo.c`
- `sndplay.c`
- `sndrecord.c`
- CLM and Guile interfaces
- Sndlib and the CREATE auralizer
- Building sndlib

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

147

Excerpt from sndinfo.c

```
mus_sound_initialize();
if (mus_file_probe(argv[1])) {
    date = mus_sound_write_date(argv[1]);
    srates = mus_sound_srates(argv[1]);
    chans = mus_sound_chans(argv[1]);
    samples = mus_sound_samples(argv[1]);
    comment = mus_sound_comment(argv[1]);
    length = (float)samples / (float)(chans * srates);
    loops = mus_sound_loop_info(argv[1]);
    type = mus_sound_header_type(argv[1]);
    format = mus_sound_data_format(argv[1]);
}
fprintf(stdout, "%s\n srates: %d\n chans: %d\n length: %f\n",
        argv[1], srates, chans, length);
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

148

Auralizer audio_io.c

```
void setup_DAC() {
    if (mus_sound_initialize() < 0)
        while("Error: unable to initialize sndlib", 1);
    audio_port = mus_audio_open_output(MUS_AUDIO_DEFAULT,
        sample_rate, 2, MUS_COMPATIBLE_FORMAT, 0);
    ...
}
void output_loop() {
    while (1) {
        // write sample buffer to the DAC
        // This will block until the write's done.
        mus_audio_write(audio_port,
            (char *) (xf_bufptr[xf_bufnum]), buf_len);
        ...
    }
}
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

149

Compiling sndplay on Linux

- `cc -c io.c -O -DLINUX`
- `cc -c audio.c -O -DLINUX`
- `cc -c headers.c -O -DLINUX`
- `cc -c sound.c -O -DLINUX`
- `cc sndplay.c -o sndplay -O -DLINUX audio.o io.o headers.o sound.o -lm`
- Or
- `ld -r audio.o io.o headers.o sound.o -o sndlib.a`
- `cc sndplay.c -o sndplay -O -DLINUX sndlib.a -lm`

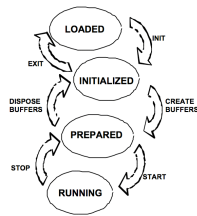
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

150

ASIO

- Steinberg cross-platform
 - *Very fast, light-weight, and reliable*
- Comprehensive straight-C API with object life-cycle
- See header file (only)
- Relation to VST



CSB MAT 240A stephen@mat.ucsb.edu 2008

151

ASIO Functions

```

ASIOError ASIOInit(ASIODriverInfo *info);
ASIOError ASIOExit(void);
ASIOError ASIOStart(void);
ASIOError ASIOStop(void);
ASIOError ASIOGetChannels(long *numInputChannels, long
*numOutputChannels);
ASIOError ASIOGetLatencies(long *inputLatency, long *outputLatency);
ASIOError ASIOGetBufferSize(long *minSize, long *maxSize, long
*preferredSize, long *granularity);
ASIOError ASIOCanSampleRate(ASIOSampleRate sampleRate);
ASIOError ASIOGetSampleRate(ASIOSampleRate *currentRate);
ASIOError ASIOSetSampleRate(ASIOSampleRate sampleRate);
  
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

152

More ASIO Functions

```

ASIOError ASIOGetClockSources(ASIOClockSource *clocks, long *numSources);
ASIOError ASIOSetClockSource(long reference);
ASIOError ASIOGetSamplePosition (ASIOSamples *sPos, ASIOTimeStamp
*tStamp);
ASIOError ASIOGetChannelInfo(ASIOChannelInfo *info);
ASIOError ASIOCreateBuffers(ASIOBufferInfo *bufferInfos, long numChannels,
long bufferSize, ASIOCallbacks *callbacks);
ASIOError ASIODisposeBuffers(void);
ASIOError ASIOControlPanel(void);
void *ASIOFuture(long selector, void *params);
ASIOError ASIOOutputReady(void);
  
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

153

ASIO Structures

- ASIOTimeStamp, ASIOSample, ASIOBoolean, ASIOSampleRate
- ASIOTimeInfo
- ASIOCallbacks
- ASIODriverInfo
- See hostsample.cpp

CSB MAT 240A stephen@mat.ucsb.edu 2008

154

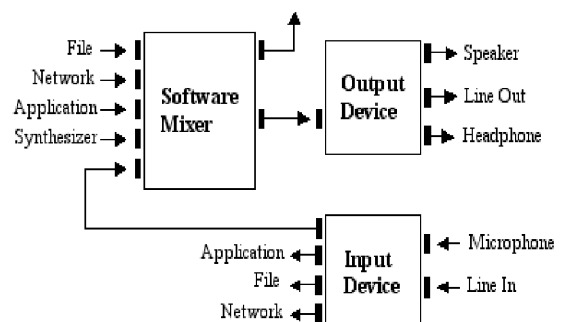
JavaSound

- Low-level object-oriented API for sound and MIDI (Java Media Framework (JMF) is higher-level API)
- Intended to support many capabilities and application domains
- See API object javadoc file
- See Java presentation slides

CSB MAT 240A stephen@mat.ucsb.edu 2008

155

JavaSound Objects



CSB MAT 240A stephen@mat.ucsb.edu 2008

156

JavaSound Packages

- Central packages
 - javax.sound.sampled
 - javax.sound.midi
- Service provider interfaces
 - javax.sound.sampled.spi
 - javax.sound.midi.spi

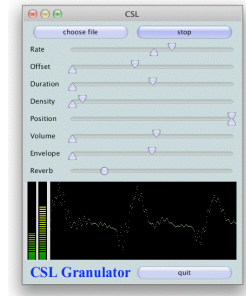
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

157

Other Audio APIs

- JUCE
- Siren
- CSL I/Os
- OpenAL
- OtherJava
- OpenFrameworks
- Kyma



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

158

Review

- APIs for audio IO
 - Platform-specific
 - Cross-platform
 - Language-specific
 - So alike, so different...

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

159

Coding Exercises

- Blocking IO vs call-back model
- C structs vs C++ stream/port/device objects
- Platform-bound APIs
- Cross-platform APIs
 - PortAudio, RTAudio
- Applications
 - MAT240D synths
 - CSL, STK
 - VOIP client

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

160

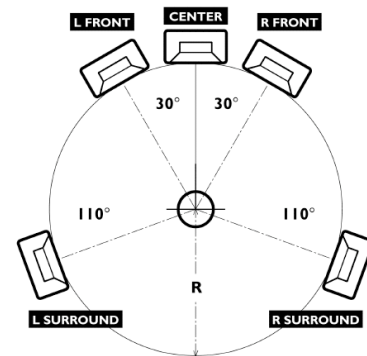
What's Next

- Topic 2 Readings
 - A Child's Garden of Sound File Formats
 - LibSndFile API doc.
 - New Applications of the SDIF
 - Streaming Stored Audio and Video
 - SHOUTCAST Streaming API
 - Code examples: lsf -- list sound files, EBICSF
 - Other references & links

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

161



Topic 2: Storage & Streaming

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

162

Topic 2: Sound Storage & Streaming Formats

- Sound Storage Formats
- Sound file formats intro and examples
- Real-time I/O and Internet streaming

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

163

Topic 2 Readings

- A Child's Garden of Sound File Formats
- LibSndFile API doc.
- New Applications of the SDIF
- Streaming Stored Audio and Video
- SHOUTCAST Streaming API
- Code examples: lsf -- list sound files, EBICSF
- Other references & links

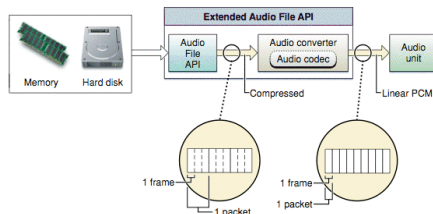
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

164

SndFile Storage APIs

Figure 3-1 Reading audio data



- File formats
- Simple, multi-part

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

165

The Sound File

- Header vs. header-less
 - Type implicit in name
 - xxx.au, xxx.snd
 - Format described in file header or resource fork
 - Fixed parameters
- Fixed-format vs. “chunked” formats
 - Description and data chunks

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

166

The Sound File Header

- “Magic number” (keyword or file ID, 32-bit, identifies file)
- Basic parameters
 - Sampling rate
 - Format/resolution
 - Number of channels
- Sample data block (typically 1)
- Other data
 - Comment, text
 - Pitch
 - MIDI data
 - Loop points

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

167

The NeXT/Sun File Header (1986)

```
typedef struct {
    int magic;          /* magic number SND_MAGIC */
    int dataLocation;    /* offset/pointer to the data */
                        /* (header can be >28 bytes) */
    int dataSize;        /* number of bytes of data */
    int dataFormat;      /* the data format code */
    int samplingRate;    /* the sampling rate */
    int channelCount;    /* the number of channels */
    char info[4];        /* optional text information */
} SndSoundStruct;
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

168

Extended Fixed Headers

- MOD, AVR, SndDesigner
 - Sample Loop points (begin/end)
 - Pitch, MIDI key
 - MIDI keyboard split, multiple samples
- IRCAM/BICSF, EBICSF
 - MaxAmp, ampl. envelopes
 - Pitch envelopes, LPC
 - Processing scripts
 - Cues, virtual files

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

169

Tagged Formats: AIFF/WAV

- Chunked file formats: simple, ubiquitous
- Chunk = ID, Size, Data

```
struct ChunkHeader {
    int32 ckID;      // char[4]
    int32 ckSize;    // size in bytes
};
```

- File: FORM, COMM, DATA, etc.

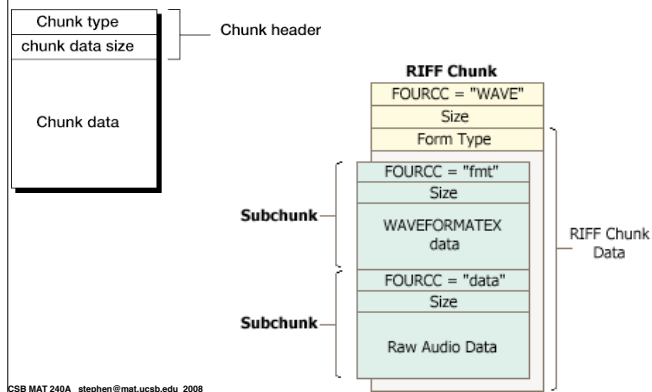
```
struct SoundDataChunk {
    int32 ckID;
    int32 ckSize;
    unsigned int32 offset;
    unsigned int32 blockSize;
};
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

170

AIFF/WAV Chunk & File



CSB MAT 240A stephen@mat.ucsb.edu 2008

171

Sound Data Processing

- CARL - UNIX shell pipes
 - sndin/out
 - Filter, gain, reverb,
 - par/chan
- SoundHack
 - Mac format conversion
 - Analysis/synthesis
- SOX
 - Portable UNIX conversion, transformation

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

172

Sound Formats and Bandwidths

- Surround HiFi (8 @ 24/96) 18 M b/s
- “CD-quality” (2 @ 16/44) ~ 1.4 M b/s
 - Losses (rel. to 24/96) easily heard over good systems
- MP3 (DCT, masking) ~128 k b/s (VBR)
 - Debatable losses relative to “CD-quality” over “mid-fi” systems for ≥ 128 k b/s bandwidth
 - Obvious losses for lower bit-rates
- RealAudio (CELP) ~ 14 k b/s
 - Monophonic, “telephone-quality”
- Best speech compression ~ 1.5 k b/s

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

173

Sound File IO APIs: libSndFile

- UNIX-like file API in C
 - Struct SNDFILE* as opaque ptr
 - Struct sfinfo has members from file header
 - s_rate, #chans, #frames, format...
- Typed, reformatting r/w calls (e.g., *sf_write_short*)
- Easy to integrate with your own “sound” object
- Examples

```
SNDFILE * sf_open(char *path, int mode, SF_INFO *sfinfo);
int sf_read_float(SNDFILE *sndfile,
                 float *ptr, sf_count_t items);
int sf_seek(SNDFILE *sndfile,
            sf_count_t frames, int whence);
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

174

libSndFile Formats

	Micro- soft WAV	SGI / AIFF / AU / AIFC	Sun / IRIX / NAX / AU / SND	Header- less RAW	Pulse Audio File PAF	Commo- dore Amiga IFF / SVX	Sphere NW WAV	HCAM SF	Creative VOC	Sound Forge W64	GNU Octave 2.0 MAT4	GNU Octave 2.1 MAT5	Portable Voice Format PVT	FastTracker 2 MTX	IBM Toot Kit ITK	Apple CAF
Unsigned 8 bit PCM	R/W	R/W		R/W					R/W	R/W		R/W				
Signed 8 bit PCM		R/W	R/W	R/W	R/W	R/W	R/W						R/W			
Signed 16 bit PCM	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W
Signed 24 bit PCM	R/W	R/W	R/W	R/W	R/W		R/W	R/W								R/W
Signed 32 bit PCM	R/W	R/W	R/W	R/W			R/W	R/W			R/W	R/W	R/W	R/W		R/W
32 bit float	R/W	R/W	R/W	R/W			R/W			R/W	R/W	R/W				R/W
64 bit double	R/W	R/W	R/W	R/W						R/W	R/W	R/W				R/W
u-law encoding	R/W	R/W	R/W	R/W			R/W	R/W	R/W	R/W	R/W					R/W
A-law encoding	R/W	R/W	R/W	R/W			R/W	R/W	R/W	R/W						R/W
DMA ADPCM	R/W															
MS ADPCM	R/W									R/W						
GSX E-10	R/W	R/W		R/W												
G721 ADPCM 16kps	R/W															
G722 ADPCM 24kps				R/W												
G723 ADPCM 16kps				R/W												
12 bit DVIW		R/W		R/W												
16 bit DVIW		R/W		R/W												
24 bit DVIW		R/W		R/W												
Q8 Dialogic ADPCM				R/W												
8 bit DPCM															R/W	
16 bit DPCM															R/W	

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

175

LibSNDFile Coding

```
// built-in types
SNDFILE *file ;
SF_INFO sfinfo ;

// header struct members
sfinfo.format = filetype | SF_FORMAT_PCM_16 ;
sfinfo.samplerate = 48000 ;
sfinfo.channels = 2 ;

// UNIX file model
file = sf_open (filename, SFM_READ, &sfinfo_rd);

// typed read/write fcn's.
k = sf_write_short (file, data, this_write);
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

176

Other Stream IO APIs

- MP3 decoders
 - See below
- OpenAL
- XMLAudio
- Java Media Framework
- Siren
- SDIF
- QT server

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

177

SDIF: Sound Description
Interchange Format

- See <http://cnmat.CNMAT.Berkeley.EDU/SDIF>
- Data is a sequence of frames (like AIFF/RIFF chunks) organized into streams
- All data is in matrices of fixed size
- The data in a frame are 2D (or 1D or oD) matrices of floating-point numbers, with each column corresponding to a parameter like frequency or amplitude and each row representing an object like a filter, sinusoid, or noise band.
- Each frame type defines its format

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

178

SDIF Frames and Matrices

- A frame consists of the following:
 - A frame type ID
 - A count of the size, in bytes, of the frame
 - A time tag
 - A stream ID
 - The number of matrices in the frame
 - The matrices themselves
- A matrix consists of the following:
 - A matrix type ID
 - A code for the type of the data in the matrix
 - The numbers of rows and columns in the matrix
 - The data themselves
 - Optional byte padding to make the overall matrix size a multiple of 64 bits

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

179

SDIF Frame Types

- Frame Types to be Standardized
 - Spectral envelopes (sampled and parametric)
 - Cepstral coefficients
 - LPC coefficients
 - Formants
 - Wavelets
 - Diphones
 - "Note lists"

Frame Type ID	Frame Type	Columns of Main Matrix
1REQ	Fundamental Frequency Estimates	Fundamental frequency, confidence
1STE	Discrete Short-Term Fourier Transform	Real & imaginary bin values
1PIC	Picked Spectral Peaks	Freq, amp, phase, confidence
1TEC	Sinusoidal Tracks	Index, freq, amp, phase
1HEM	Pseudo-harmonic Sinusoidal Tracks	Harmonic partial #, freq, amp, phase
1RES	Resonances / Exponentially Decaying Sinusoids	Freq, amp, decay rate, phase
1TDS	Time Domain Samples	Channels of sample data

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

180

Programming with SDIF

- SDIF Library
- 2000 ICMC Analysis/Synthesis Session
 - SNDAN
 - SMS
 - Loris
 - MDRx
 - IRCAM Suite
- See <http://cnmat.CNMAT.Berkeley.EDU/SDIF/ICMC2000>
- Examples: SDIF-text convertor, Loris I/O

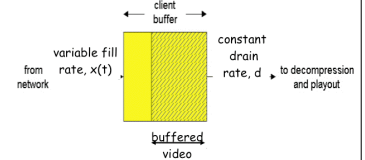
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

181

Content Streaming

- Categories of application
 - 1:1 or 1:n (one source)
 - Streaming of stored content (podcasts, YouTube)
 - Streaming of live content (webcasting)
 - 1:1 or n:m (multi-source)
 - 2-way interactive streaming (Skype, iChat)
- Types of errors
 - Delay
 - Jitter
 - Packet loss



CSB MAT 240A stephen@mat.ucsb.edu 2008

182

Sound Streaming Issues

- Database server (disk I/O)
- Network server (100BaseT, ATM)
- Client-side player (lightweight, plug-in)
 - Network interface
 - Buffer
 - I/O driver
- Network considerations
 - Quality-of-service (QoS)
 - Rate negotiation
 - Distributed buffers (multicast)

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

183

Streaming Support

- Characterize the error sources
 - Jitter, drop-outs
- FIFO buffers to cache signals
 - OS-level, stream level
- Handling jitter: add buffers
- Dynamic QoS negotiation
 - Move/resize buffers, change bit-rate
- Distributed aggregation (multi-cast hubs)
- Streaming, monitoring and control protocols

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

184

Streaming Examples

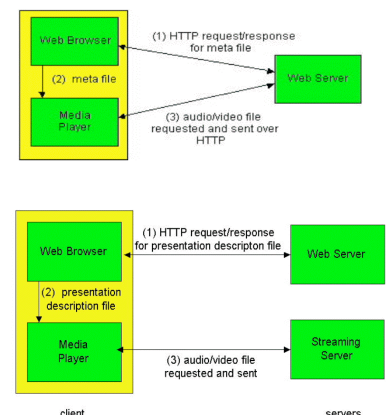
- RealAudio
 - Variable QoS, bandwidth (5-96 kbps, lo-fi)
 - Simple (proprietary) client
 - Mono-directional 1:n, no distributed buffers
- Network Audio System (NAS)
 - Fixed QoS (CD/au)
 - Symmetrical client/server
 - Bi-directional n:m, distributed buffers
- Others: QT, Liquid, MS, SMIL, ...

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

185

Browser/Player Streaming

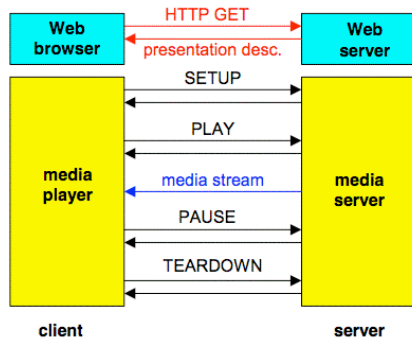


CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

186

The Details



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

187

Streaming References

- WWW streaming
- RTP/RTSP/RTCP (see MAT201B)
- QT server
- Streaming to mobile/embedded devices
- SDIF & OSC streaming
- CSL RFS

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

188

Review

- Sound file formats and audio data/metadata
- Special-purpose IO APIs for files
 - libSndFile
 - SDIF
- Streaming protocols and formats

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

189

Coding Examples

- Using libSndFile in a wrapper object for sound or sound stream
- LibSndFile utilities: lsf, play/record, gui
- Audacity etc. programming
- Using SDIF for signal data
- Streaming examples

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

190

What's next?

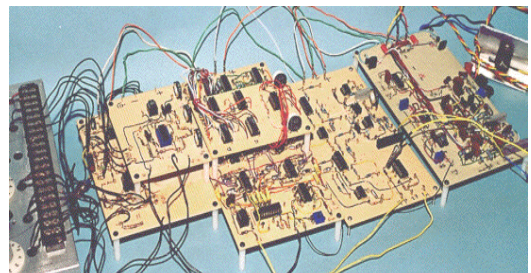
- Topic 3 readings
 - Tutorial & Overview of MPEG Audio
 - AC-3: Flexible Coding for Audio Transmission and Storage
 - Design and Implementation of AC-3 Coders

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

191

Topic 3: Compression



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

192

Topic 3: Compression formats and streaming

- Compression formats, codecs, and players
- Future formats
- Real-time I/O and Internet streaming
- Control protocols

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

193

Topic 3 Readings

- Overview of MPEG Audio, K. Brandenburg and M. Bosi, Dolbylabs JAES 1997.
- AC-3: Flexible Coding for Audio Transmission and Storage, Dolbylabs
- Design and Implementation of AC-3 Coders, Steve Vernon, Dolbylabs. QuickTime MP-3 Player
- SHOUTCAST Streaming API
- MP3.com streaming API

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

194

See also

- Web Links
 - MP3 & AAC technology
 - Listening tests and comparisons
 - Open-source codecs
- EBU Listening tests on Internet Audio Codecs
- RealAudio vs. WMA tests

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

195

Simple Compression Techniques

- Run-length encoding
- Dictionary-based loss-less compression (Huffman)
- Compression of multi-channel sound (sum/difference)
- LPC-based and FFT-based compression

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

196

MPEG & MP3

- MPEG Group & FHG IIS
- MPEG Standard Versions (past and future)
 - See table in Brandenburg & Bosi
 - Standard (and patent) describe representation format, not codec algorithm
- MPEG 2 Audio Layers and Bandwidth
 - 2.1, 2.2, 2.3 = MP3
- Basic Techniques
 - Time-frequency representation
 - Temporal and spectral masking
 - Joint stereo representation
 - Numerical compression
 - Asymmetrical codec

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

197

MP3 Quality vs Bit-rate

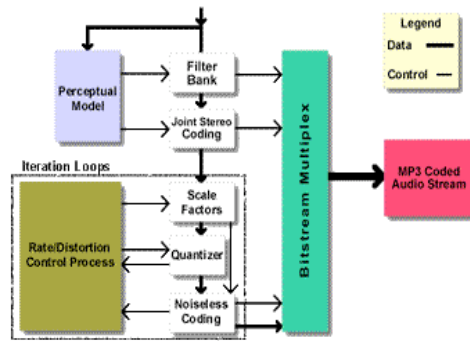
sound quality	bandwidth	mode	bitrate	reduction ratio
telephone sound	2.5 kHz	mono	8 kbps *	96:1
better than short-wave	4.5 kHz	mono	16 kbps	48:1
better than AM radio	7.5 kHz	mono	32 kbps	24:1
similar to FM radio	11 kHz	stereo	56...64 kbps	26...24:1
near-CD	15 kHz	stereo	96 kbps	16:1
CD	>15 kHz	stereo	112...128kbps	14...12:1

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

198

MP3 Encoder Internals

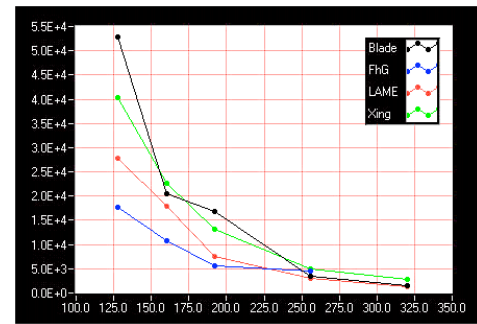


CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

199

MP3 Encoder Error-rates

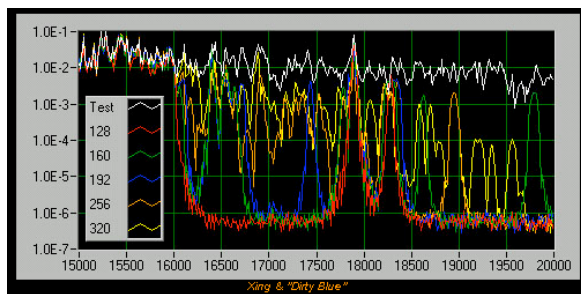


CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

200

Comparing Bitrates



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

201

Using MP3 Encoders

- Command-line (e.g., Linux)
- Drag&drop encoders (Mac, MS-Win)
- Auto-rippers
- Configuration options
 - Bit-rate and policy
 - VBR quality
 - Stereo merge
 - Speed/quality knob
 - Info for ID3 header

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

202

Blade Encoder Options

Usage: bladeenc [global switches] input1 [output1 [switches]] input2 ...

- [kbit], -br [kbit] Set MP3 bitrate. Default is 128 (64 for mono output).
- crc Include checksum data in MP3 file.
- delete, -del Delete sample after successful encoding.
- private, -p Set the private-flag in the output file.
- copyright, -c Set the copyright-flag in the output file.
- copy Clears the original-flag in the output file.
- mono, -dm Produce mono MP3 files by combining stereo channels.
- leftmono, -lm Produce mono MP3 files from left stereo channel only.
- swap Swap left and right stereo channels.
- rawfreq=[freq] Specify frequency for RAW samples. Default 44100.
- rawbits=[bits] Specify bits per channel for RAW samples. Default 16.
- rawmono Specifies that RAW samples are in mono, not stereo.
- rawstereo Specifies that RAW samples are in stereo (default).

Input/output files can be replaced with STDIN and STDOUT respectively.

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

203

Lame Encoder Options

USAGE : lame [options] <infile> [outfile]

<infile> and/or <outfile> can be "-", which means stdin/stdout.

OPTIONS :

- m mode (s)tereo, (j)oint, (f)orce or (m)ono (default j)
force = force ms_stereo on all frames. Faster and uses special Mid & Side masking thresholds
- b <bitrate> set the bitrate, default 128kbps
(for VBR, this sets the allowed minimum bitrate)
- s sfreq sampling frequency of input file(kHz) - default 44.1
- resample sfreq sampling frequency of output file(kHz)- default=input sfreq
- mp3input input file is a MP3 file
- v use variable bitrate (VBR)
- V n quality setting for VBR. default n=4
0=high quality, bigger files. 9=smaller files
- t disable Xing VBR informational tag
- nohist disable VBR histogram display

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

204

Lame Options 2

- h use (maybe) quality improvements
- f fast mode (low quality)
- k disable sfb=21 cutoff
- d allow channels to have different blocktypes

Specifying any of the following options will add an ID3 tag

- tt <title> title of song (max 30 chars)
- ta <artist> artist who did the song (max 30 chars)
- tl <album> album where it came from (max 30 chars)
- ty <year> year in which the song/album was made (max 4 chars)
- tc <comment> additional info (max 30 chars)

MPEG1 samplerates(kHz): 32 44.1 48

bitrates(kbs): 32 48 56 64 80 96 112 128 160 192 224 256 320

MPEG2 samplerates(kHz): 16 22.05 24

bitrates(kbs): 8 16 24 32 40 48 56 64 80 96 112 128 144 160

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

205

MP3 Encoder Performance

Test encoding the same file (Linux HOW-TO)

BladeEnc 0.91 (c) Tord Jansson

Completed. Encoding time: 00:05:58 (0.78X)

LAME version 3.50 (www.sulaco.org/mp3)

GPSYCHO: GPL psycho-acoustic model version 0.74.

Frame | CPU/estimated | time/estimated | play/CPU | ETA
10756/ 10756(100%) | 0:02:28/ 0:02:28 | 0:02:29/ 0:02:29 | 1.9074 | 0:00:00

GOGO-no-coda ver. 2.24 (Feb 12 2000)

{ 10751/ 10753 } 100.0% (2.94x) re:[00:00:00.03] to:[00:01:35.42]

End of encoding -- time= 95.430sec

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

206

AAC & AC-3

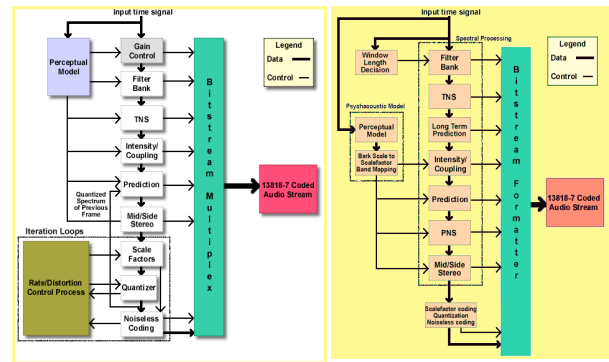
- AAC = MP4 (not backward-compatible)
- 128 kbps and up (DVD = 384 kbps, theaters 640 kbps)
- Support up to 48 channels and up to 96 kHz Fs
- More efficient encoder than MP3 (RT/HQ/Eff??), HW support available

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

207

AAC Encoder Structure



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

208

AAC vs MP3 Technology

	PsyTEL AAC	MPEG Layer III (MP3)
Maximum Channels	48 Main + 16 LFE Channels, 64 channels total	2 (Stereo)
Max. bit-rate per channel	512 kbit/s (at 96 kHz)	100 kbit/s
Sampling Rates Supported	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48, 64, 88.1 and 96 kHz	32, 44.1, 48 kHz (MPEG-1) and 16, 22.05 and 24 kHz (MPEG-2)
Frame Prediction	Yes (Long Term Predictor)	No
Stereo @128 kbit/s	Very High Quality	Medium Quality
VBR Support	Native	Optional
Joint Stereo Coding	Fine (per frequency band)	Rough (per frame)
Additional Tools	Yes (TNS coding and PNS)	No
Spectral Resolution @48 kHz (smaller is better)	23.4 Hz	41.6 Hz
Filterbank	MDCT	Hybrid Filterbank
Pre-Echo Control	Excellent	Weak
Impulse Response @48 kHz	5.3 ms	18.0 ms (pre-echo)
Further Development	Yes (MPEG-4 V2)	No

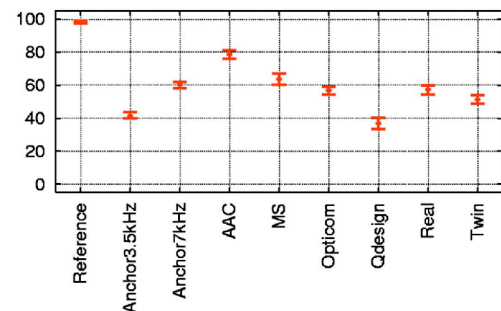
CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

209

Comparing Formats 1 (EBU)

2e) Site:DR+NRK -- all items -- Bitrate:64kbps_Stereo

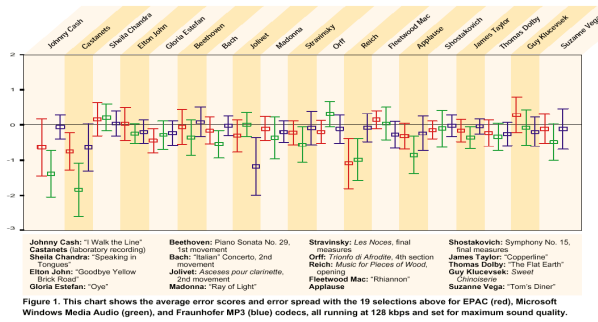


CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

210

EPAC, WMA, MP3 (Stereo Review)



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

211

RealAudio vs WMA (NSTL)

NSTL obtained the following results during the testing. The values in each column represent the number of participants, out of 100, that felt the clip played in that format sounded most like the original music clip.

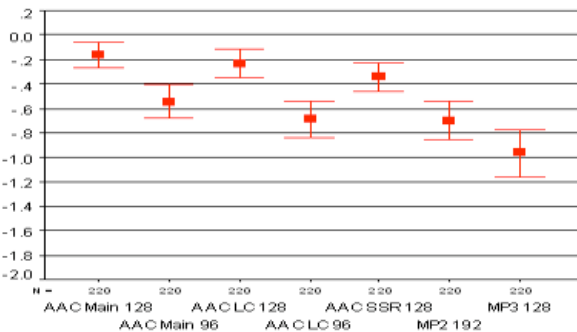
Song/Format	Windows Media Audio	RealAudio 8
La Grange, 20kbs	16	84
La Grange, 32kbs	27	73
The Obvious Child, 20kbs	38	62
The Obvious Child, 32kbs	42	58
Totals	123	277

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

212

AAC vs MP3 Quality



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

213

Future Codecs

- High-end
 - MPEG-7, MPEG-21
 - AAC/AC-3 in the future
 - MLP and DVD-audio
- Mid-range
 - Better MP3
 - Next generation of WMA, QT, TWIN, etc.

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

214

MP3 Decoder APIs

- Parsing/expanding an MP3 stream
- Decoder call-back APIs
- Example: libMad

```
mad_flow output(void *data,
    struct mad_header *header,
    struct mad_pcm *pcm);
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

215

libMad API Calls

```
// libMad data structs
struct buffer buffer;
struct mad_decoder decoder;

// init fcn registers all our (6) call-backs
// (several of which might be NULL)
mad_decoder_init(&decoder, &buffer,
    input_fcn, header_fcn, filter_fcn,
    output_fcn, error_fcn, message_fcn);

result = mad_decoder_run(&decoder, MAD_DECODER_MODE_SYNC);
// wait...
mad_decoder_finish(&decoder);
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

216

Using libMAD

- `input_fcn` - set-up
- `header_fcn` - parsed file header
- `filter_fcn` - tweak filter coeffs
- `output_fcn` - process a buffer of output
- `error_fcn` - handle decoder errors
- `message_fcn` - progress function callback

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

217

Streaming Servers for MP3

- Server, Connection, and Client
- Network Server Background
 - Configuration
 - Start/stop
 - Logging/debugging/accounting
- Streaming Connections
 - Bandwidth considerations
- MP3 Player Clients (can read disk or stream)
 - `mpg123 http://ip_address:8000`
 - `streamripper -h megajukebox -p 8000`

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

218

MP3 Servers

- Icecast: streaming server
 - Options: port, config file, password, # clients, foreground, logging, home dir.
- Shout: content server for Icecast
 - Playlist, bitrates, aux. Info
- Apache MP3 Module
 - Say: `perldoc Apache::MP3` or see <http://www.perldoc.com/cpan/Apache/MP3.html>
 - # `httpd.conf` or `srm.conf`
 - `AddType audio/mpeg mp3 MP3`
 - # `httpd.conf` or `access.conf`
 - `<Location /songs>`
 - `SetHandler perl-script`
 - `PerlHandler Apache::MP3::Sorted`
 - `PerlSetVar SortField Title`
 - `PerlSetVar Fields Title,Artist,Album,Duration`
 - `</Location>`

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

219

Using Codecs & Streamers

- File- or buffer-oriented MP3 reading, e.g., Mac/Carbon MP3 player
- FMAK MP3 decoder API
- Other examples?
- Shoutcast server config
- Higher-level services: MP3.com protocol & API

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

220

MP3 Decoder Class

```
class MP3_API M3Decode {
public:
    M3Decode();
    ~M3Decode();
    bool DecodeFile(const char *infile, SongObj * song,
                   bool merge, bool decimate);
    bool Setup(MPEG_Args *m3args);
    void PrintInfo(MPEG_Args *m3args);
    bool Decode(MPEG_Args *m3args);
    void Terminate(uint32 returncode, MPEG_Args *m3args);
private:
    SynthesisFilter *filter1, *filter2;
    Obuffer *buffer;
    LayerIII_Decoder *l3decoder;    // ...
};
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

221

Review

- Perceptual coding techniques and standards
- MP3 and AAC encoders and decoders
- APIs for streaming conversion

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

222

Coding Examples

- Using libMad in a sentence
- Making a player transport bar
- Advanced streaming formats
- Loss-less compression APIs

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

223

What's Next?

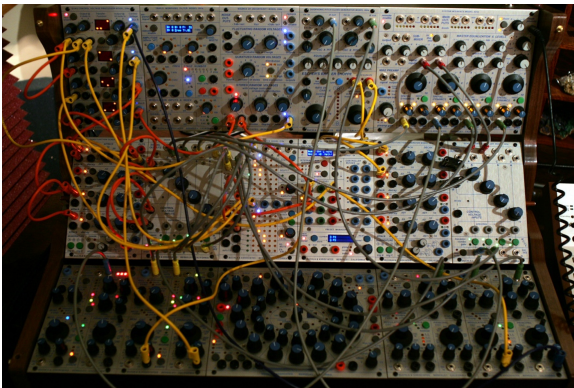
- Topic 4 Readings
 - Wikipedia: Plug-in (computing)
 - Audio Unit Development fundamentals
 - Steinberg VST Plug-in SDK Doc
 - Linux Audio Plug-ins: LADSPA
 - Writing Browser Plug-ins in Java, Javasoft

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

224

Topic 4: Plug-in APIs



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

225

Topic 4: Plug-in Architectures

- Browser, VM, Player, and Tool plug-in architectures
- APIs
 - Front, middle, and back-end
- Targets
 - Browsers
 - VMs
 - Players
- Code examples

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

226

Topic 4 Readings

- Readings
 - Wikipedia: Plug-in (computing)
 - Audio Unit Development fundamentals
 - Steinberg VST Plug-in SDK Doc
 - Linux Audio Plug-ins: LADSPA
 - Writing Browser Plug-ins in Java, Javasoft

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS & TECHNOLOGY PROGRAM

227

Plug-in Basics

- What are you plugging in to?
 - Web browser
 - Virtual Machine
 - Media content editing tool or player
 - Other (?)
- APIs to consider
 - “Central stores” or main()
 - “Driver” or back-end
 - Doing the work

CSB MAT 240A stephen@mat.ucsb.edu 2008

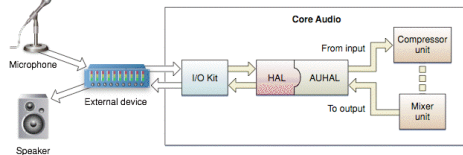
MEDIAS & TECHNOLOGY PROGRAM

228

Round-trip Example

- Stream in sample buffers
- Call a list of processing units
- Play output sample buffer stream
- HAL, host & client

Figure 3-4 The AUHAL used for input and output



CSB MAT 240A stephen@mat.ucsb.edu 2008

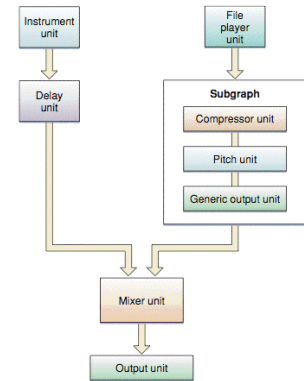
MEDIA RATS & TECHNOLOGY PROGRAM

229

Complex DASP Graphs

- Plug-in composition & graph management

Figure 2-3 A subgraph within a larger audio processing graph



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

230

Kinds of Plug-ins

- Web browsers
 - HTTP interface
 - Window management
 - Tags and types
- Java/Smalltalk VMs
 - Calls from above (native methods)
 - Interface to VM kernel (wrappers for special functions)
- Media Tools -- VST, TDM, DirectX
 - Buffer I/O and streaming
 - Buffer processing
 - GUI

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

231

Generic Plug-in Outline

- Init & check-in
- Register 1 or more call-back functions
- Start server thread
- Handle call-backs or buffer interrupts
- Shut down & clean-up on exit
- Sound familiar (as APIs)?

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

232

Steinberg VST Plug-ins

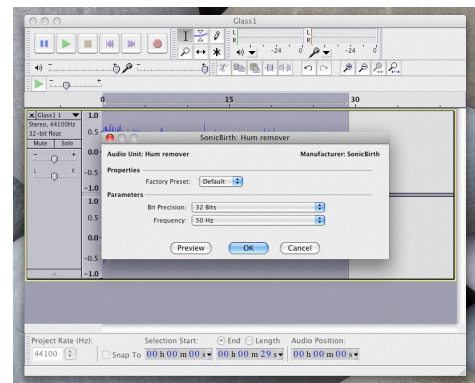
- What is a VST Plug-in? -- In the widest possible sense a VST-Plug-in is an audio process. A VST Plug-in is not an application. It needs a host application that handles the audio streams and makes use of the process the VST plug-in supplies.
- See VST Plug-Ins SDK 2.0.pdf

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

233

VST in Audacity

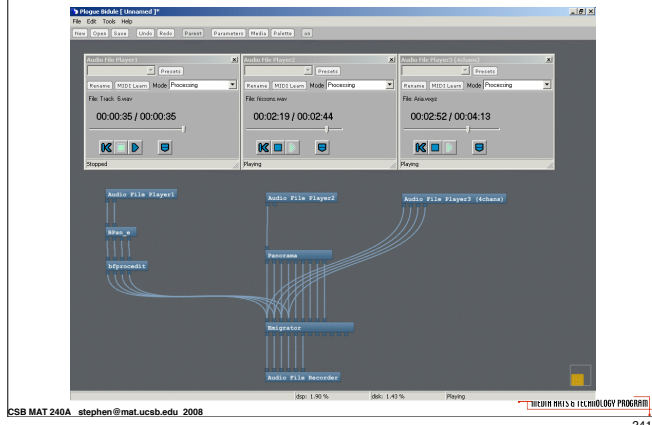


CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA RATS & TECHNOLOGY PROGRAM

234

Plogue Bidule Host



CSB MAT 240A stephen@mat.ucsb.edu 2008

241

Browser Plug-ins

- MIME types table
 - Parallel to UNIX device table
 - Handlers can be dynamically loaded
- EMBED HTML tag
 - Attaching an application to a page
- Registering for a MIME type
 - Function `NPP_GetMIMEDescription()` is called

CSB MAT 240A stephen@mat.ucsb.edu 2008

242

Plug-in Flow

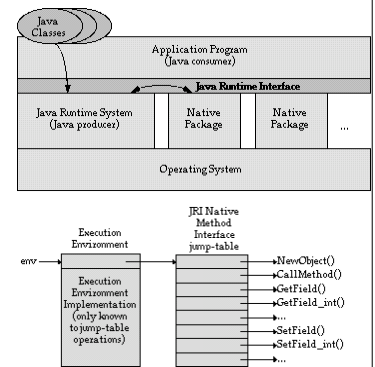
- See <http://developer.netscape.com/docs/manuals/communicator/plugin/index.htm>
- Load and instantiate plug-in
- Call to `NPP_Initialize()` -- class set-up, allocate memory shared by all instances
- Call to `NPP_New()` for each viewer
- Call to `NPP_SetWindow` to pass window to plug-in
- ... plug-in takes over, stream I/O, display, etc.
- Call to `NPP_Destroy()` when user leaves page or closes active view
- Call to `NPP_Shutdown()` to free memory
- See examples

CSB MAT 240A stephen@mat.ucsb.edu 2008

243

Java Runtime Interface

- Java VMs and native-methods
- Calling in to a Java VM
- Environment pointer and jump-table



CSB MAT 240A stephen@mat.ucsb.edu 2008

244

Using JRI

- Local and global references
- Constructing objects
- Accessing object fields
- Calling methods
- Defining Native Methods
- Working with Exceptions
- Security
- Plug-ins, VMs and Pluglets

CSB MAT 240A stephen@mat.ucsb.edu 2008

245

VisualWorks User Primitives

1. Copy the contents of the userprim folder into a new directory.
3. Copy 'StdCLib' and 'NetManage WinSock Lib' from the bin:powermac folder of your
2. Optional: Replace the PICT resource ID 128 in file vw5i.3.rcs with your own picture using ResEdit.
3. Modify the versionString value in the 'customiz.c' file if necessary.
4. Replace 'validate.o' in the following line of makefile 'VisualWorksUser.make' with the name of the file containing your own implementation of primitives:
 - Objfiles = customize.o validate.o
- 5.. in XCode

CSB MAT 240A stephen@mat.ucsb.edu 2008

246

VisualWorks User Primitives

```
See UserPrim.h
/*
 * UPCToSTbool( aUpBool ) --
 * Convert aUpBool to a Smalltalk Boolean.
 */
extern upHandle UPENTRY UPCToSTbool(upBool);

/*
 * UPInstVarAt( aUpHandle, index ) --
 * Return the value of the one based index'th instance variable.
 */
extern upHandle UPENTRY UPInstVarAt(upHandle, upInt);

/*
 * UPisArrayOfFloat(aUpHandle) --
 * Is aUpHandle a Array of Floats?
 */
extern upBool UPENTRY UPisArrayOfFloat(upHandle);
```

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

247

Other Plug-in Examples

- Audio (see above)
- Science.org Netscape plug-in step-by-step
- Apps with plug-ins
 - Illustrator, photoshop, etc.
- Java pluglet
- Graphics/GUI
 - Windowing, OpenGL, JUCE

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

248

Review

- Plug-in APIs
 - What's the set-up process
 - What are the call-back fcn signatures you have to write?
 - Process sample buffer
 - Control event or param update
 - Set-up and auto-discovery
- Host issues
- Writing hosts

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

249

Coding Examples

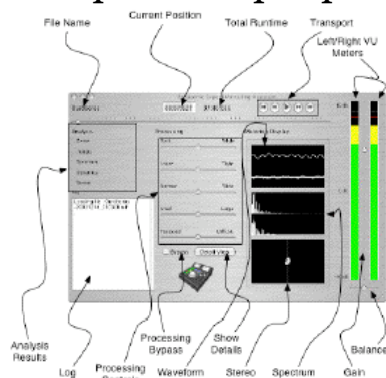
- Imports from MAT240D
- Plug-ins using CSL
- JUCE plug-ins
- Spatialization and geometry streaming
- Tools in DAWs
- Links to AlloSphere

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

250

Topic: Wrap-up



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

251

Topic: Course Review

Sound I/O APIs and Audio Streaming

- Introduction
 - Digital audio basics
 - SW development tools
- Topic 1: Sound IO APIs
- Topic 2: Sound Storage/Streaming Formats
- Topic 3: Audio Compression Formats
- Topic 4: Plug-in DSP/Streaming Architectures

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIAS RATS & TECHNOLOGY PROGRAM

252

Not Covered Here

- MAT 200C: MM engineering theory: the “why?”
- MAT 201B: CS & programming theory & practice
- MAT 240D: Sound synthesis techniques
- MAT 240E: Control & streaming (MIDI, OSC)
- MAT 594O: Embedded & mobile systems

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

253

Putting it all together

- Audio IO in applications
 - Sound/music apps
 - Telecomm apps
 - MM creation
 - Audio in general-purpose apps
- DAWs & DASP tools
- Performance instruments
- Other MAT 240 topics
- MAT 275, MAT 594O, MAT 201B

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

254

Where do we go from here?

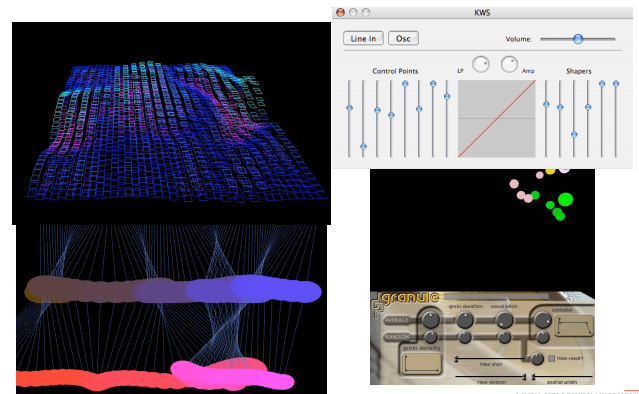
- MAT 240B: Spectral Transformations
 - Frequency analysis and processing
 - Digital filters and effects
- Rest of 240
 - Spatial, synthesis, control, databases
- CSL, Siren, GLV, Synz, MINT Projects at UCSB
- Other MAT core courses

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

255

MAT 240 Student Projects



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

256

Thank you!



CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

257



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240A:
DIGITAL AUDIO PROGRAMMING:
THE SERIES,
 EPISODE 1, TAKE 5
SOUND I/O APIS
 FALL, 2008

PRESENTATION SLIDES BY STEPHEN TRAVIS POPE,
 STP@MAT.UCSB.EDU

CSB MAT 240A stephen@mat.ucsb.edu 2008

MEDIA ARTS & TECHNOLOGY PROGRAM

258

MAT 240B Digital Audio Programming: Spectral Transformations (Winter, 2007)

The MAT 240 sequence is a six-part (two-year) practical programming course; it consists of hands-on software development devoted to digital audio and multimedia development. Students read a selection of papers from the literature, with the emphasis on learning to use the current state-of-the-art programming methods, tools, and programming interfaces. Programming assignments involve C/C++/Java software development on Linux, Macintosh, MS-Windows, various plug-in architectures, and possibly other platforms.

MAT240B focuses on the development of software for the spectral processing of digital sound. We will use several libraries for spectral analysis, processing, and resynthesis (e.g., FFT libraries and vocoder programs), as well as exploring digital filter design and other classes of spectral effects and transformations.

Students are expected to know the basics of digital audio signal representation and processing, and to be proficient in C, C++, or Java (Smalltalk, SuperCollider, LISP, and/or XML are a plus). Grading will be on the basis of in-class participation and programming projects.

Course Outline

- Time-domain and Frequency-domain Signals
- Transformations and Analysis/Synthesis Systems
- Fourier Analysis and the FFT
- FFT Software Libraries
- FFT-based Vocoder & Compression
- Digital Filters: Theory and Design
- FIR and IIR Filter Libraries
- Linear Prediction and LPC vocoders
- Pitch Detection and Analysis
- Applications

Instructor

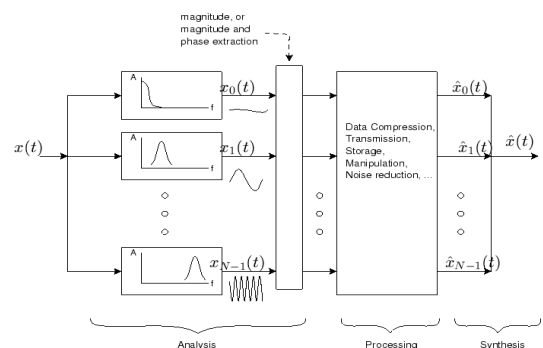
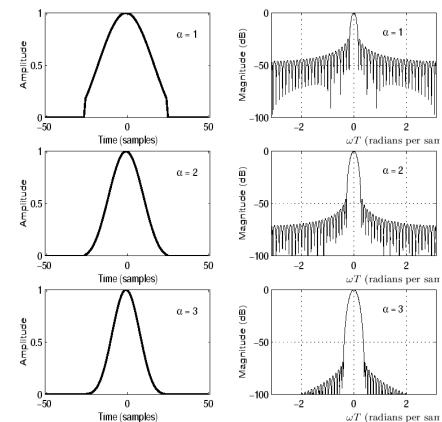
- Stephen T. Pope

Meeting time and place

- M/W 1:00 - 2:50 PM; Music 2215

Electronic Resources


- Course Web Site
See <http://www.create.ucsb.edu/240>
- Email Mailing List
Post to 240@zydeco.mat.ucsb.edu
See <http://zydeco.mat.ucsb.edu/mailman/listinfo/240> to join



MAT 240B Code Archive

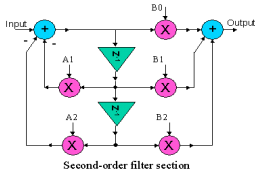
- DASP Frameworks
- Filter Design
- FFT packages
- Analysis/Synthesis
- Utilities

▶	AFsp-v7r1	⊕	2.8 MB
▶	AVIPlugin	⊕	2 MB
▶	carl-0.02	⊕	13.7 MB
▶	ClassCode	⊕	3.8 MB
▶	cmix-UNIX	⊕	2.8 MB
▶	csound504_src	⊕	19.5 MB
▶	DAFD	⊕	160.3 KB
▶	ellipse	⊕	126.2 KB
▶	FFTReal	⊕	41.9 KB
▶	fftw-3.1.2	⊕	21.5 MB
▶	filter	⊕	15.1 KB
▶	FilterDesign-v5r1	⊕	608.2 KB
▶	fir	⊕	44.1 KB
▶	JSyn	⊕	2.4 MB
▶	Kabal-AFsp	⊕	4 MB
▶	libsamplerate-0.1.2	⊕	8.2 MB
▶	libsndfile-1.0.17	⊕	12.7 MB
▶	Loris-linux	⊕	6 MB
▶	Loris-Mac	⊕	7.1 MB
▶	LPC-ccrma	⊕	2.9 MB
▶	LPC-Plugin	⊕	90.9 KB
▶	Lynn_Fuerst	⊕	2.3 MB
▶	MP3Player	⊕	6.8 MB
▶	newmat11	⊕	1.1 MB
▶	PRC_Book_Code	⊕	11.2 MB
▶	pv	⊕	48.5 KB
▶	pv_fix	⊕	171.8 KB
▶	pv2	⊕	760.6 KB
▶	PVC-3.0-linux	⊕	1.5 MB
▶	PVoc-Plugin	⊕	84.7 KB
▶	SC3_FFT_Examples	⊕	19 KB
▶	SDIF_Code	⊕	7.9 MB
▶	signal1.4	⊕	1 MB
▶	snack2.1.3	⊕	2.2 MB
▶	sndan	⊕	3 MB
▶	soundtools	⊕	345 KB
▶	srconv	⊕	61.9 KB
▶	DSP_Primer.c	⊕	29.4 KB
▶	README	⊕	1.8 KB




MAT 240B: The Digital Audio Programming Series

— [MAT 240B: Spectral Transformations
Stephen Travis Pope, stephen@create.ucsb.edu
Winter Quarter, 2007




Second-order filter section



1

The MAT 240 Series

- [Hands-on programming courses using (primarily) C, C++, and Java for digital audio application development
- [Six-quarter (two-year) course series
- [Students use a variety of software development tools on MS-Windows, Linux/UNIX, and the Macintosh




MAT 240B

2

MAT 240 Topics (6 quarters)

- [A: File I/O and Streaming Media
- [B: Spectral Transformations
- [C: Spatial Processing of Sound
- [D: Sound Synthesis Techniques
- [E: Control and Multi-rate Processing
- [F: Analysis and Music Information Retrieval




MAT 240B

3

MAT 240B Course Topics

- [Time-domain and Frequency-domain Signals
- [Transformations and Analysis/Synthesis
- [Fourier Analysis and the FFT
- [FFT Software Libraries, Vocoders & Applications
- [Digital Filters: Theory and Design
- [FIR and IIR Filter Libraries
- [Linear Prediction and LPC vocoders
- [Pitch Detection and Analysis
- [Advanced Applications




MAT 240B

4

Logistics

- [Meeting time - Monday/Wednesday 1:00-2:50 PM,
— Music 2215 (CREATE class room)
- [Student participation (required)
- [Grading - Participation, presentations, projects
- [Office Hours - M/W 10:00 - 11:00 AM, South Hall 4340
- [Web site and mailing list - <http://create.ucsb.edu/240>
— 240@zydeco.mat.ucsb.edu




MAT 240B

5

Textbooks & References

- [Curtis Roads: The Computer Music Tutorial (MIT Press)
- [F. Richard Moore: Elements of Computer Music (Prentice-Hal)
- [D. Gareth Loy: Musimathics (MIT 2006)
- [Ken Steiglitz: A Digital Signal Processing Primer (Addison-Wesley)
- [P. Lynn & W. Fuerst: Introductory Digital Signal Processing (J. Wiley)
- [Julius O. Smith: CCRMA Music 420 Notes (heavily quoted)
- [WWW links at <http://create.ucsb.edu/240>



MAT 240B

6

Software We'll Use (mostly C and C++)

- F. R. Moore Book Examples
- UCSD CARL filters & SR-converter
- MIT FFTW libraries
- CREATE CSL & MixViews source
- Csound PV and LPC code
- Various filter design programs
- Squeak PVOC and LPC plug-ins
- Kabal's LibTSP
- WAVE Signal Processing Programs



MAT 240B

7

MAT 240B Outline

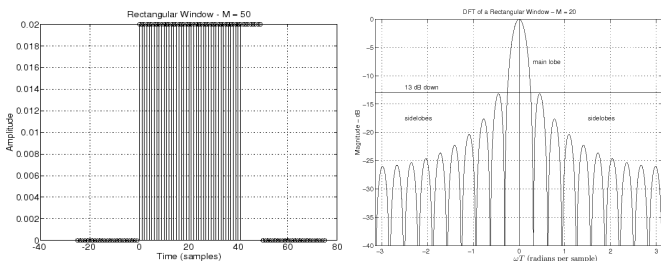
- Week 1 Time-domain and frequency-domain signals - transformations and analysis/synthesis systems
- Week 2 Fourier analysis, windowing, and the FFT - FFT software libraries
- Week 3 FFT-based vocoders & compression - spectral descriptions and file formats
- Week 4 Digital filters: theory and design
- Week 5 FIR and IIR filter libraries
- Week 6 Filter examples and applications
- Week 7 Linear prediction and LPC vocoders
- Week 8 Pitch detection and analysis
- Week 9 Integrated Applications
- Week 10 Random topics, review, projects



MAT 240B

8

Topic 1: Time- and Frequency-domain Signals



MAT 240B

9

Topic 1: Time- and Frequency-domain Signals

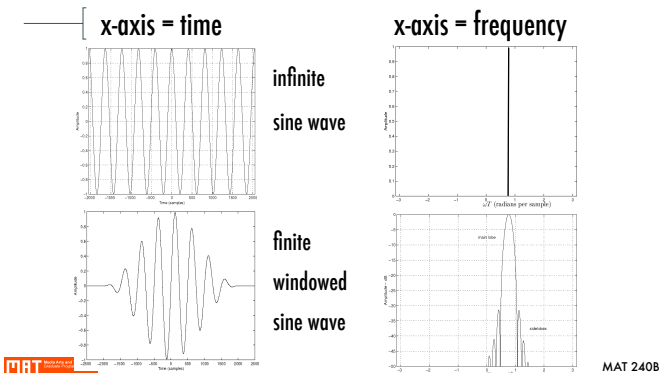
- Waveforms and spectra
- Analysis/synthesis systems
- Analysis for compression
- FFT & Convolution
- Readings:
 - Smith: Lecture 1
 - Moore: Chapter 2
 - Steiglitz: Chapters 8
 - Lynn&Fuerst: Chapter 3



MAT 240B

10

Functions of Time (wave) and Functions of Frequency (spectrum)



MAT 240B

11

Signal Domains

- Time-domain
 - Sampled discrete audio signal
 - Control signal
 - 1-D, n-D, geometry
 - Impulse response
 - Other T-D semantics
- Frequency-domain
 - Spectrum (inst. frame or continuous 3-D)
 - F_0 track
 - LPC spectrum
 - Wavelet-modulus-domain



MAT 240B

12

Frequency-domain Transforms

- [How to derive a spectrum from a time-domain signal?
- [Swept band-pass filter
- [Swept oscillator (heterodyne)
- [Filter bank
- [Fourier transform
- [Polynomial approximation (linear prediction)
- [Others?



MAT 240B

13

Fourier Analysis/Synthesis

- [Fourier theorem - periodic signals as sum of related sines
- [Fourier transform - continuous/discrete, infinite-/short-time

Time Duration		
Finite	Infinite	
Fourier Series (FS)	Fourier Transform (FT)	cont.
$X(k) = \frac{1}{P} \int_0^P x(t) e^{-j\omega_k t} dt$	$X(\omega) = \int_{-\infty}^{+\infty} x(t) e^{-j\omega t} dt$	time
$k = -\infty, \dots, +\infty$	$\omega \in (-\infty, +\infty)$	t
Discrete FT (DFT)	Discrete Time FT (DTFT)	discr.
$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\omega_k n}$	$X(\omega) = \sum_{n=-\infty}^{+\infty} x(n) e^{-j\omega n}$	time
$k = 0, 1, \dots, N-1$	$\omega \in (-\pi, +\pi)$	n
discrete freq. k	continuous freq. ω	



MAT 240B

14

DSP Notations (from Smith)

Frequency and Time:

- ω denotes continuous radian frequency (rad/sec)
- f denotes continuous frequency in Hertz (Hz)
- $\omega = 2\pi f$
- ω_k denotes discrete frequency, $\omega_k = 2\pi(k/N)f_s$
- $\omega, \omega_k \in \mathbb{R}$ (frequencies are always real)
- T = sampling interval (sec)
- f_s = sampling rate, $f_s = \frac{1}{T}$
- $t_n = nT$ (discrete time)
- $n, k \in \mathbb{Z}$ (integers)
- $t, t_n \in \mathbb{R}$ (times are always real)

Signal Notation:

- $x \in \mathbb{C}^N$ means x is a length N complex sequence
- $\omega = \omega(\cdot)$
- $X = \text{DFT}(x) \in \mathbb{C}^N$
- $X(k) = \text{DFT}_k(x)$
- $X(k) \in \mathbb{C}$
- $x(n) = \text{IDFT}_n(X)$
- $x(n) \in \mathbb{R}$ or \mathbb{C}
- $n, k \in \mathbb{Z}$ or $n, k \in \mathbb{Z}_N$ (integers modulo N)
- For $x \in \mathbb{C}^\infty$, $X = \text{DTFT}(x) \in \mathbb{C}_{2\pi}^\infty$
- \bar{x} = conjugate of x
- $\angle x$ = phase of x



MAT 240B

15

Interpretations of the DFT

- [Multiply the signal by a swept sine wave:
 $x(t) e^{-j\omega t}$ for ω in $+\infty$ to $-\infty$
- [Matrix multiplication $X = Wx$ (W is square of coefficients)
- [Projection coefficients of signal x onto N sinusoidal components (equal-spaced complex roots of 1)
- [Coordinate transform from \mathbb{R}^N to $X(k)$



MAT 240B

16

DFT in Pseudocode

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\omega_k n}$$

$$k = 0, 1, \dots, N-1$$

- [To get a spectrum $X(k)$ (k is spectral bin #)
- [For every output frequency k from 0 to $F_s/2$
 - Compute the vector product of the input signal $x(n)$ with a complex sine of freq. ω_k (i.e., $e^{-j\omega_k n}$, bin center freq.)
 - That (scalar) value is the energy at freq. ω_k , or $X(k)$
- [Process requires $O(n^2)$ (or worse) operations
- [Process is hard to accelerate (even with huge buffers)



MAT 240B

17

DFT in Pictures (from Loy Musimathics)

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	
k	Test Signal	Cosine Probe Phasor	Cosine Product Signal	Sine Probe Phasor	Sine Product Signal	Cosine Sum (Real)	Sine Sum (Imaginary)	
0						0	+	0i
1						0	+	0i
2						0	+	-0.5i
3						0	+	0i
4						0	+	0i
5						0	+	0i
6						0	+	0.5i
7						0	+	0i



MAT 240B

18

Slow Fourier Transform in SuperCollider

```
sweepsize = 1024;           // source: approx. pulse, 10 overtones
source = Signal.sineFill(sweepsize, [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]);
sweep = Signal.newClear(sweepsize); // create swept sine buffer
index = 0.0;
sweepfcn = { arg x, i; var frq; // function to create a swept sine
  frq = x;
  index = index + frq;
  sin(index)
};
sweep.waveFill(sweepfcn); // fill swept sine buffer
sinefcn = { arg x, i; sin(x) }; // function to create a static sine
sine = Signal.newClear(sweepsize); // create sine buffer
spect = Signal.newClear(sweepsize); // "convolve" sweep * source
spectfcn = { arg x, i; // fcn to "convolve" source & sine
  sine.waveFill(sinefcn, 0, (x*2pi)); // create correct freq. sine
  sum = 0.0; // mult sine * source and integrate
  (source * sine).integral / sweepsize
};
spect.waveFill(spectfcn, 0, 16); // fill "spectrum": 16 overtones
```



MAT 240B

19

FFT Code Examples

- SuperCollider spectrum display
 - Basic spectrum display, IFFT signal synthesis
- FFT example in SuperCollider - FT via signal multiplication
- TCL Examples - "widget" example in Snack
- Basic FRM Code - FFT and phase vocoder



MAT 240B

20

Getting started Coding

- Platforms, tools, compilers, libraries
- See MAT 240A Web site

Platform	IDE	Snd Libs
MS-Windows	VisualStudio, gcc	DirectX, ASIO
Linux/UNIX	gcc, KDE	OSS, ALSA, Jack
Macintosh	Xcode, gcc	CoreAudio
All	Eclipse, Smalltalk	PortAudio, CSL, JavaSound, JavaMF, libSndFile



MAT 240B

21

Code examples

- Basic FFT function
 - DSP_Primer.c
 - pv FRM book
 - SndAn (UIUC)
 - Mxv (Scott, CREATE)
 - WAVE Signal
 - Csound pvan app.
- First steps
 - Live-in vs. file I/O
 - Windowing or not
 - FFT/op/IFFT in a loop
- Platform
 - PA-callback
 - CSL
 - VST plug-in
- Using FFTW
- Signal Windowing
 - See above packages



MAT 240B

22

About the Code.zip File

- AFsp-v7r1 - Peter Kabal's LibTSP audio IO, filters, etc. in C
- DAFD - Digital Audio Filter Design and Implementation Utility
- DSP_Primer.c - Digital Signal Processing Routines in C
- filter - UCSD CARL FIR/IIR filter implementation
- FFT_Examples.sc - SuperCollider code for a simple FT
- JSyn - Phil Burke's Java synthesis library, includes filters
- Loris - Kelly Fitz's Loris analysis/synthesis package in C++, Linux
- LPC Folder - Perry Cook's minimal LPC implementation
- Lynn_Fuerst - Code to "Introductory Digital Signal Processing," FFTW-312 - FFTW, see fftw.org
- pv - Dick Moore's vocoder as in "Elements of Computer Music"
- signal1.4 - John Culling's WAVE Signal Processing Programs for Linux
- snack2.1.3 - A sound toolkit for Tcl/Tk and Python by Kare Sjolander
- sndan - UIUC UNIX Sound Analysis Package



MAT 240B

23

Also get...

- libsndfile, portaudio, portmidi, rtmidi
- CSL
- CLAM
- Qt
- OpenGL
- Csound, SuperCollider



MAT 240B

24

Properties of the DFT

- [Linearity
- [Time reversal
- [Symmetry (phase, magnitude, conjugate)
- [Shift
- [Convolution
- [Correlation
- [Stretch/Repeat
- [Downsampling/Aliasing



MAT 240B

25

Why Fourier?

- [Physical relevance (how the ear works)
- [Link to filter banks and add. synth. (parameter est. for a model)
- [Can represent any LTI as a FT system
- [May be compact (and maybe not)
- [Can be implemented efficiently (FFT)
- [Some processes (complex filters, long convolution) are more efficient in the frequency domain
- [Many applications: analysis, coders, classifiers, processing, etc.



MAT 240B

26

What do you do with a Spectrum?

- [Peak-finding
- [Formant-finding
- [Peak-tracking
- [F_0 analysis
- [MFCC coefficients
- [Thresholding
- [Freq.-shifting
- [Additive cross-synthesis



MAT 240B

27

Spectral Processing Applications

- [CSL/FMAK peak finding/tracking
- [SMS in CLAM
- [MQAN in SNDAN
- [HPS in CSL
- [Other implementations and applications



MAT 240B

28

How do you make the FT Fast?

- [Insight into handling of interleaved DFT buffers
 - [Take the formulation of the DFT and split the input into buffers of even and odd samples, resp. (each as $1/2 F_s$ but with phase shift)
- $$V[k] = \sum_{n=0}^{N-1} W_N^{kn} v[n] \quad (\text{where } W \text{ is complex phasor})$$
- $$V[k] = \sum_{n \text{ even}} W_N^{kn} v[n] + \sum_{n \text{ odd}} W_N^{kn} v[n]$$
- $$= (\sum_{r=0}^{N/2-1} W_{N/2}^{kr} v[2r]) + W_N^k (\sum_{r=0}^{N/2-1} W_{N/2}^{kr} v[2r+1])$$
- [The two sums are nothing else but $N/2$ -point transforms of, respectively, the even subset and the odd subset of samples.

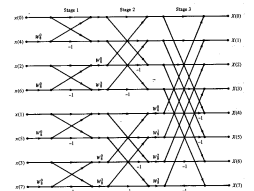


MAT 240B

29

OK, so...

- [We can keep on reducing and recombining to reduce any power-of-2-length FT to a set of length-2 FTs that are then combined with the complex exponential factors
- [This is easy, but requires a lot of shuffling of the data set for the even/odd subdivision, leading to the famous FFT butterfly diagram



MAT 240B

30

Other Frequency Transformations

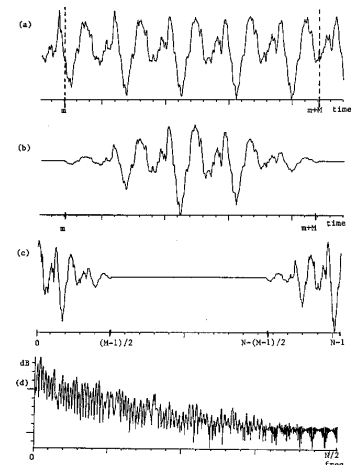
- Higher-level analysis based on FFT
 - Partial Trackers
 - Signature classifiers
 - Noise reduction
 - Pitch detection
- Linear Prediction and LPC
- Wavelet Transform



MAT 240B

31

Windowing



MAT 240B

32

Windowing

- Problem: the real world uses finite-length time-varying signals
- Solution: analyze signal in short “windows” of fixed size
- Impact: need to consider frequency response of the window function and the time/frequency accuracy trade-off

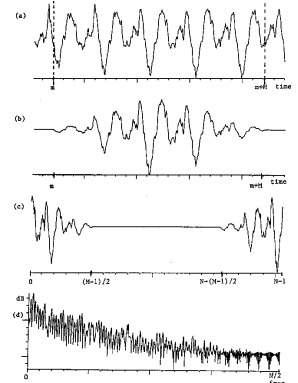


MAT 240B

33

Windowing Overview

- Input signal
- Windowed Signal
- Shifted (zero-phase) window
- FFT gives magnitude spectrum



MAT 240B

34

Windows and Spectral Smear

- Any finite-length window will degrade the accuracy of frequency analysis.
- Window spectra generally have a central lobe and symmetrical side lobes.
- There are many kinds of window functions with varying spectral properties.
- The important variables are (a) the width of the central lobe, (b) the 1st side-lobe level, and (c) the side-lobe roll-off rate

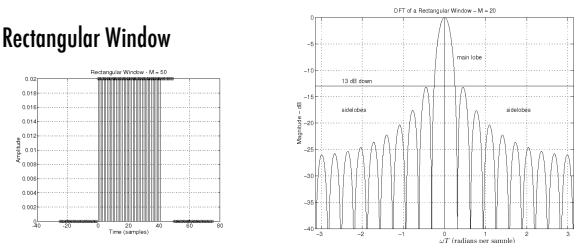


MAT 240B

35

Windows and their Spectra

- Rectangular Window



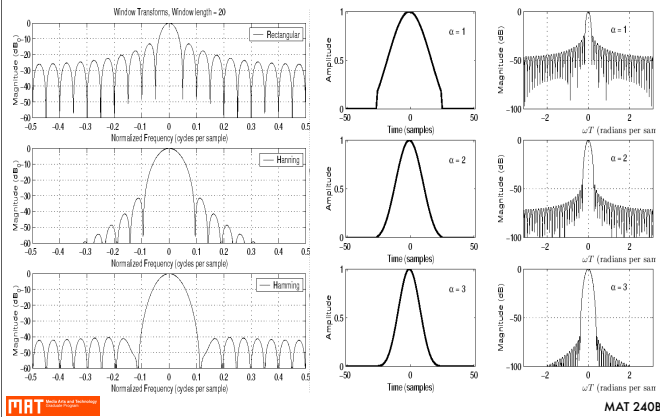
- Triangle (Bartlett) wider main lobe and lower 1st side-lobe (both by a factor of 2), as well as better roll-off



MAT 240B

36

Window Functions



37

General Windowing Issues

- It's an underconstrained design problem!
- Both time-domain and frequency-domain (as well as magnitude and phase) characteristics are relevant.
- There are some new (2000) techniques that promise good time and frequency features.

MAT 240B

MAT 240B

38

Main-lobe widths

- For $S = f_s / (\text{side-lobe width in Hz})$

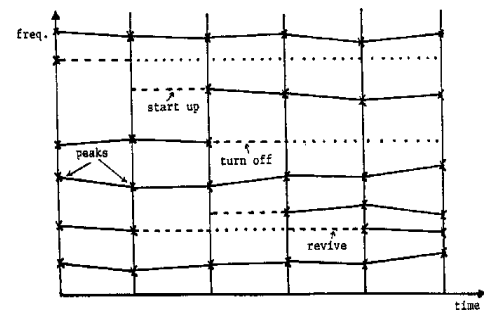
Window Type	Main-Lobe Width B_w (Hz)
Rectangular	$2S$
Hamming	$4S$
Hann	$4S$
Generalized Hamming	$4S$
Blackman	$6S$
L -term Blackman-Harris	$2(L+1)S$
Kaiser	depends on β
Chebyshev	depends on ripple

MAT 240B

MAT 240B

39

Advanced FFTs



MAT 240B

40

Advanced FFTs

- FFTW: Fastest FFT in the West
 - Portable C code
 - Supports non-power-of-2 window lengths
 - Very fast!
 - Large and complex GNU-ware
- Spectral Interpolation by Zero-padding (in the time-domain)

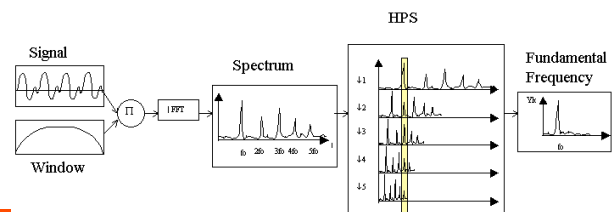
MAT 240B

MAT 240B

41

Harmonic Product Spectra

- Decimation of FFT spectra, summation, and spectral peak location
- Assumes overtones are significant, not that fundamental is



MAT 240B

42

Getting more from a Spectrum

- [1: Think of spectrum in signal correlation terms
- [2: Harmonic tones \rightarrow regularly spaced spectral peaks
- [Take autocorrelation of spectrum and look for peaks
- [The peaks relate to fundamental pitches in a warped frequency space (bin #)
- [They are called Mel frequency spectral coefficients



MAT 240B

43

Mel-Freq Cepstral Coefficients

- [Steps:
 - Signal
 - FT
 - Log magnitude (PDS)
 - Phase unwrapping
 - FT (or DCT)
 - [Name reversal
 - [Interpretations
 - Quefrency
 - Mel-scale
 - Mel-scale filters
- Instead of AC, use FFT or DCT of PDS
 — Leads to interesting statistics of higher-level spectral properties, see next section

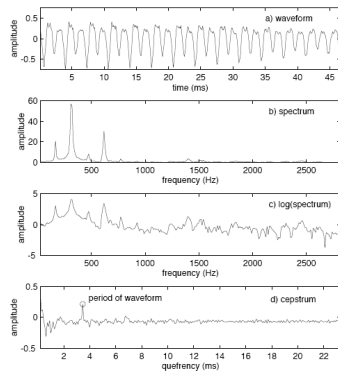


MAT 240B

44

MFCC Analysis

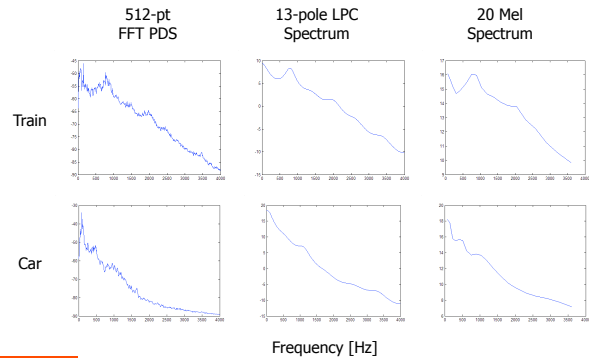
- [Analogy
 - Start with log spectrum of mixed complex tones: several sets of related partial peaks
 - Take, e.g., the autocorr. of the FFT PDS
 - Warped frequencies of peaks correspond to fundamental frequencies of overtone series



MAT 240B

45

Comparison FFT / LPC / MFCC (by Andrianakis & White)



MAT 240B

46

FFT Applications & Extensions

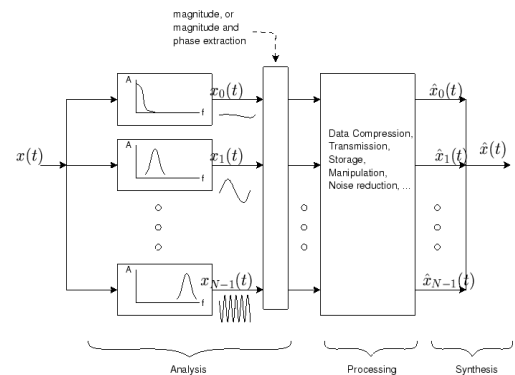
- [Vocoders and sinusoidal modeling
- [Partial Tracking (MQAN)
- [Transient Modeling
- [Deterministic + Stochastic Modeling
- [Bandwidth-enhanced partials
- [Continuous Phase Analysis



MAT 240B

47

Old-fashioned Vocoder



MAT 240B

48

Vocoder Structure

- [Read a window of the input signal
- [Apply (vector *) window function
- [Spectral analysis
 - FFT of window (FFT length may not = window size) (may maintain phase information)
 - Filter-bank analysis
- [...do something with spectral data...
- [Resynthesize signal from spectral data
 - Overlapp-add or sum-of-sines
- [Shift input by step size (may not = window size)

MAT

MAT 240B

49

Windowing for COLA

- [Overlap-add (OLA) resynthesis places special requirements on the window shapes and hop sizes (constant OLA constraint)
- [Options
 - Rectangular window with no overlap
 - Hanning window (raised cosine) with 50% overlap
 - Bartlett (triangular) with 50% overlap
 - Blackman window with 66% overlap

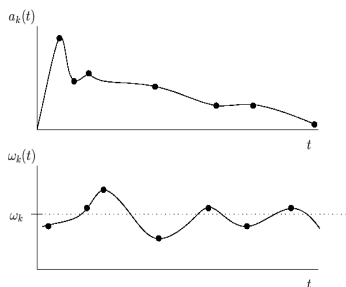
MAT

MAT 240B

50

Analysis Data

- [Remember: each "bin" or "channel" delivers functions of amplitude and center-freq (and possibly phase)



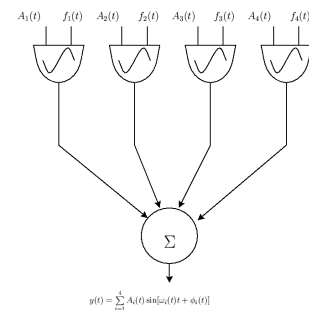
MAT

MAT 240B

51

Additive Synthesis Model

- [Sum-of-sines

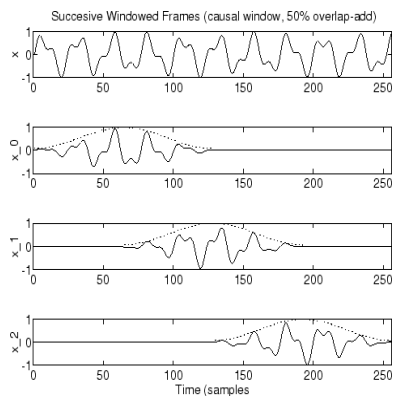


MAT

MAT 240B

52

Constant-OLA Model (IFFT)



MAT

MAT 240B

53

Vocoder Applications

- [Compression of "tonal" sounds via sinusoidal modeling
- ["Malleable" audio representation
 - Pitch/time warping
 - Transposition/time compression
 - Morphing and cross-synthesis
- [Pitch detection
- [Transcription
- [IFFT as synthesis technique

MAT

MAT 240B

54

FRM's PV Command

pv R N Nw D I P Np synt [-i] < samples > samples

- R - sampling rate in Hz
- N - FFT length in samples (must be 2^n)
- Nw - window size in samples
- D - decimation factor in samples [0 for synthesis only]
- I - interpolation factor in samples [0 for analysis only]
- P - oscillator bank pitch factor [0 for overlap-add resynth.]
- Np - linear prediction order [0 for none]
- synt - synthesis channel threshold [0 for all channels]
- i means r/w integer samples (floats by default)
- Filter fundamental = R/N
- Nw = N or $i * N$
- D = Nw / 8 (very conservative)
- Duration scale of result = $1/D$



MAT 240B

55

Vocoder Processing

- [Time warping = vary output buffer size
- [Pitch warping = transpose magnitude spectrum
- [Noise reduction = "gate" components given a threshold
- [Cross-synthesis = apply one spectrum to another
- [Convolution = multiply spectrum with spectrum of (e.g.,) impulse response
- [See SuperCollider and FRM pv code



MAT 240B

56

Problems with Vocoders

- [Poor model of noise, transients, "dense" sounds, inharmonic sounds
- [Hope for at most 1 partial per bin (needs fine-grained FFT)
- [Not reversible (formally)
- [May be compute-intensive
- [Can actually increase data bandwidth!

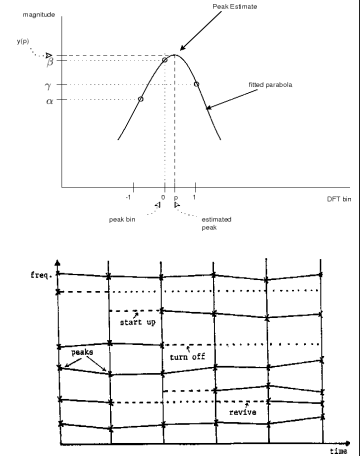


MAT 240B

57

Partial-Tracking

- [Spectral peak detection (parabolic interpolation, possibly using phase unwrapping)
- [Peak matching & tracking
- [Data on track births and deaths (important)
- [See mqan.c file in SNDAN



MAT 240B

58

In Code

- [Peak-picking and tracking rely on methods that incorporate numerical smoothing, heuristics, and iterative techniques
- [Examples: boundary conditions (ignore peak at 0 or $f_s/2$)
- [Thresholds and re-try (look for 2-4 peaks over threshold x , if too many, raise x , if too few, lower x)
- [Many other examples...



MAT 240B

59

Multi-stage Analysis Loops

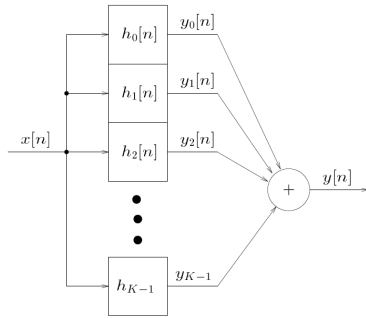
- [Segment/window input
- [Preliminary analysis (silence threshold)
- [Perform FFT
- [Spectral processing: peaks, tracks
- [Track birth/death statistics, transients
- [Pitch detection: HPS or autocorrelation
- [Other analysis processes (see later)
- [Feature weighting, data reduction



MAT 240B

60

Filter Banks



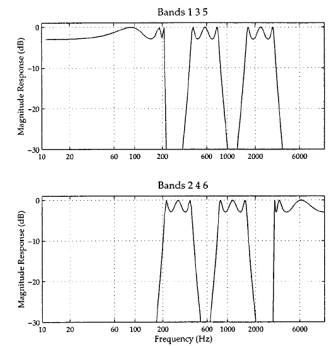
MAT

MAT 240B

61

Filter Banks

- [Filter bank vs. spectrum
- [When to use
- [How to implement
 - Analog
 - Digital



MAT

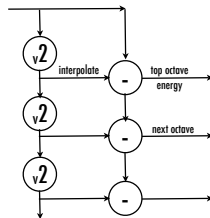
MAT 240B

62

Decimation for Filter Banks

Routine:

- Down-sample signal by a factor of 2
- Interpolate samples
- Subtract from original signal
- Difference is energy in top octave
- Repeat for next octave
- Filter banks widely used in coding and compression



MAT

MAT 240B

63

Working with Spectral Data

- [Instrument signatures
 - [Solo instrument identification
 - [Location of style-indicative instruments
 - [Still a challenge in mixed/compressed music
- [Pitch tracking
 - [Melody instrument vs bass tracking
 - [Generally requires segmentation and/or band-pass filtering
- [Spectral statistics
 - [Slope, centroid, band weights, variety, etc.

MAT

MAT 240B

64

Code Exercises

- [FFT spectral extraction
 - Real vs/ complex
 - Magnitude/phase spectra
- [Weighting of spectral data
 - Numerical scaling
 - Perceptual scaling
- [Peak location, continuation
- [MFCC and HPS

MAT

MAT 240B

65

Hybrid Models

- [Problem: not all useful sounds consist solely of slowly changing harmonic overtone spectra
 - Noisy transients (esp. attack)
 - Model as sines + noise
 - Inharmonic or stretched spectra

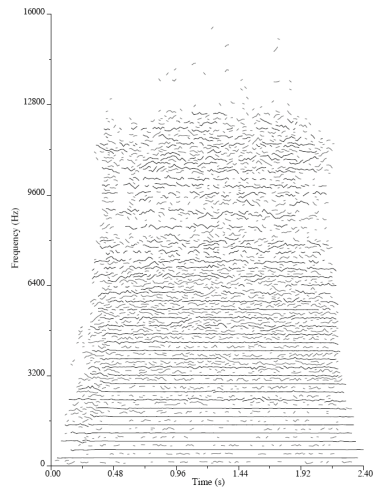
MAT

MAT 240B

66

Example: Breathy Flute

When are the upper partials so chaotic, and when is the FFT trying to model noise as partials?

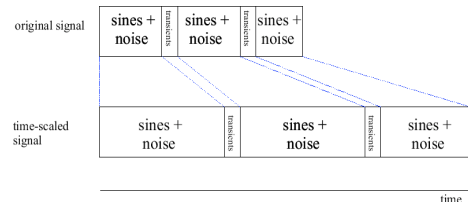


MAT 240B

67

Transient Modeling

- Find transient regions (based on some measure of change) and model them separately
- Use samples, wavelets, or LPC for transients
- Do not time-scale the transients!



MAT 240B

MAT 240B

68

Deterministic + Stochastic Modeling (SMS)

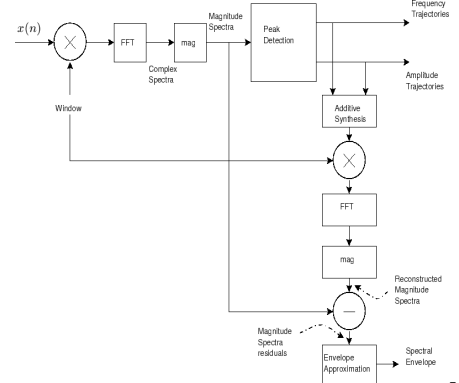
- FFT models noise as partials with chaotic phase (easy to spot in peak continuation process)
- FFT captures "deterministic" parts of spectrum
- Original signal - FFT resynthesis = noise component
- Use some good noise modeling technique such as low-order LPC for the stochastic component

MAT 240B

MAT 240B

69

SMS Analysis

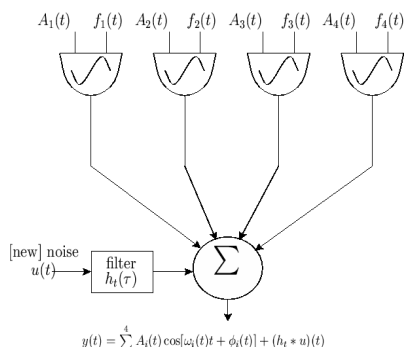


MAT 240B

MAT 240B

70

SMS Synthesis Model



MAT 240B

MAT 240B

71

Bandwidth-Enhanced Partial (Loris)

- Take FFT, track spectral peaks
- Tweak peak frequencies and times (!)
- Resynthesize and subtract to locate noise
- Model noise as a non-zero bandwidth energy distributed around the partials
- Each partial has a % of its energy dedicated to noise
- Allows for interesting transformations & morphs

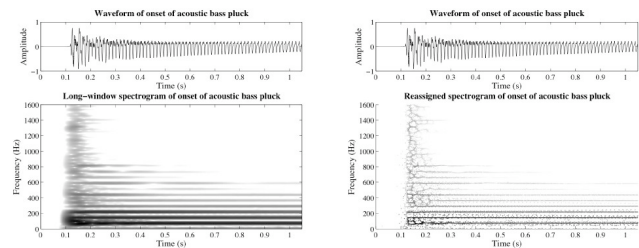
MAT 240B

MAT 240B

72

Loris Example: Bass Note

Before/after bandwidth reassignment



MAT

MAT 240B

73

Using Loris

- C++ library
- C API
- Python via SWIG
- Smalltalk via SWIG

```
#! python
import loris, os, time
print 'analyzing voice file'
a = loris.Analyzer( 100 ) # arg is freq resolution
cf = loris.AiffFile( '1.2a1.aiff' )
v = cf.samples()
sampleRate = cf.sampleRate()
vox = a.analyze( v, sampleRate )
loris.channelize( vox, loris.createFreqReference( vox, 0,
1000 ), 1 )
loris.distill( vox )
loris.exportAiff( 'voxtest.aiff', loris.synthesize( vox,
sampleRate ), sampleRate, 16 )
```

MAT

MAT 240B

74

FFT Applications

- Phase Vocoder
- Loris Morphing
- SMS Morphing
- Noise Reduction
- Compression and Perceptual Coding
- Dynamic Filters
- Others...

MAT

MAT 240B

75

Exercises

- IO Frameworks & plug-in architectures
- PV-based applications
- Higher-level spectral processing
- Multi-feature analysis/resynthesis

MAT

MAT 240B

76

Summary

- Time-domain and spectral-domain signals
- Fourier transforms and DTF
- FFT usage
- Windowing and signal massaging
- FFT Applications
- LPC
- Pitch-tracking
- Applications

MAT

MAT 240B

77

End of Topic 1



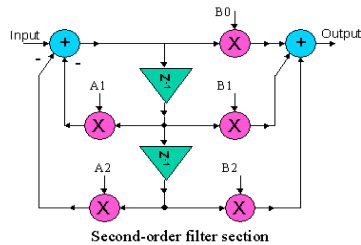
MAT

MAT 240B

78

MAT 240B: Digital Audio Programming: Spectral Transformations

Topic 2: Digital Filters



MAT

MAT 240B

79

Topic 2: Digital Filters

- Theory and Concepts
- Filter Design and Transformations
- Digital Filter Implementations
- Dynamic Filters
- Complex & Parametric Filters
- Applications

MAT

MAT 240B

80

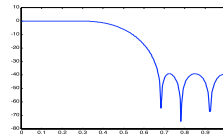
Background

Filter: spectral, phase response

- Time-and frequency-domain
- Window spectra

Analog filters

- R/L/C circuits and complex math
- The difference between resistance R and impedance Z



MAT

MAT 240B

81

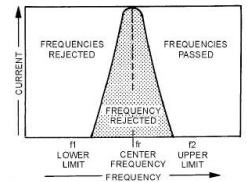
Kinds of Filters

Filter classes

- Low-pass, high-pass
- Band-pass, band-reject
- Composite, continuous multi-band, resonant

Filter topologies

- Matrix equation description
- Description of filter bands, response regions
- Filter freq/phase response



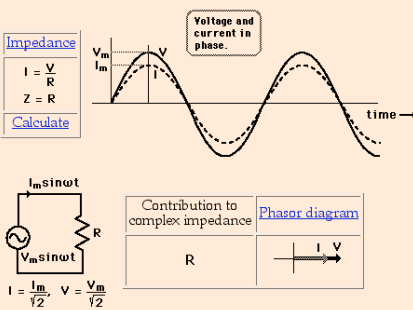
MAT

MAT 240B

82

Analog Electronics: Resistors

Resistor AC Response



Analogies for circuits: plumbing, acoustical systems

Resistance

$$I \propto V$$

MAT

MAT 240B

83

Capacitors

Capacitor AC Response

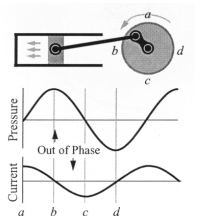
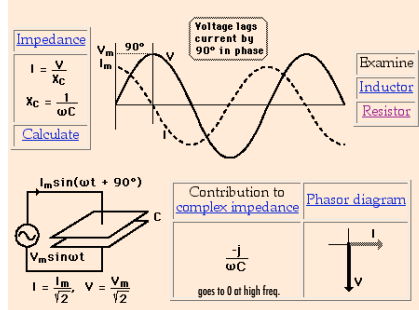


Figure 8.16
Motor and acoustical capacitor.

$$I \propto \Delta V$$

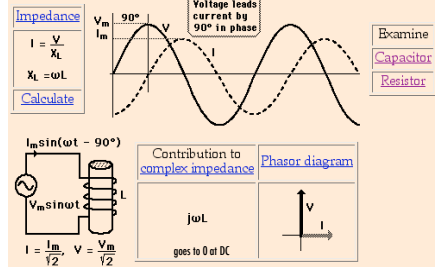
MAT

MAT 240B

84

Inductors

Inductor AC Response



MAT 240B

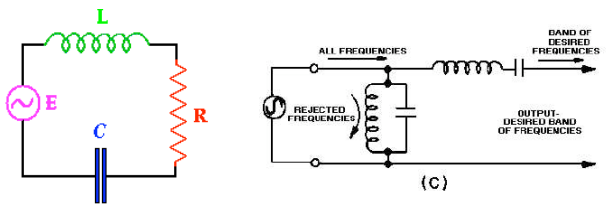
85

MAT 240B

85

RC, RL, and RLC Circuits

<http://webphysics.ph.msstate.edu/ic/library/21-5/CircuitE.html>



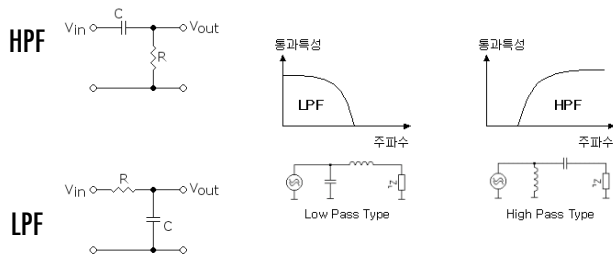
MAT 240B

86

MAT 240B

86

Analogue Filters

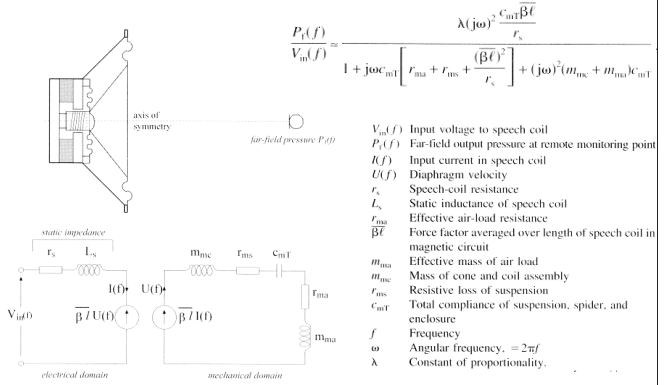


MAT 240B

MAT 240B

87

Acoustical Circuit Model Transfer Function



MAT 240B

MAT 240B

88

Digital Filters

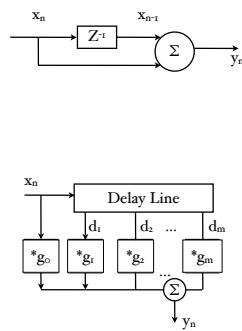
Simplest Example

- Forward Sample Averager
- $y_n = (x_n + x_{n-1}) / 2$
- "Function smoother"
- = Low-pass filter

General case: Multi-tap delay line

$$y_n = (g_0 * x_n) + (g_1 * x_{n-1}) + \dots + (g_n * x_{n-dm})$$

- Short delays: filter
- Long delays: reverberator



MAT 240B

MAT 240B

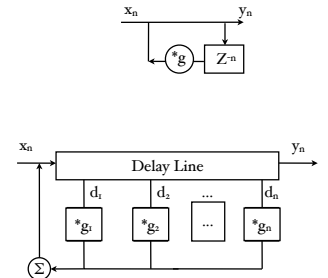
89

Another example

Feedback loop

- $y_n = (x_n + g * y_{n-1})$
- = comb filter

General form: feedback delay line



MAT 240B

MAT 240B

90

Filter Classes

- [Feed-forward delay-lines
 - Play an impulse through it, output ends at most dm samples later
 - = Finite impulse response (**FIR**) filter
- [Feed-back delay-lines
 - Play an impulse through it, output recirculates "forever"
 - = Infinite impulse response (**IIR**) filter



MAT 240B

91

Filter Classes

- [IIR Filters (Feedback systems)
 - Speed/space efficient and flexible
- [FIR Filters (Delay lines)
 - Impulse response is simply the coefficients
 - Phase is linear if coefficients are symmetrical (i.e., $b_0 = b_n, b_1 = b_{n-1}$, etc)



MAT 240B

92

General Form

- [Combination of feed-forward and feed-back terms (poles and zeros)
- [Feed-forward weighted averager = 1 zero

$$y(n) = A_0 * x(n) + A_1 * x(n-1)$$
- [Feed-back = comb filter = 1 pole

$$y(n) = A_0 * x(n) - B_1 * y(n-1)$$
- [General 2-pole/2-zero

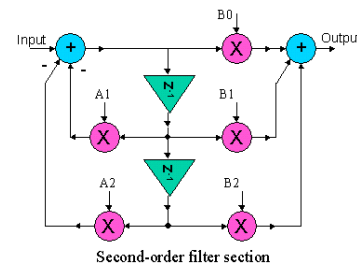
$$y[n] = g_0 * x[n] + g_1 * x[n-1] + g_2 * x[n-2] - h_1 * y[n-1] - h_2 * y[n-2]$$



MAT 240B

93

2nd-Order General Form



$$y(z) = x \prod_{i=1}^n \frac{1 - a_1 z^{-1} - a_2 z^{-2}}{b_0 + b_1 z^{-1} + b_2 z^{-2}}$$



MAT 240B

94

General Form to Z-form

- [$Y[n]$ polynomial
- [Substitute z for $n-1$, z^2 for $n-2$, etc.
- [Solve polynomial for roots in z
- [Example IIR system

$$y[n] = x[n] + k * y[n-1]$$
- [Expressed in Z:

$$Y(z) = X(z) + K * (Y(z) * z^{-1})$$

$$H(z) = Y(z) / X(z) = 1 / (1 - kz^{-1})$$

$$= z / (z - k)$$
- [This is 0 for $z = 0$ and infinite for $z = k$
- [These are the 1 zero and 1 pole of the function



MAT 240B

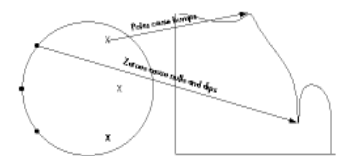
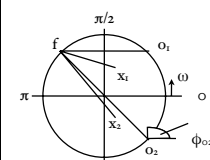
95

Pole-Zero Diagrams

- [Z-plane and the unit circle (rubber sheet analogy)
- [Poles and zeros as roots of xfer function

$$|H(\omega)| = \frac{\text{Product of vectors to zeros}}{\text{Product of vectors to poles}}$$

$$\Theta(H(\omega)) = \sum \text{zero angles} - \sum \text{pole angles}$$

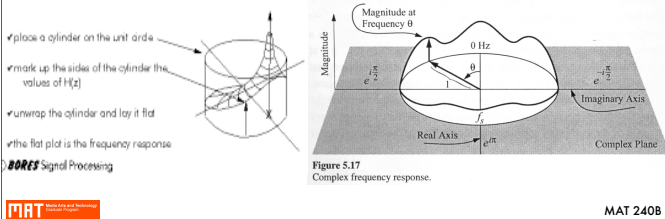


© BORES Signal Processing

96

Pole-Zero Diagram Examples

- See DSP Skriptum p. 80 (or other reference)
- Other topics
 - Frequency transformation (see below)

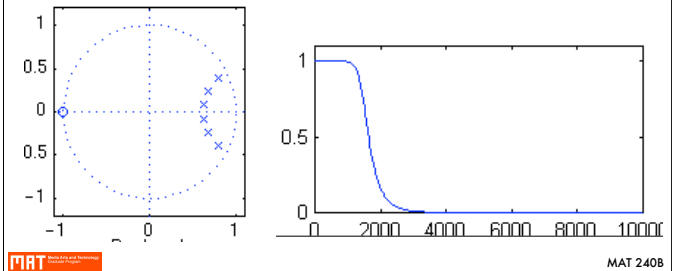


MAT 240B

97

Example: 6-pole low-pass Butterworth Filter (wmin.ac.uk)

- $F_s = 20 \text{ kHz}$, $F_0 = 1500 \text{ Hz}$
- Designed using MATLAB

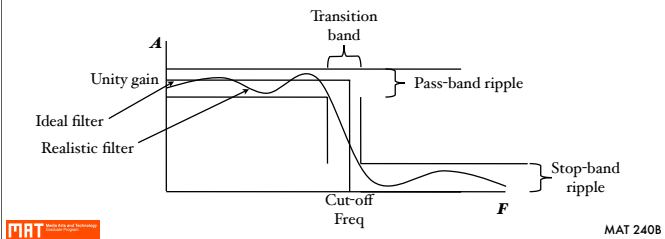


MAT 240B

98

Real-world Filters

- See CARL Filter Program filter.c
- See cmix Filters
- General Filter Design

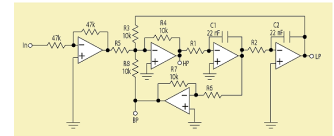


MAT 240B

99

Filter Topologies

- Each has different characteristics
 - Butterworth: flat pass-band response, monotonic stop-band attenuation
 - (T)Chebyshev: equal-sized errors on pass-band and stop-band, equiripple above or below knee
 - Bessel: linear phase in pass-band
 - Elliptical: flexible specifications, equal-sized errors in both bands
 - Cauer: excellent stop-band
 - ...many others



1. The building block of the basic filter provides low-pass, band-pass, and high-pass outputs.

MAT 240B

100

Complex Filters

- Two Options
 - Higher-order filters
 - Series/Parallel combinations of simple filter
- Some design programs decompose, some combine
- Parametric Filters
 - Some topologies allow design with Q separated from F_0

MAT 240B

101

Problems with Digital Filters

- Numerical stability (poles on the unit circle are very vefry bad, lead to ringing)
- Round-off errors are everywhere (floats are limited precision)
- Efficiency (higher-order filters may be slower than FFT/IFFT combination)

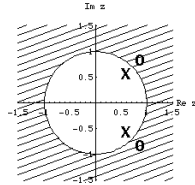
MAT 240B

MAT 240B

102

All-Pass Filters (?)

If the p/z pairs are each on the same radius (symmetrical on the unit circle), the frequency response remains flat, but one can design an arbitrary phase/delay response



103

FIR Spec Example

Specifications: $\omega_p = 0.33\pi$,
 $\omega_s = 0.67\pi$, $\alpha_p = 0.01$,
 $\alpha_s = 0.01$

(OK State ECEN 377) Design of Constant-Coefficient Linear-Phase FIR Filter Using the Remez Exchange Technique in MatLab

```
[n,f0,a0,w]=remezord([.33 .67],[1 0],[.01 .01])
```

```
b=remez(n,f0,a0,w);
```

```
[h,w]=freqz(b,1,256);
```

```
hd=20*log10(abs(h));
```

```
zplane(b)
```

```
figure(2)
```

```
plot(w/pi,hd)
```

Output

$n = 10$, $f_0 = [0.33 \ 0.67]$, $a_0 = [1 \ 1 \ 0 \ 0]$, $w = [1 \ 1]$

$b = [0.0248 \ -0.0000 \ -0.0767 \ 0.0000 \ 0.3074$
 $0.5000 \ 0.3074 \ 0.0006 \ -0.76719 \ -0.0046 \ 2.4776]$

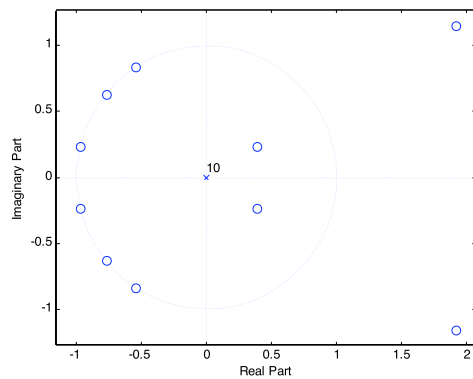
NOTE: Coeff. that appear as zero are small non-zero.

MAT 240B

MAT 240B

104

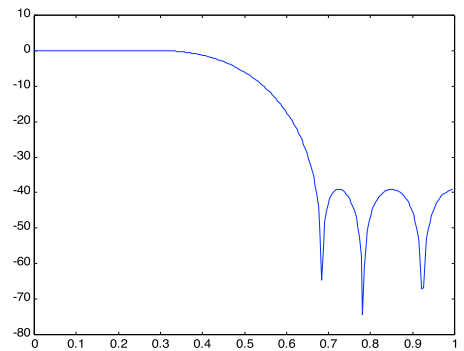
Pole-Zero Diagram for Remez Filter Example



MAT 240B

105

Pole-Zero Diagram for Remez Filter Example



MAT 240B

MAT 240B

106

Form a cascade of second-order sections

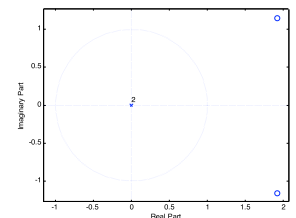
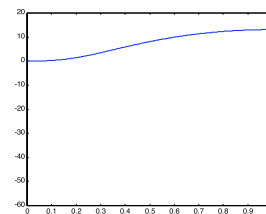
```
z=roots(b);
b1=poly([z(1) z(2)]); [1.0000 -3.8231 4.9915]
k1=b1(1)+b1(2)+b1(3); k1 = 2.1684
b2=poly([z(3) z(4)]); [1.0000 1.9429 1.0000]
k2=b2(1)+b2(2)+b2(3); k2 = 3.9429
b3=poly([z(5) z(6)]); [1.0000 1.5487 1.0000]
k3=b3(1)+b3(2)+b3(3); k3 = 3.5487
b4=poly([z(7) z(8)]); [1.0000 1.0955 1.0000]
k4=b4(1)+b4(2)+b4(3); k4 = 3.0955
b5=poly([z(9) z(10)]); [1.0000 -0.7659 0.2003]
k5=b5(1)+b5(2)+b5(3); k5 = 0.4344
```

MAT 240B

107

Plot of First Section

$$b_1(z) = (1.0000 - 3.8231z + 4.9915z^2) / 2.1684$$



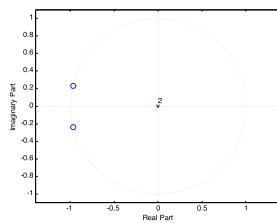
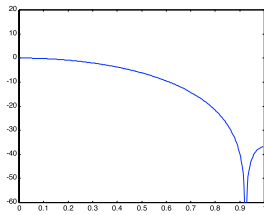
MAT 240B

MAT 240B

108

Plot of Second Section

$$b_2(z) = (1.0000 + 1.9429z^{-1} + 1.0000z^{-2}) / 3.9429$$



```
[h2 w2]=freqz(b2/k2,1,256);
h2d=20*log10(abs(h2));
plot(w2/pi,h2d)
axis([0 1 -60 20])
figure(2)
zplane(b2)
```

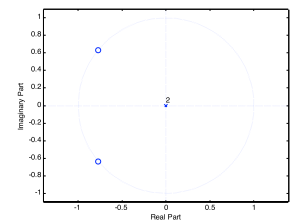
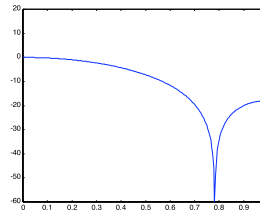
MAT

MAT 240B

109

Plot of Third Section

$$b_3(z) = (1.0000 + 1.5487z^{-1} + 1.0000z^{-2}) / 3.5487$$



```
[h3 w3]=freqz(b3/k3,1,256);
h3d=20*log10(abs(h3));
plot(w3/pi,h3d)
axis([0 1 -60 20])
figure(2)
zplane(b3)
```

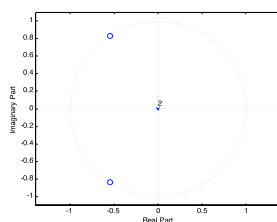
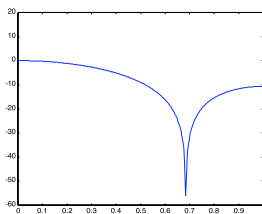
MAT

MAT 240B

110

Plot of Fourth Section

$$b_4(z) = (1.0000 + 1.0955z^{-1} + 1.0000z^{-2}) / 3.0955$$



```
[h4 w4]=freqz(b4/k4,1,256);
h4d=20*log10(abs(h4));
plot(w4/pi,h4d)
axis([0 1 -60 20])
figure(2)
zplane(b4)
```

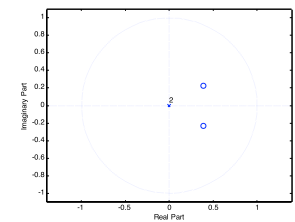
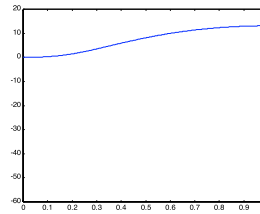
MAT

MAT 240B

111

Plot of Fifth Section

$$b_5(z) = (1.0000 - 0.7659z^{-1} + 0.2003z^{-2}) / 0.4344$$



```
[h5 w5]=freqz(b5/k5,1,256);
h5d=20*log10(abs(h5));
plot(w5/pi,h5d)
axis([0 1 -60 20])
figure(2)
zplane(b5)
```

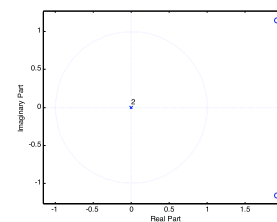
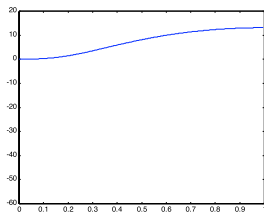
MAT

MAT 240B

112

Cascade of First Section

$$t_1(z) = b_1/k_1$$



```
[ht1 w]=freqz(b1/k1,1,256);
ht1d=20*log10(abs(ht1));
plot(w/pi,ht1d)
axis([0 1 -60 20])
figure(2)
zplane(t1)
```

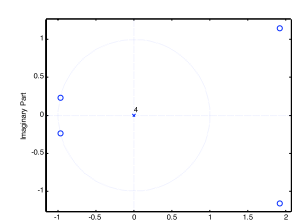
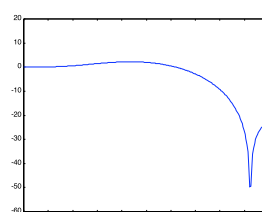
MAT

MAT 240B

113

Cascade of Second Section

$$t_2 = \text{conv}(t_1, b_2) / k_2$$



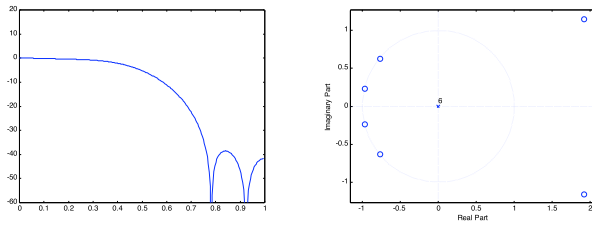
```
[ht2 w]=freqz(t2,1,256);
ht2d=20*log10(abs(ht2));
plot(w/pi,ht2d)
axis([0 1 -60 20])
figure(2)
zplane(t2)
```

MAT

MAT 240B

114

Cascade of Third Section

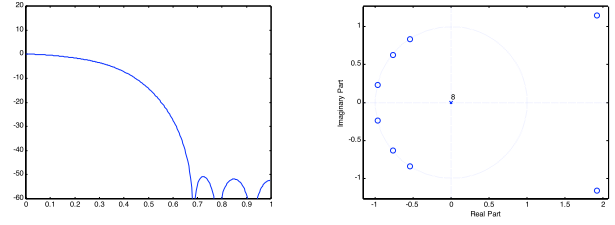
$$t3 = \text{conv}(t2, b3) / k3$$


```
[h3 w]=freqz(t3,1,256);
ht3d=20*log10(abs(h3));
plot(w/pi,ht3d);
axis([0 1 -60 20]);
figure(2)
zplane(t3)
```

MAT 240B

115

Cascade of Fourth Section

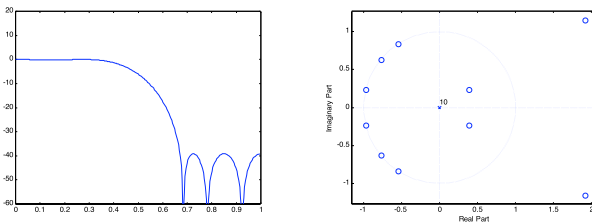
$$t4 = \text{conv}(t3, b4) / k4$$


```
[h4 w]=freqz(t4,1,256);
ht4d=20*log10(abs(h4));
plot(w/pi,ht4d);
axis([0 1 -60 20]);
figure(2)
zplane(t4)
```

MAT 240B

116

Cascade of Fifth Section

$$t5 = \text{conv}(t4, b5) / k5$$


```
[h5 w]=freqz(t5,1,256);
ht5d=20*log10(abs(h5));
plot(w/pi,ht5d);
axis([0 1 -60 20]);
figure(2)
zplane(t5)
```

MAT 240B

117

Examples on the Web

- [<http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt>
- [<http://www.dspguide.com/>
- [<http://www.ensc.sfu.ca/people/faculty/cavers/ENSC380/pzresponse.htm>
- [<http://www.mathworks.com/access/helpdesk/help/toolbox/signal/>
- [http://www.bores.com/courses/intro/iir/5_poles.htm

MAT 240B

MAT 240B

118

Filter Design

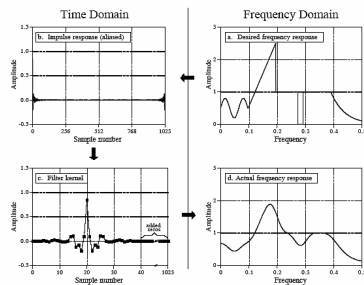


FIGURE 17-1 Example of IIR filter design. Figure (a) shows the desired frequency response, with 513 samples running between 0 to 0.5 of the sampling rate. Taking the inverse DFT results in (b), an allocated impulse response composed of 1024 samples. To form the filter kernel, (c), the allocated impulse response is truncated to 48 samples, shifted to the right by 48 samples, and multiplied by a Hamming or Blackman window. In this example, 48 is 48. The program in Table 17-1 shows how this is done. The filter kernel is formed by padding it with zeros and taking the DFT, providing the actual frequency response of the filter, (d).

MAT 240B

119

Calculating Filter Coefficients

```
/* Calculate common variables used in every filter. */
void calcCommon( double freq, double bandwidth ) {
    // Variables are double inst. Vars.
    // Relative freq (to SR)
    double ratio = freq / Synth.getFrameRate();
    // Radian frequency
    omega = 2.0 * Math.PI * ratio;
    cs = Math.cos( omega ); // sine and cosine
    sn = Math.sin( omega );
    // Q = "steepness"
    Q = sn / (Math.log(2.0) * bandwidth * omega);
    // Alpha = generic coefficient
    alpha = sn * sinh( 0.5 / Q );
}
// From JSyn
```

MAT 240B

MAT 240B

120

Low-Pass Coefficients

```

/* Calculate LowPass filter coefficients. */
/* y[n] = (c0/d0)*x[n] + (c1/d0)*x[n-1] + (c2/d0)*x[n-2] - (d1/d0)*y[n-1] - (d2/d0)*y[n-2] */
void lowPass( double freq, double bandwidth ) {
    calcCommon( freq, bandwidth ); // see above
    double c0 = (1.0 - cs) * 0.5;
    double c1 = (1.0 - cs);
    double c2 = (1.0 - cs) * 0.5;
    double d0 = 1.0 + alpha;
    double d1 = -2.0 * cs;
    double d2 = 1.0 - alpha;
    double temp = 0.5 / d0;
    A0_Value = c0 * temp;
    A1_Value = c1 * temp;
    A2_Value = c2 * temp;
    B1_Value = d1 * temp;
    B2_Value = d2 * temp;
}

```



MAT 240B

121

Band-Pass Coefficients

```

/* Calculate bandPass filter coefficients. */
void bandPass( double freq, double bandwidth ) {
    calcCommon( freq, bandwidth );
    double c0 = alpha;
    double c1 = 0.0;
    double c2 = -alpha;
    double d0 = 1.0 + alpha;
    double d1 = -2.0 * cs;
    double d2 = 1.0 - alpha;
    double temp = 0.5 / d0;
    A0_Value = c0 * temp;
    A1_Value = c1 * temp;
    A2_Value = c2 * temp;
    B1_Value = d1 * temp;
    B2_Value = d2 * temp;
}

```



MAT 240B

122

Models of Filters

- [Time-domain analogies (see above)
 - Simple, don't scale
- [Windows as filter (see part 1)
- [Convolution for filters
 - Process smears the response
- [Vocoder as filter
 - Compute intensive, artifacts



MAT 240B

123

Windows as Filters

- [Window length determines transition band width
- [Pass-band and stop-band ripple determined by window side-lobes (window type)
- [Ripple largest near transition band
- [Narrow main lobe means narrow transition band
- [Tighter specs require longer filters
- [Very long FIR filters can be designed easily, though not usually optimum



MAT 240B

124

Window/Filter Design

- [Ideal filter has sinc as impulse response
- [Window the (infinite) sinc with a finite window
- [Shift the zero-phase filter to get a linear-phase (causal) filter (Phase slope is 1/2 the window length)
- [Apply parameter transformations to make other filter classes



MAT 240B

125

Better Filter Design Techniques

- [A huge (still active) area!
- [Many techniques (numerical, iterative)
- [Many functions in C, Matlab, etc.
- [StateOfTheArt
 - CRemez for optimal FIR with arbitrary magnitude and phase characteristics
 - Second-order Cone Problems (SOCP)
 - Hilbert transform



MAT 240B

126

Coding Examples

- Basic 2nd-order filter functions
 - CSL
 - Jsyn, Csound, mxv, Cmix, etc.
- Filter design packages
 - Mills DAFD
 - Cmix ellipse
 - CARL FIR
 - Signal fir



MAT 240B

127

Convolution



MAT 240B

128

Convolution and Filters

- Convolution
 - Definition, examples
- Time-domain solution (nested loop)
- FFT-based solution - use vocoder, then multiply spectral frames with FFT of (FIR) impulse response
- Same artifacts/problems as vocoder
- Constraints and work-arounds for long convolution
- Trade-offs

$$(f * g)(t) = \int f(\tau)g(t - \tau) d\tau$$



MAT 240B

129

Filter Performance

- Direct convolution is $O(N^2)$
- FFT-based is $O(N \log_2 N)$
- Number of Multiply/Add

N	FFT	Direct Convolution
4	176	16
32	2560	1024
64	5888	4096
128	13,312	16,384
256	29,696	65,536
2048	311,296	4,194,304



MAT 240B

130

Time-varying filters

- Harder...
 - Numerical stability
 - Coefficient interpolation and "zippering"
- Convolution method: need to take FFT of changing filter (i.e., 2 FFTs running at run-time)



MAT 240B

131

Sample Rate Conversion

- Simple interpolation/decimation
- Need good filter to handle artifacts
- Several good-sounding packages exist, but the filters need to be carefully designed



MAT 240B

132

Summary

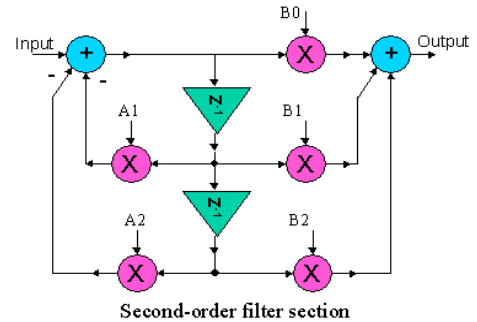
- [Analog and Digital filters
- [Basic FIR & IIR structures
- [Canonical form
- [Coefficient computation
- [Convolution for FIR filters
- [Applications



MAT 240B

133

End of Topic 2

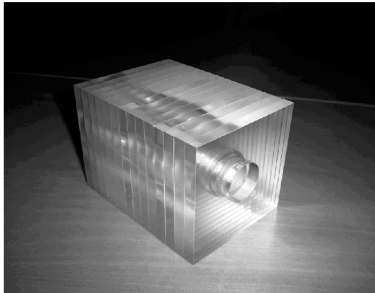


MAT 240B

134

MAT 240B: Digital Audio Programming: Spectral Transformations

- [Topic 3: Linear Prediction



MAT 240B

135

Topic 3: Linear Prediction

- [Voice and Vocal Tract Models
- [LPC model and execution
- [Implementing LPC encoders
- [LPC vocoders
- [Programming LPC



MAT 240B

136

Human Voice Production

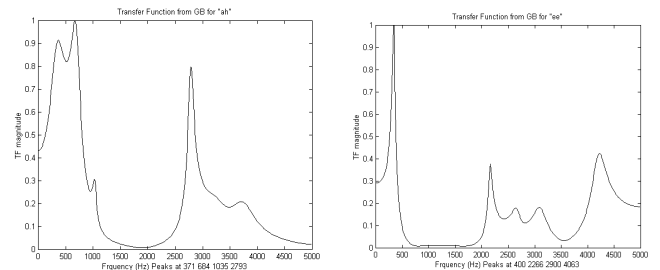
- [Source/Filter Model
- [Glottal pulses or noise and vocal tract
- [Models of the source
- [Models of the Filter



MAT 240B

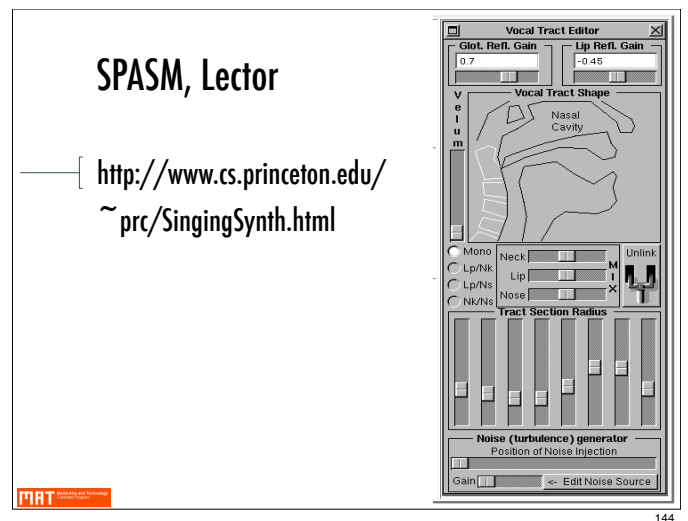
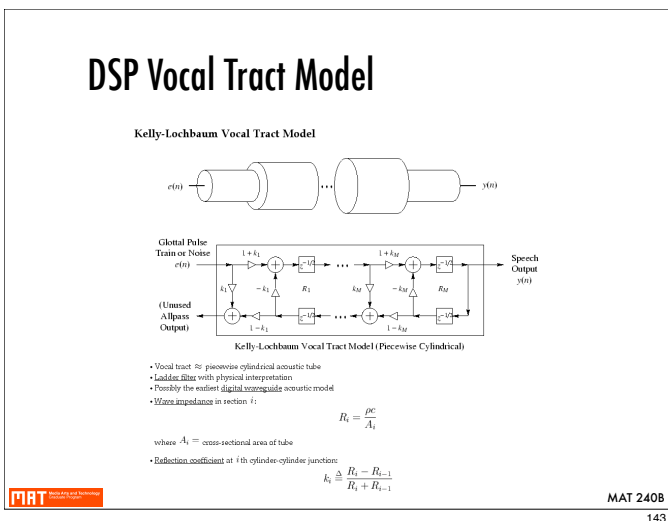
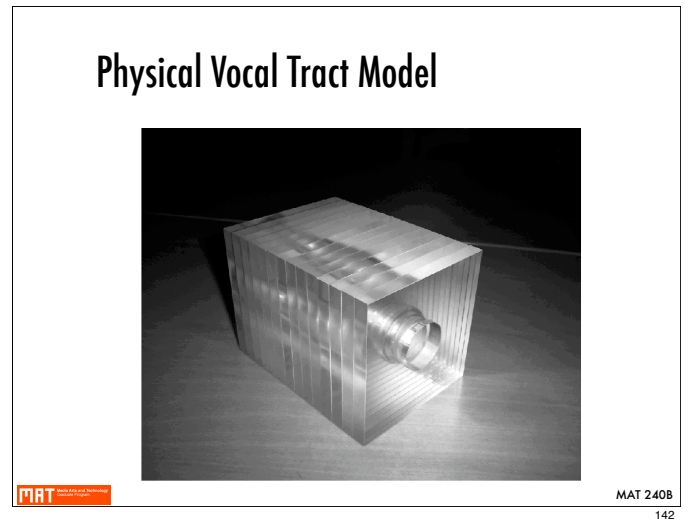
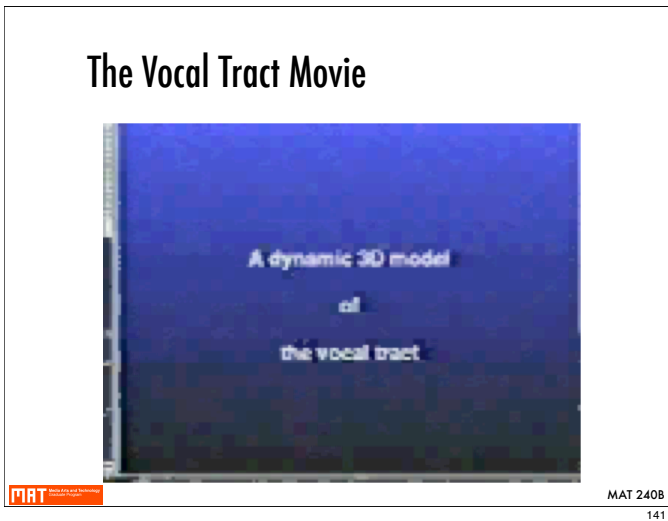
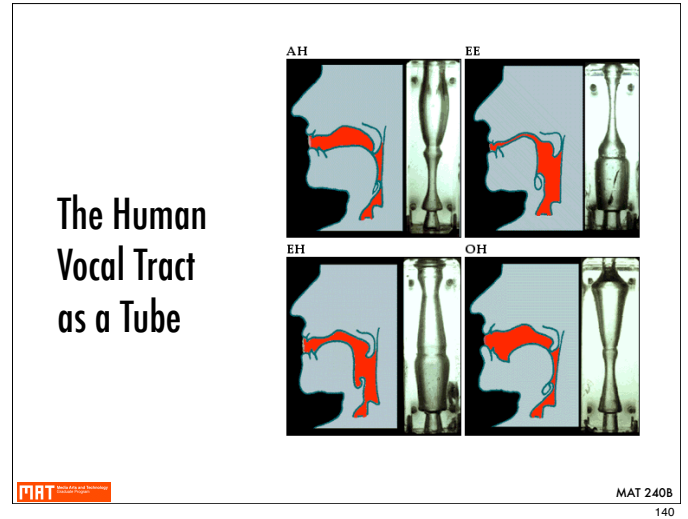
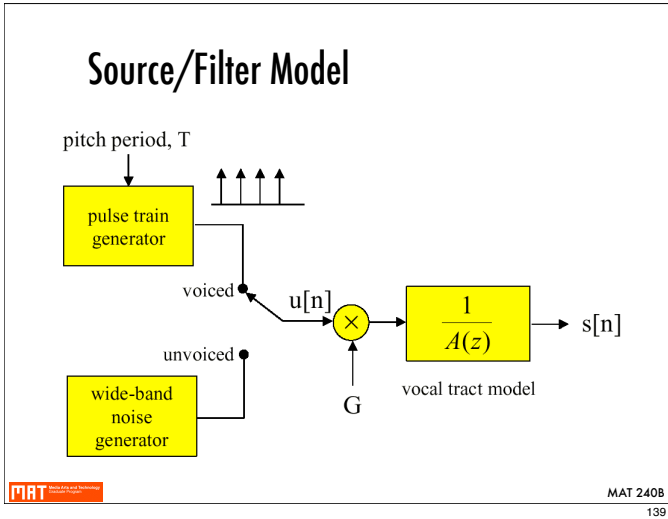
137

Vowel Spectra



MAT 240B

138



The LPC Idea

- Model a sequence of samples with a polynomial
 - $s(n) = a_1s(n-1) + a_2s(n-2) + \dots + a_ps(n-p)$
 - $= \sum_{k=1}^p a_k s(n-k) + \text{error}$
 - p th-order polynomial
- Expect some error and capture it.
- For periodic sounds, the error itself is periodic
- Polynomial can be turned into a filter!



MAT 240B

145

Weights and Error Functions

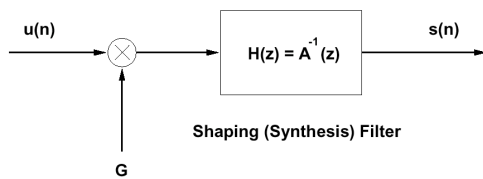
- By a simple transformation, we can view the weights a_k as the coefficients of an all-pole filter
- For a large class of sounds, the error is either (a) noise (unvoiced speech) or (b) an impulse train (voiced speech)
- This leads to a very convenient physical model



MAT 240B

146

LPC Model



- $u(n)$ = excitation function derived from LPC error – may include pitch function
- G = gain
- $H(z)$ = time-varying filter derived from LPC polynomials



MAT 240B

147

So what?

- LPC analysis provides a nice source-filter model (glottal pulses + vocal tract)
- The LPC filter captures the spectral envelope of the signal well
- There is a small family of excitation functions that can be used to model a large class of signals (see CELP)
- LPC is especially applicable to speech
- Both analysis/resynthesis can be implemented efficiently



MAT 240B

148

LPC Analysis

- Two methods for deriving pole weights: autocorrelation and covariance
- Complexity generally scales with window size and number of poles
- Error can be classified into noise or periodic
- Pitch-tracking necessary to exactly determine pitch of excitation



MAT 240B

149

LPC: The Math

- Polynomial $s(n) \approx a_1s(n-1) + a_2s(n-2) + \dots + a_ps(n-p),$
- As summation $s(n) = \sum_{i=1}^p a_i s(n-i) + Gu(n),$
- In Z-domain $S(z) = \sum_{i=1}^p a_i z^{-i} S(z) + GU(z)$
- Error signal $e(n) = s(n) - \tilde{s}(n) = s(n) - \sum_{k=1}^p a_k s(n-k)$
- Transfer function $H(z) = \frac{S(z)}{GU(z)} = \frac{1}{1 - \sum_{i=1}^p a_i z^{-i}} = \frac{1}{A(z)}$



MAT 240B

150

LPC APIs: LibTSP in CSL/FMAK

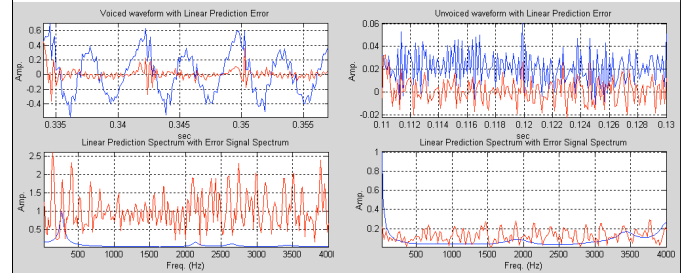
```
// LPC analysis -- get autocorrelation
// void SPautoc (const float x[], int Nx, float cor[], int Nt)
SPautoc (mSampleBuffer, mWindowSize, mCorr.GetRawData( ), mLPCOrder + 1);
// Find predictor coefficients from autocorrelation values
// double SPcorXpc (const float rxx[], float pc[], int Np)
featureTable.mLPCResidual = SPcorXpc (mCorr.GetRawData( ),
mPred.GetRawData( ), mLPCOrder);
// Convert predictor coefficients to filter coefficients
// void SPpcXec (const float pc[], float ec[], int Np)
SPpcXec (mPred.GetRawData( ), mLPCCoeffs, mLPCOrder);
// Convert predictor coefficients to cepstral coefficients
// void SPpcXcep (const float pc[], int Np, float cep[], int Ncep)
SPpcXcep (mPred.GetRawData( ), mLPCOrder, mLPCCoeffs, mCepst);
// Locate the spectral peaks
// locate_peaks(FtVector * fv, FtVector * res, epsilon, spacing);
mPeakExtractor.LocatePeaks( mPred, featureTable.mLPCFormants, 4 );
```

MAT

MAT 240B

151

Examples for Voiced and Unvoiced Speech

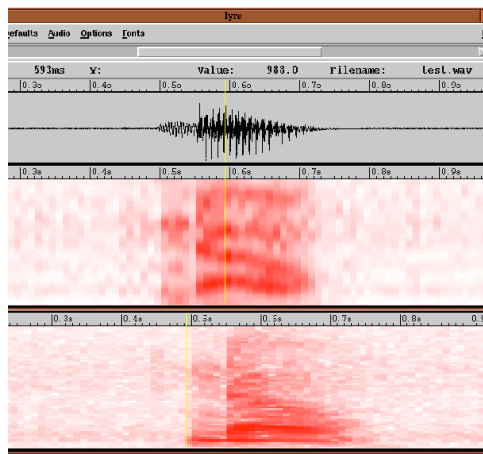


MAT

MAT 240B

152

LPC vs. Power Density Spectrum

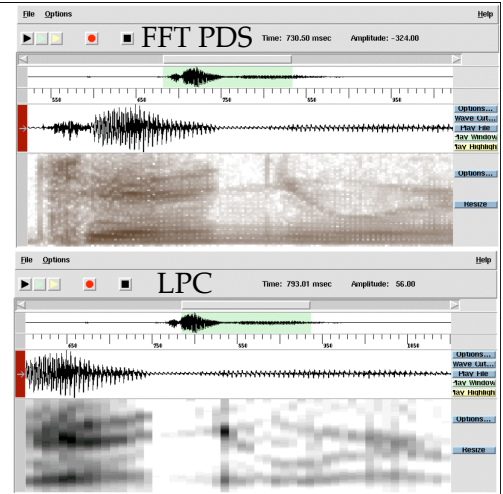


MAT

153

Spectral Analysis Comparison

Pitch/time trade-off
Scalability of the model
Feature extraction

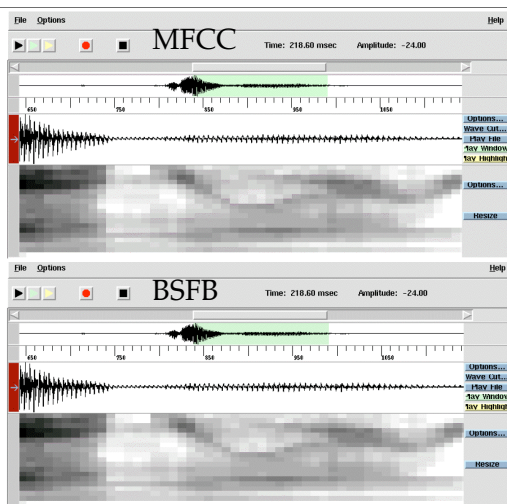


MAT

154

Spectra

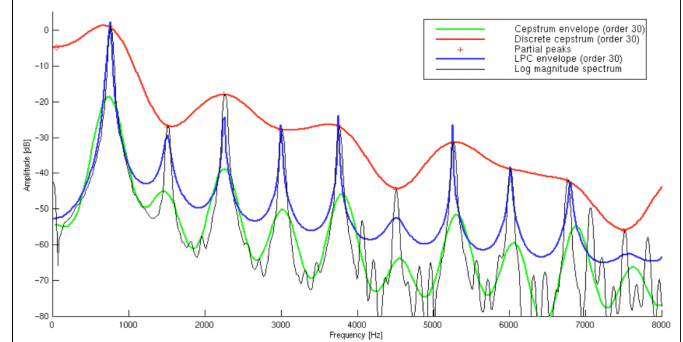
MFCC
triangle
Bark
trapezoid



MAT

155

Smoothed Spectrum Types



MAT

MAT 240B

156

Problems

- [Both methods have potential stability problems because we need to invert the signal covariance matrix, which may be hard if it's nearly singular
- [Resulting filters can have round-off problems (lots of poles very close to the unit circle)
- [It is not clear how to interpolate between filters on resynthesis



MAT 240B

157

LPC Implementations

- [CSL/TSP
- [Csound
- [Cmix
- [Squeak (2 versions)
- [Mxv (with GUI)



MAT 240B

158

LPC in Csound

- [Lpanal program
- [Lpread and lpreson opcodes
- [Lpanal:
 - lpanal [flags] infilename outfilename
 - lpanal performs both lpc and pitch-tracking analysis on a soundfile to produce a time-ordered sequence of frames of control information suitable for Csound resynthesis. Analysis is conditioned by the control flags.



MAT 240B

159

LPAnal Flags

- [-a [alternate storage] – asks lpanal to write a file with filter poles values rather than the usual filter coefficient files.
- [-s srates – sampling rate of the audio input file.
- [-c channel – channel number sought. The default is 1.
- [-b begin – beginning time (in seconds) of the audio segment.
- [-d duration – duration (in seconds) of the audio segment.
- [-p npoles – number of poles – default is 34, max 50.
- [-h hopsize – hop size (in samples) – default is 200, max 500.
- [-C string – text for the comments field of the lpfile header.
- [-P mincps – lowest frequency (in Hz) of pitch tracking.
- [-Q maxcps – highest frequency (in Hz) of pitch tracking. The narrower the pitch range, the more accurate the pitch estimate. The defaults are -P70, -Q200.



MAT 240B

160

LPC Playback in csound

- [Read LPC file into a set of variables
- [krmsr, krmso, kerr, kcps
- [**lpread** ktmpnt, ifilcod [, inpoles [, ifrmrate]]
- [Filter a signal given the most recent lpread
- [or **lpreson** asig



MAT 240B

161

LPC analysis in Cmix

- [lpc [-o lpc_anal_file] [-p #poles]
- [[-f framesize] [-v] [-i inskip]
- [[-d duration] soundfile
- [ptrack [flags] [soundfile]
- [stablize...
- [merge...



MAT 240B

162

LPC Playback in Cmix

- [dataset("sampledataset.lpc", 24)
- [lpcstuff(thresh = .0007, randamp = .1, 0, 0, 0)
- [lpcplay(start=start+incr+1, incr,
transp = 8, frame1, frame2, amp, D, cf, bw)

MAT

MAT 240B

163

Using LPC

- [Source preprocessing (sample rate)
- [Pitch-tracking and associated analysis
- [LPC data storage
- [Compression
- [Cross-synthesis
- [Challenges

MAT

MAT 240B

164

Where to go?

- [Using LPC
- [Porting LPC implementations
- [Stand-alone LPC vocoders for cross-synthesis

MAT

MAT 240B

165

Summary

- [Vocal tract and source-filter models
- [LPC paradigm
- [Implementing LPC
- [Associated analysis
- [LPC for compression, cross-synthesis
- [Applications

MAT

MAT 240B

166

Other Spectral Transformation Techniques

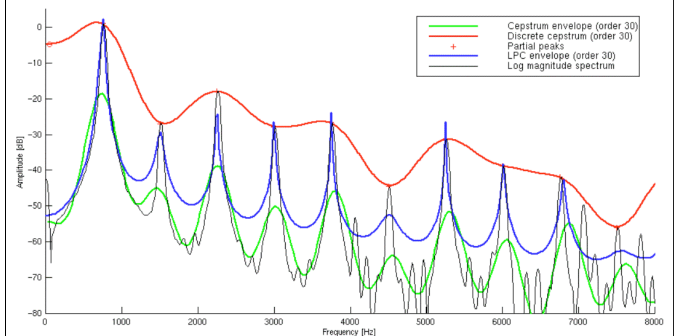
- [Wavelets as spectra
- [Granular processing and hybrid time/frequency-domain representations
- [Vintage filter implementations
- [Spatialization with convolution
- [Convolution-based synthesis
- [See MAT 240B: The Sequel

MAT

MAT 240B

167

MAT 240B: Spectral Transformations



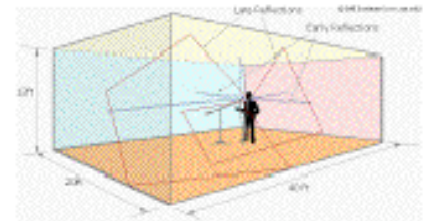
MAT

MAT 240B

168

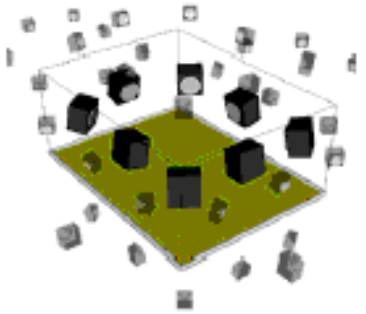
MAT 240C Digital Audio Programming: Spatial and Surround Sound (Spring, 2007)

The MAT 240 sequence is a six-part (two-year) practical programming course; it consists of hands-on software development devoted to digital audio and multimedia applications. Students read a selection of papers from the literature, with the emphasis on learning to use the current state-of-the-art programming methods, tools, and programming interfaces. Class assignments involve C/C++/Java programming on Linux, Macintosh, MS-Windows, various plug-in APIs, and other platforms.



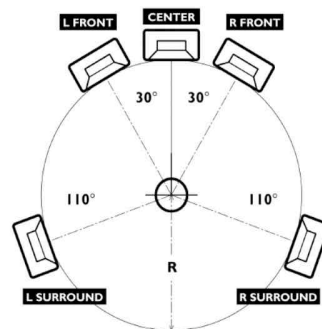
MAT 240C concentrates on the processing of digital audio signals for multi-channel panned or spatialized output. Starting with simple stereophonic models, we investigate the representation of localized sound, reverberation, multi-channel panning, head-related transfer functions, and techniques for producing convincing spatialized sound. We develop and evaluate examples that use several existing surround sound APIs.

Students are expected to know the basics of digital audio signal representation and processing, and to be proficient in C, C++, or Java (Smalltalk, SuperCollider, LISP, and/or XML are a plus). Grading will be on the basis of in-class participation and programming projects.



Course Outline

- Monophonic and stereophonic sound
- Inter-channel panning methods
- Decorrelation and binaural sound
- Sound-in-rooms and reverberation
- Techniques for artificial reverberation
- Standards for multi-channel sound projection
- Simulation of 3D sound localization
- Using head-related transfer function data
- Applications of spatial and surround sound



Instructor

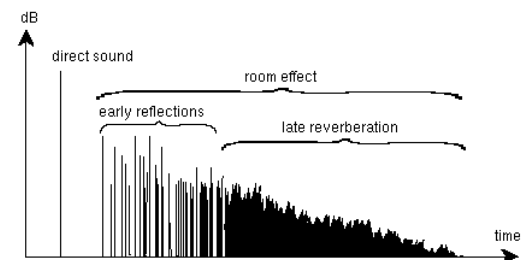
- Stephen T. Pope (stp@mat.ucsb.edu)

Meeting time and place

- T/Th 2:00 - 3:50 PM, Music 2215

Electronic Resources

- Course Web Site
See <http://create.ucsb.edu/240>
- Email Mailing List
See <http://www.mat.ucsb.edu/mailman/listinfo/240> to join



MAT 240C Reader Contents

Topics

- Spatial Sound, Spatial Hearing
- Recording and Representation
- Stereo Processing
- Surround Sound
- Reverberation
- The Head-Related Transfer Function
- Spatialization
- Applications and tThe UCSB CNSI AlloSphere

Spatial Sound, Spatial Hearing

Room Acoustics Modeling, P. Edelbrock, 1996

The Simulation of Moving Sound Sources, J. Chowning, JAES, 1971

Creating Interactive Virtual Acoustic Environments, L. Savioja et al, JAES, 1999

The Future of Audio Technology—Surround and Beyond. JAES 54(9): 2006.

Enhanced Multichannel Audio. JAES 54(6): 2006

CATT-Acoustic Room Acoustics Prediction and Desktop Auralization SW and Related Data Sheets, CATT, Inc., 2000

Direction and Space—the Final Frontiers. F. E. Toole, Harman Int'l. 2002

How many loudspeakers? What kind? Where do we put them? F. E. Toole, Harman Int'l. 2002

Recording and Representation

General Considerations in Audio Multichannel Recording, J. Wuttke, AES, 2001

Forging Ahead with Spatial Audio Coding. JAES 54(12): 2006

Stereophonic Recording Techniques: Old Challenges, New Approaches. JAES 54(3): 2006.

Novel Surround Sound Microphone and Panning Techniques. JAES 52(2): 2004.

Stereo Processing

Multichannel Signal Processing Tools: Differences to Multiple Single Channel Processing, S. Nielsen, AES, 2001

Two-to-Five Channel Sound Processing, R. Irwin and R. Aarts, JAES, 2001

A Balanced Stereo Widening Network for Headphones, O. Kirkeby, AES, 2002

Frequency Domain Techniques for Stereo to Multichannel Upmix, C. Avendano, AES, 2002

Surround Sound

Surround Mixing: History and Techniques, H. Massey, kindoflound.com, 2001

Surround Sound: Past, Present, and Future, Dolby Labs, 1999

Surround Formats, A. Chiarella and M. Polk, Polk Audio, 2003

What is the LFE Channel?, Dolby Labs, 2000

Surround Sound: A Chance for Enhanced Creativity. JAES 54(6): 2006

Reverberation

The Sweet Sound of Reverb. J. Miller, Sweetwater #28
Reverberation, Master Handbook of Acoustics
Efficient Models for Reverberation and Distance Rendering, J-M Jot, ICMC, 1997
Reverb and Room Simulation in the Multichannel Era, K. Christensen, AES, 2001
Efficient Convolution without I/O Delay, W. Gardner, JAES, 1995
Room Acoustic Simulator, N. Morgan, US Patent 4,338,581, 1982
CATT PureVerb and MultiVolver data sheets

Head-Related Transfer Functions

Overview of the Head-Related Transfer Functions (HRTFs), W. Yang, Penn, 2001
Binaural Audio in the era of Virtual Reality. JAES 51(11): 2003
HRTF Measurements of a KEMAR Dummy-Head Microphone, B. Gardner and K. Martin, MIT Media Lab, 1994
The CIPIC HRTF Database, V. R. Algazi., Mohonk, 2001
Wavelet-based Spectral Smoothing For Head-related Transfer Function Filter Design, H. Hacıhabiboglu, B., and A. Fionn, AES, 2002

Spatialization

Synthesizing 3-D Sound Scenes..., J-M Jot, 1996
Real-Time Audio Spatialization with Inexpensive Hardware, D. A. Burgess, GATech, 1992
A Beam-Tracing Approach to Acoustic Modeling, T. Funkhouser et al., Bell Labs, undated
Virtual Sound Source Positioning Using VBAP, V. Pulkki, JAES 45(6): 1997
Basic Ambisonics. D. G. Malham, U. York, 1995.
Wavefield Synthesis—A Promising Spatial Audio Rendering Concept. G. Thiele, DAFX'04
Wavefield Synthesis—Generation and Reproduction of Natural Sound Environments. T. Sporer, DAFX'04

Applications and The AlloSphere

Audio in the UCSB CNSI AlloSphere. S. T. Pope, CREATE, 2005
Zirkonium Manual. C. Ramakrishnan, ZKM, 2007
Ventriloquist Presentation Slides. D. McCoy, MAT
Applications of WFS in Electronic Music and Sound Art. M. Baalman IDEA #7.
Updates of the WONDER SW Interface using WFS. M. Baalman LAC 2005.

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT Media Arts and Technology Graduate Program

University of California Santa Barbara

MAT 240C: Digital Audio Programming: Spatial and Surround Sound

Prepared by Stephen Travis Pope
UCSB, Spring, 2007 (stp@create.ucsb.edu)

MAT 240C

1

MAT 240C Topics

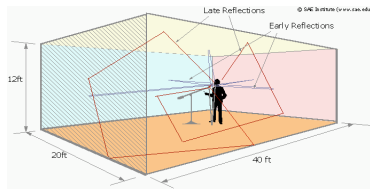
- 1: Spatial Sound, Spatial Hearing
- 2: Recording and Representation
- 3: Matrix Playback Processors
- 4: Reverberation and Convolution
- 5: Head-Related Transfer Functions
- 6: Spatialization
 - VBAP, Ambisonics, WFS, systems and control
- 7: Applications and the UCSB AlloSphere

MEDIA ARTS & TECHNOLOGY PROGRAM**MAT 240C**

2

Lecture 1 Outline

- Goals
- Logistics
- Materials
- Course Overview
- Projects
- Topic 1

**MAT 240C**

3

Goals for MAT 240C

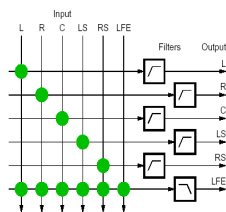
- The MAT 240 series
 - Background in programming
 - Background in psychoacoustics
- MAT 240C Topics
 - Spatial Sound, Spatial Hearing
 - Recording, Representation, Processing, Playback
 - Reverberation, Convolution, HRTFs
 - Spatialization

MEDIA ARTS & TECHNOLOGY PROGRAM**MAT 240C**

4

Logistics

- Instructor
 - Stephen T. Pope (stp@mat.ucsb.edu)
- TA
 - Eric Newman (ericnewman@umail.ucsb.edu)
- Meeting time
 - Tuesday/Thursday 2:00 - 3:50 PM
 - CREATE class room, Music 2215
- Grading
 - Attendance and class participation
 - 3-4 Programming projects
 - At least 1 "stand-alone"
 - At least 1 contribution to a group project

**MAT 240C**

5

Materials

- Reader
 - Comprehensive
 - Available at UCSB book store
 - TOC on-line
 - Important to course content
 - Comments or additions are welcome!
- Presentation Slides
 - Available at UCSB book store and on web site

**MEDIA ARTS & TECHNOLOGY PROGRAM****MAT 240C**

6

Other Materials

- Web site: <http://create.ucsb.edu/240C>
(Comments/corrections/additions welcome!)
- Mailing list: 240@mat.ucsb.edu,
<http://zydeco.mat.ucsb.edu/mailman/listinfo/240>
- Software: (Code ZIP file) mixer/panners,
reverberators, auralizer, HRTF, etc.
- CSL web site <http://create.ucsb.edu/CSL>

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

7

Course Schedule

- **Week 1: Introduction, history, overview**
 - Spatial Hearing and psychoacoustics
 - Formats for multichannel sound
- **Week 2: Multichannel sound processing**
 - Recording and mixing techniques for spatial sound
 - Converting between formats, stereo matrices
 - Surround sound formats and home theater
- **Week 3-4: Reverberation**
 - Reverberation, room models, and spaciousness
 - Software for artificial reverberation, time-domain, convolution-based, low-latency

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

8

Course Schedule

- **Week 5: Head-Related Transfer Functions**
 - The HRTF and its use
 - HRTF databases and applications
- **Week 6-7: 3-D spatial sound**
 - 3-D sound for interactive systems, “auralizers” and VE systems
 - VBAP, ambisonics, and wavefield synthesis
 - Spatialization systems and control
- **Week 8-9: Applications**
 - Real-time interactive systems
 - Integration in the UCSB AlloSphere

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

9

Reader in Detail

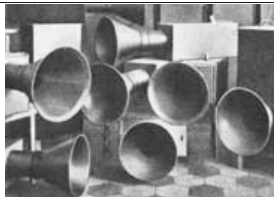
- **Spatial Sound, Spatial Hearing**
 - Room Acoustics Modeling
 - The Simulation of Moving Sound Sources
 - Creating Interactive Virtual Acoustic Environments
 - Future of Audio Technology: Surround
 - CATT Acoustics Data
 - F. Toole white papers
- **Recording and Representation**
 - General Considerations in Audio Multichannel Recording
 - Forging Ahead with Spatial Audio Coding
 - Novel Surround Sound Mic/Panning Techniques

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

10

Reader in Detail



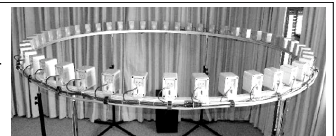
- **Stereo Processing**
 - Multichannel Signal Processing Tools: Differences to Multiple Single Channel Processing
 - A Method to Convert Stereo to Multi-channel Sound
 - A Balanced Stereo Widening Network for Headphones
 - Freq. Domain Techniques for Stereo to Multichannel Upmix
- **Surround Sound**
 - Surround Mixing: History and Techniques
 - Surround Sound: Past, Present, and Future
 - Surround Formats, What is the LFE Channel?

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

11

Reader in Detail



- **Reverberation**
 - Sweet Sound of Reverb
 - Reverberation, Master Handbook of Acoustics
 - Efficient Models for Reverberation and Distance Rendering
 - Reverb and Room Simulation in the Multichannel Era
 - Efficient Convolution without I/O Delay
 - Room Acoustic Simulator
- **Head-Related Transfer Functions**
 - Overview of the Head-Related Transfer Functions (HRTFs)
 - HRTF Measurements of a Dummy-Head Microphone
 - The CIPIC HRTF Database
 - Wavelet-based Spectral Smoothing For HRTF Filter Design

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

12

Reader in Detail

- **Spatialization**
 - Synthesizing 3-D Sound Scenes...
 - Real-Time Audio Spatialization with Inexpensive Hardware
 - A Beam-Tracing Approach to Acoustic Modeling
 - VBAP Paper
 - Ambisonics Intro
 - WFS Descriptions
- **Applications and the AlloSphere**
 - AlloSphere Audio Design
 - Wonder, Ventriloquist, Zirkonium

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

13

Projects

- **Small stand-alone projects**
 - As in previous 240 courses
- **Group projects**
 - Real-time convolution-based reverberator (as stand-alone program and/or as a VST plug-in)
 - CSL Spatialization framework apps
 - “HRTF player”
 - VR “auralizer”
 - Working in the CNSI spaces

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

14

What's Next?

- **Topic 1 Readings**
 - Hearing physiology and psychology
 - Room acoustics
 - Spatial sound overviews
 - Historical formats
 - Playback systems

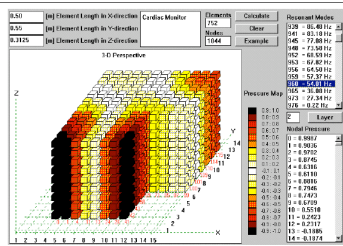


Figure 8. Pressure distribution of the 54th mode

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

15

MAT 240C Digital Audio Programming: Spatial and Surround Sound Topic 1: Overview, History

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

16

Topic 1: Overview, History

- **Spatial sound overview**
 - Room acoustics
 - Hearing perception
 - Spatial hearing
- **Some history**
 - Spatial sound recording
 - Reproduction formats
 - Playback systems

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

17

Topic 1 Readings

- **Spatial Sound, Spatial Hearing**
 - Room Acoustics Modeling
 - The Simulation of Moving Sound Sources
 - Creating Interactive Virtual Acoustic Environments
 - Future of Audio Technology: Surround
 - CATT Acoustics Data
 - F. Toole white papers

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

18

Spatial Audio



- Helps is in everyday life
 - Hear the radio over the road noise in the car
 - Disambiguate similar sources (“cocktail party effect”)
- Spatial hearing has different properties than spatial vision
 - We can track multiple sources at the same time
 - We can localize sources in all directions
 - We can mask/unmask very quickly
 - There are many levels of sound semantics

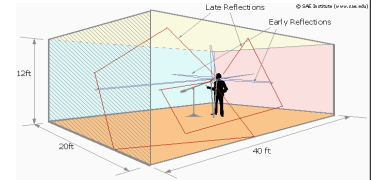
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

19

Sound in Space and Spatial Hearing

- Acoustics
 - Sound generation, projection, and radiation
 - Sound in enclosed spaces
 - Room acoustics
- Psychoacoustics
 - Mechanisms of hearing
 - Binaural hearing
 - Spatialization cues



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

20

Sound Generation, Projection, and Radiation

- Acoustical sources
 - Excite air via resonance or expulsion
 - Are generally highly *anisotropic* in both amplitude and spectrum; radiation pattern is a 4-D function
 - Generate time-varying spectra
- Properties of radiating surfaces and volumes
 - Volumetric radiation pattern model
 - Wavefront radiation model

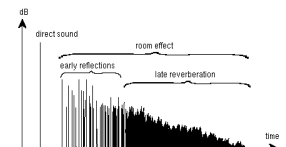
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

21

Sound in Enclosed Spaces

- Reflections (reverberation)
 - Early reflections tell us about the geometry of the room and the relative positions of the source and listener
 - Late reflections (reverb tail) tell us about the size and materials of the enclosing space
- Characteristics of surfaces
 - Reflection
 - Absorption
 - Transmission
 - Diffusion



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

22

Room Acoustics

- Acoustical features of “rooms”
 - Resonances, standing waves
 - Calculate “cavity modes” just as if the room were an organ pipe (below about 200 Hz)
 - Frequency-dependent decay characteristics
 - Coupling within multi-room structures
- Issues
 - T60 time (time for energy to die to -60 dB)
 - Mean free path

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

23

Room Resonances

- Calculating room resonant modes based on room geometry

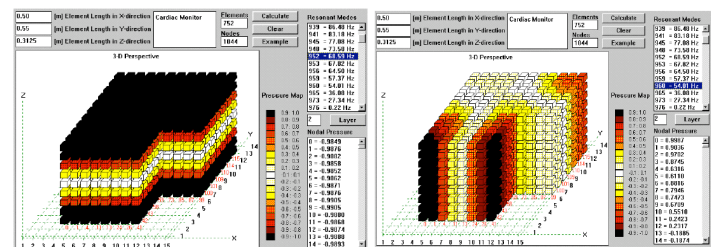


Figure 9. Pressure distribution of the 60Hz mode.

Figure 8. Pressure distribution of the 54Hz mode.

MEDIA ARTS & TECHNOLOGY PROGRAM

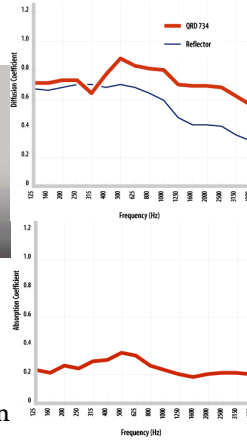
MAT 240C

24

Material Properties



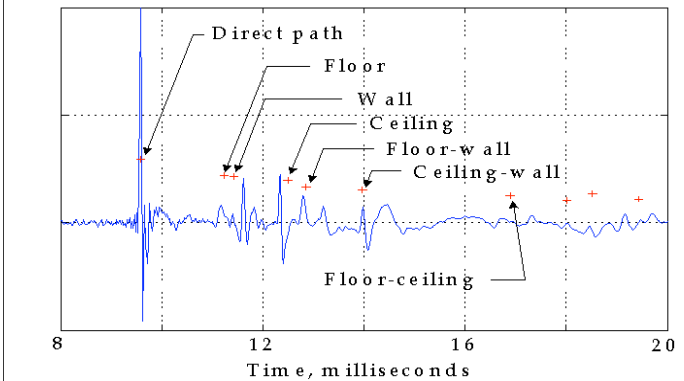
- Reflection
- Absorption
- Transmission
- Diffusion
- All as functions of frequency



MAT 240C

25

Calculated vs. Measured Impulse Responses



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

26

Comments on Rooms

- Optimum reverberation time is a compromise between clarity (short reverberation time), sound intensity (high reverberant level), and liveness (long reverberation time).
- The optimum reverberation time of an auditorium is dependent on the use for which it is designed.
- Reflected sound arriving from the sides seems to be important to the overall “reverberance” of the room.

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

27

Comments on Rooms, 2

- Important subjective attributes of concert hall acoustics include *intimacy, liveness, warmth, loudness of direct sound, reverberant sound level, definition or clarity, diffusion or uniformity, balance and blend, ensemble, and freedom from noise.*
- *Spatial impression* and *early decay time* are important. The spatial impression is dependent on the early reflections from the sides and above. The initial rate of decay of reverberation is more important than the total reverberation time.

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

28

Comments on Rooms, 3

- Echoes, flutter echoes, sound focusing, sound shadows, and background noise should be avoided in an auditorium design.
- The greater the early decay time (<2 sec), the greater the preference for the concert hall. Above 2 sec, the trend is reversed.
- Narrow halls are generally preferred to wide ones.
- Preference is shown for halls having high “binaural dissimilarity”
- Less “definition” is preferred. Definition represents the ratio of energy in the first 50 milliseconds to the total energy.

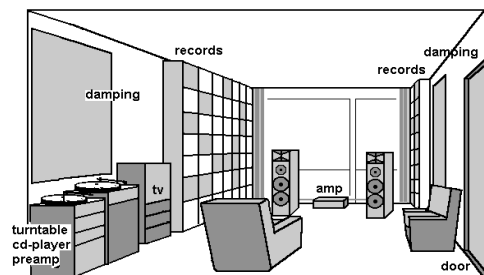
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

29

Room Treatment

- A whole topic on its own; see MAT 242A



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

30

Psychoacoustics

- Physiology of hearing
 - Anatomy of the ear
 - Neurology and the spiral ganglia
- How do we process what we hear?
 - Basic frequency-domain representation
 - Transients and noise
 - Binaural effects and spatial hearing
- Hearing and cognition

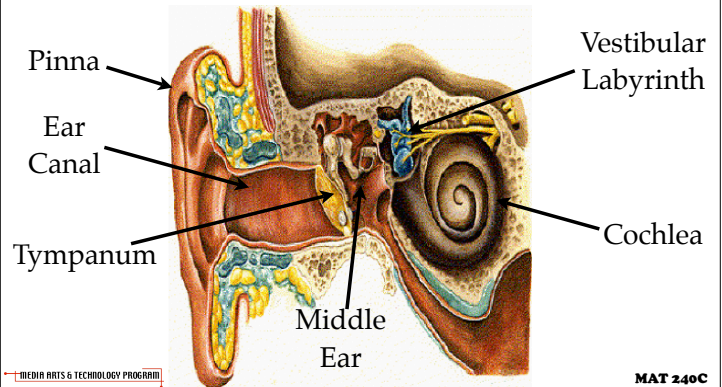
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

31

Mechanisms of Hearing

- Three-stage ear



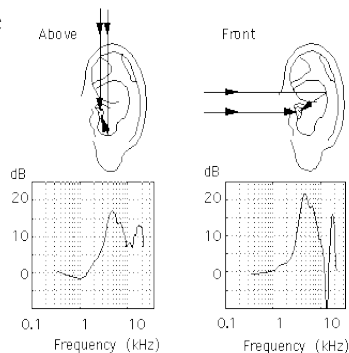
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

32

Pinna Shape and Reflections

- Individual differences make a huge difference



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

33

Hearing

- Frequency range: $\sim 1000:1$ (~ 10 octaves)
- Amplitude range: ~ 32 trillion:1 (!) (~ 140 dB)
- Some time-domain and some frequency-domain processing
- Different programs for:
 - Speech
 - Music
 - Environmental sound

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

34

Binaural Hearing

- Interaural differences (stereo decorrelation)
 - Arrival time = IATD
 - Base amplitude = IAAD
 - Early reflections & spectrum = HRTF
 - NB: also monaural
- These give us the basic spatialization cues:
 - Amplitude
 - Time/phase
 - Spectral

MEDIA ARTS & TECHNOLOGY PROGRAM

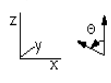
MAT 240C

35

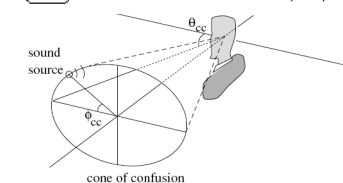
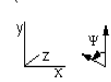
Geometry & Spatialization Cues

Geometry

(seen from the top)

Source
Filter (medium)
Listener

(seen from the back)



Spatial Cues

Distance (d) = $(b + a) / 2$
 Loudness $\propto d^2$
 L/R ratio $\propto (b - a) / e$
 Low-pass filter $\propto d^2$
 Direct/reverb ratio $\propto d^2$
 Spatial low-pass filter ratio $\propto \theta$
 Spatial band-reject filter $\propto \psi$
 Inter-aural time delay $\propto (a - b) / e$
 Initial/decay reverb ratio $\propto \psi$
 (many more possible)

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

36

Importance of Spatial Cues

	Orchestra elements	Soloists	Room	Audience
Horizontal direction	● ● ●	● ● ●	●	● ●
Elevation	○	○		○
Near-head distance				○ ○ ○
Distance, spatial depth	● ● ●	● ● ●		● ●
Spatial impression			● ● ●	●
Envelopment			● ●	● ● ●
Sound colour	● ● ●	● ● ●	● ● ●	● ●

MAT 240C

37

Relations between Cues

	Direct sound	Early reflections	Reverberation	Environmental non-reflected sound
Horizontal direction	● ●	●		
Elevation	○ ○	○		
Near-head distance	○ ○			
Distance, spatial depth		● ●	●	
Spatial impression		● ●	● ●	
Envelopment			● ●	● ●
Sound colour	● ●	●	● ●	

MAT 240C

38

Topic 1B Readings

- Recording and Representation
 - General Considerations in Audio Multichannel Recording
 - Forging Ahead with Spatial Audio Coding
 - Novel Surround Sound Mic/Panning Techniques

MAT 240C

39

A History of Spatial Music

- Venice, 1550s (A. Gabrieli, Willaert, Tallis): spatially separated instrumental groups (vocalists, organ pipes), “dialog form” and echo effects
- Romantics: Berlioz, Verdi, Mahler, Ives
- Early electromechanical recording and reproduction: monophonic
- 1881: Clement Adler’s 80-channel spatial telephone at the Paris Exhibition
- 1950s: Stereo (various versions)
- 1970s: Quad (2 versions)
- 1990s: “Surround sound” (many versions)



MAT 240C

40

Formats for Multichannel Sound

- Monophonic sound
 - Analog pressure wave from point-receiver microphone
 - Wavefront microphone
 - Microphone in ear canal
- Encoding of spatial information
 - The signal-per-channel assumption
- Stereophony
- Quad encoding
- Pluriphonic representations
- N.M-channel theater sound

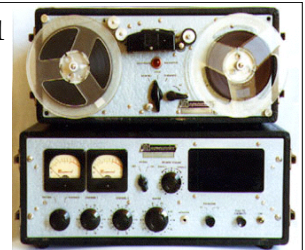
MAT 240C

41

Stereophony

- Left/right channels
 - Binaural stereo
 - 45° stereo
 - M/S stereo
 - Headphone stereo
- Handling crosstalk

1951



1954 (NB: 2 heads!)

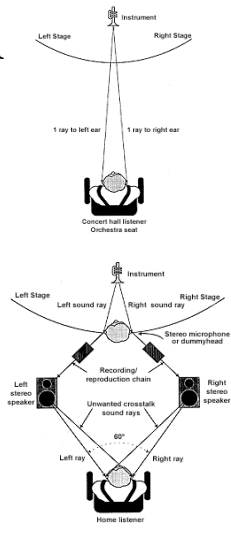


MAT 240C

42

Stereo Decorrelation

- Problem: how to model stereo (mic placement, speaker placement, assumptions about listening room) to maintain the stereo image?
- Challenge: decorrelation of the stereo channels and cross-talk at playback

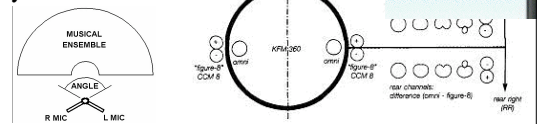


43

Binaural Recording

- Decorrelated channels
- Recorded via dummy head or head-mounted microphones
- Intended for headphone playback
- Binaural synthesis?

Derivation of right front / right rear signals (rough)



44

Other Flavors of Stereo

- E.g., 3-Channel LCR Stereo (1958)



MAT 240C

45

Quad Encoding

- Speakers: corners vs. sides
- 1970s standards
 - CD-4 & SQ Quad on LPs
 - Quad FM broadcast
 - Quad on 4-channel (r2r and 8-track) tapes
 - "Stereo-to-quad" matrix decoding (cheesy)
- The content issue
 - HW developed or licensed by content providers

MAT 240C

46

Quad Formats & Content

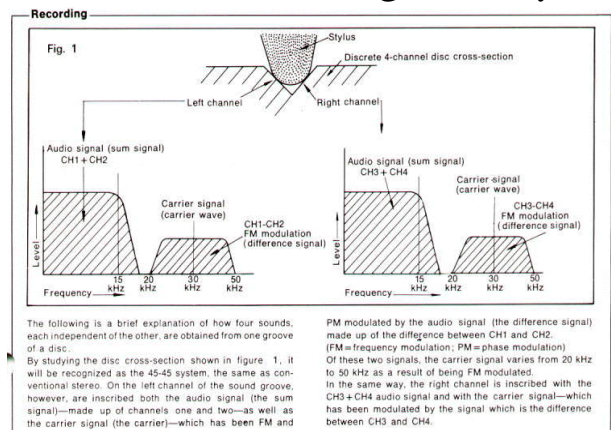
1970s

- **SQ** - Developed by CBS; adopted by Capitol, CBS, CTI, Columbia, EMI, Epic, Vanguard
- **CD-4** (Quadradisc) - JVC/RCA; Arista, Atlantic, Elektra, Fantasy, JVC, Nonesuch, RCA, Reprise
- **QS** - Sansui; ABC, Advent, Decca, MCA, Vox
- **EV (Stereo-4)** - Electro Voice
- **DY (Dynaquad)** - Dynaco
- **UD-4** - Nippon/Columbia



47

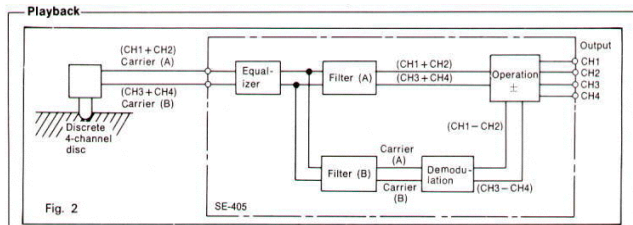
CD-4 Quad Recording on Vinyl LPs



MAT 240C

48

CD-4 Quad Playback on Vinyl LPs



The block diagram will serve to explain how four independent sounds are picked up from the discrete 4-channel disc. First, both the sum signal of CH1+CH2 and the carrier signal (which has been modulated by the difference signal of CH1-CH2) are picked up from the left channel of the disc by the CD-4 cartridge and are sent to the SE-405 unit, where they enter, without change, filter A and filter B after passing through the equalizer (RIAA curve). These filters separate the signals into (1) the sum signal of CH1+CH2 and (2) the modulated carrier signal. The sum signal enters the operation circuitry without change, while the modulated carrier signal passes through the demodulation circuitry

and enters the operation circuitry, becoming a difference signal CH1-CH2 (which is a modulated wave). Then addition and subtraction are accomplished by the operation circuitry.

Specifically:

Addition: $(CH1 + CH2) + (CH1 - CH2) = 2(CH1)$

Subtraction: $(CH1 + CH2) - (CH1 - CH2) = 2(CH2)$

By this process, CH1 and CH2 are sent to the output. In the same way, the signal picked up from the right channel of the disc is sent out as CH3 and CH4. This explanation has, we hope, served to explain how four signals are picked up from one groove of a disc as completely independent sounds.

MAT 240C

49

Quad Hardware

- CD-4 stylus and decoder

4DD-5

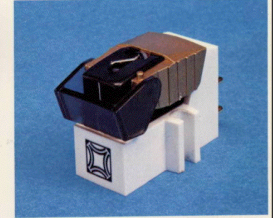
New CD-4 Disc Demodulator For Discrete 4-Channel Records
Now you can enjoy the full realism of 4-channel stereo sound available through CD-4 record system with this new CD-4 demodulator. The 4DD-5 is compact and has a wide frequency response so that you can enjoy the highest of the highs and the lowest of the lows. Full-range 4-channel sound!



SPECIFICATIONS	
Type	CD-4 disc demodulator
Semiconductors	6BCs, 6FETs, 27transistors and 23 diodes
Frequency response	20Hz to 16,000Hz
Input sensitivity	1.5mV
Output level	300mV
Input impedance	100 kΩ
Output impedance	5 kΩ
Signal to noise ratio	60dB
Input jacks	2
Output jacks	4 for 4-channel 2 for 2-channel
Power consumption	8W
Dimensions	3.38" (H) x 6.78" (W) x 13.3/4" (D)
Weight	5 lbs. (excluding package)

4MD-20X

Unique 4-Channel/2-Channel Stereo Cartridge With SHIBATA Stylus
The 4MD-20X V-shape magnetic cartridge has an extended range frequency response (20 – 60,000Hz) which is as much as three times wider than a regular stereo cartridge. It means this cartridge can reproduce the perfect discrete 4-channel sound.



SPECIFICATIONS	
Application	Playback of CD-4 discrete 4-channel or 2-channel discs
Structure	Magnetic
Stylus	Shibata Stylus, 40T-20X
Output	2.0mV at 1kHz and 50mm/s
Output Balance	0.5dB
Frequency Characteristics	20 – 60,000Hz
Channel Separation	More than 30dB (at 1kHz)
Impedance	2.5kΩ (at 1kHz)
DC Resistance	100Ω
Stylus Pressure	1.5 – 2.0 gr.
Compliance	35 x 10 ⁻⁶ cm/dyne
Load	47kΩ – 100kΩ

* Design and specifications subject to change without notice

50

Quad Hardware, 2

- Decoders and Receivers

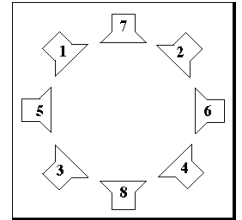


MAT 240C

51

Pluriphonic Representations

- 8-channel octophony
- 8-channel 3-D sound
- Pluriphonic systems
 - Symmetrical (US) systems
 - Asymmetrical (European) systems
- No standards for mapping sound output channels to speakers

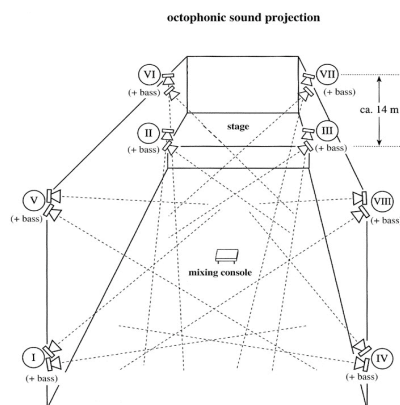


MAT 240C

52

Octophonic Setup

- Circular layout vs. "corners of the cube"



MAT 240C

53

Loudspeaker Orchestras

- MISO Lisbon,
- GRM Paris,
- BEAST Birmingham



54

UCSB Creatophone (1999)



- 15 speakers, 9 channels for frontal stereo image

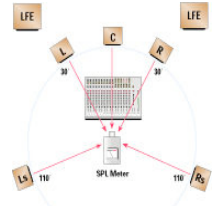
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

55

N.M-channel Theater Sound

- 5.1-channel LCR frontal image
 - 30° front separation
 - Unclear center channel signal
- Side and/or rear surround pairs
 - 110° "rear" separation
 - Possible true-rear pair or rear-center channel (6.1-channel)
- LFE channel(s)
 - Cross-over freq. Unclear
- Discussion of 5.1 to follow



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

56

Review

- Spatial sound overview
 - Room acoustics
 - Hearing perception
- Some history
 - Spatial sound recording
 - Reproduction formats

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

57

What's Next?

- Recording, synthesizing, processing, and mixing multi-channel sound
- Topic 2 readings
- Inter-format conversion
- Getting started programming!

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

58

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

Digital Audio Programming:

Spatial and Surround Sound

Topic 2: Multichannel Sound

Processing

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

59

Topic 2

- Multichannel sound processing
 - Recording and mixing techniques
 - Close-miced and distance-miced recording
 - Standard formats
 - Novel placements and mic designs
 - Relevance of spatial features
 - Processing multichannel sound
 - Detectors
 - Parallel processing
 - Challenges

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

60

Topic 2 Readings

• Stereo Processing

- Multichannel Signal Processing Tools: Differences to Multiple Single Channel Processing
- A Method to Convert Stereo to Multi-channel Sound
- A Balanced Stereo Widening Network for Headphones
- Freq. Domain Techniques for Stereo to Multichannel Upmix

Readings

• Recording and Representation

- General Considerations in Audio Multichannel Recording
- Natural 5.1 Music Recording Based on Psychoacoustic Principles

• Processing Spatial Sound

- Multichannel Signal Processing Tools: Differences to Multiple Single Channel Processing

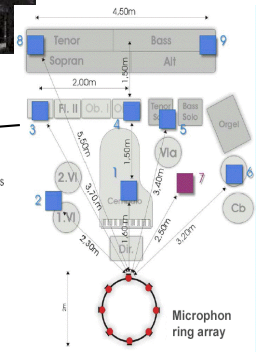
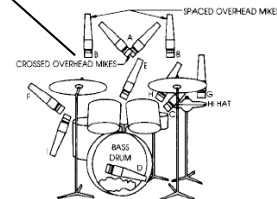
• Web references on strange mics and spatial theories

Microphone Placement

- Early: single mic in a “live” concert hall
 - 1-step recording, hall ambience caught on tape
 - Still used for many “audiophile” recordings
- Recent: multiple (or many) close mics in a “dead” studio
 - Create the spatial image in the (stereo) mix using inter-channel panning, reverb, and equalization
- Special mic placement for various instruments = the recording engineer’s skill
- Recording the source vs. recording the room
 - Choose 1 extreme or the other

Microphone Placement

- Old
- Recent
- New








Recording Spatial Sound

- Recording techniques that capture the spatial aspects of sound for multichannel playback
 - Stereo pairs
 - Binaural recording
 - Close-micing and mixing techniques
 - Recording spatiality: wavefront, ambisonics, ambiophonics, etc.

Micing Live Unamplified Ensembles

STEREO PICKUP SYSTEMS	MICROPHONE TYPES	MICROPHONE POSITIONS	
X-Y	2 - CARDIOID	AXES OF MAXIMUM RESPONSE AT 135° SPACING: COINCIDENT	
ORTF (FRENCH BROADCASTING ORGANIZATION)	2 - CARDIOID	AXES OF MAXIMUM RESPONSE AT 110° SPACING: NEAR-COINCIDENT (7 IN.)	
NOS (DUTCH BROADCASTING FOUNDATION)	2 - CARDIOID	AXES OF MAXIMUM RESPONSE AT 90° SPACING: NEAR-COINCIDENT (12 IN.)	
STEREOSONIC	2 - BIDIRECTIONAL	AXES OF MAXIMUM RESPONSE AT 90° SPACING: COINCIDENT	
MS (MID-SIDE)	1 - CARDIOID 1 - BIDIRECTIONAL	CARDIOID FORWARD-POINTED; BIDIRECTIONAL SIDE-POINTED; SPACING: COINCIDENT	
SPACED	2 - CARDIOID OR 2 - OMNIDIRECTIONAL	ANGLE AS DESIRED SPACING: 3-10 FT.	

Another View of the Standards

Stereo recording principle	coincident microphone placement		level differences + minor arrival-time differences	microphones separated by an acoustically opaque object	major arrival-time differences
Name	XY	M/S	ORTF (for example)	Jacklin disk (for ex.)	A/B
Geometry					
Distance (d) between microphones	0 cm usually vertically aligned		5 to 30 cm ↑ interdependent	depends on the object between them	40 to 80 cm or greater (up to several meters)
Angle between the main axes of the microphones	45° to 180°	90°	0° to 180°	typically 20°	0° to 90°
Acoustic operating principle of the microphones:	pressure gradient transducers, e.g. Schoeps MK 4 cardioid			usually pressure transducers, e.g. Schoeps MK 2 C omnidirectional*	
Sonic impression	--- depending on the microphones used --- clear, clear, often bright				
Spaciousness	often rather limited	satisfactory	good	very good	
Localization	potentially very good**, but the center of the stereo image tends to become oversampled.	good	adequate	indistinct (potentially unstable)	
* These recording methods can also employ pressure gradient microphones, though this is not often done.					
** The appropriate angle between microphones depends on their directional pattern and the recording pattern.					

** These recording methods can also employ pressure gradient microphones, though this is not often done.

** The appropriate angle between microphones depends on their directional pattern and the recording angle.

MAT 240C

67

Placement Taxonomy

- Coincident
 - X/Y, M/S
- Near-coincident
 - ORTF, NOS, etc.
- Widely-spaced
 - A/B, OCT (2 or 3 cardioid), Decca tree (3 omnis),
 - 5- and 7-way trees: INA, OCT/S, Fukada
- Use of filters (OCT + LPF omnis)
- Speaker placement
- Phantom source problems



MEDIA ARTS & TECHNOLOGY PROGRAM

68

Complex Micing (the norm)

- Use of "spot," "main," and "room" microphones
- Delay plans for wave-front synchronization

Microphone	Lead / Lag (+ / - ms)	Compensation (ms)	Arrival-time Gap (ms)	Comp+Gap (ms)	Resulting Delay (ms)	Initial Direction
Main L	Time Base	0	0	0	-45	-
Main C	Time Base	0	0	0	-45	-
Main R	Time Base	0	0	0	-45	-
Spot A	+25	Ref. 1	-25	-12	-37	-30°
		Ref. 2	-25	-9	-34	+30°
		Ref. 3	-25	-17	-42	-110°
		Ref. 4	-25	-20	-45	+110°
Spot B	+35	Ref. 1	-35	-19	-54	-30°
		Ref. 2	-35	-21	-56	+30°
		Ref. 3	-35	-22	-57	-110°
		Ref. 4	-35	-25	-60	+110°
Spot C	+45	Ref. 1	-45	-17	-62	-30°
		Ref. 2	-45	-11	-56	+30°
		Ref. 3	-45	-19	-64	-110°
		Ref. 4	-45	-23	-68	+110°
Room L	-60	+60	-15	+45	0	-30°
Room R	-60	+60	-15	+45	0	+30°
Room LS	-60	+60	-15	+45	0	-110°
Room RS	-60	+60	-15	+45	0	+110°

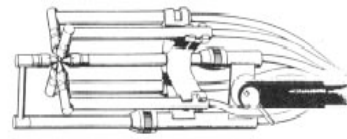
LT 240C

69

Novel Microphone Designs



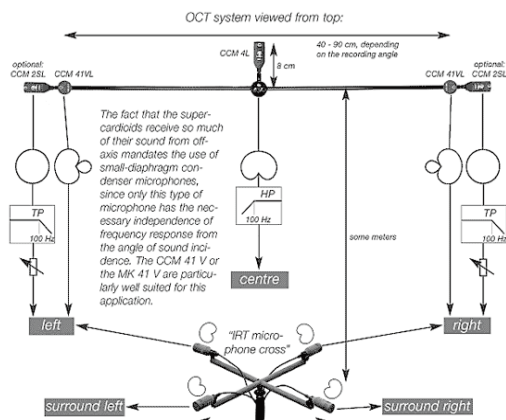
- Flexible coincident pair, sound field, binaural, OCT, and point mics



MEDIA ARTS & TECHNOLOGY PROGRAM

70

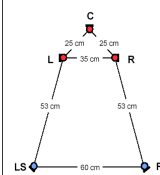
OCT Mic Configuration



MAT 240C

71

5-Channel Mic Mounts



- Geometry resembles (approximately) ITU 5.1-channel speaker layout
- See also ambiophonics



MEDIA ARTS & TECHNOLOGY PROGRAM

72

Mixing Spatial Sound

- (See MAT 242 -- a whole topic in itself)
- Base tracks without spatial information
 - Close-miced studio recordings
 - Traditional mix process creates space and positions
- Base tracks with spatial information
 - Medium-field live recordings
 - Typically use little or no processing in mix
 - Possibly mix-down to lower number of channels
 - Possible reformatting (e.g., ambisonic-to-surround) in mastering or on playback (our interest)

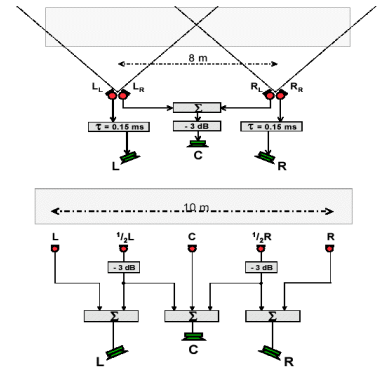
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

73

The Frontal Image

- L/R
- L/R + C
- “True” L/C/R
- See also
Creatophone



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

74

Surround Sound Mastering

- Front LCR spatial image
 - Usual techniques
- Use of surround channels
 - Multiple levels of reverb
- More than 2 surround channels (e.g., 7.1)
 - Side and rear surround
- Creating a height field (?)
- Special cases (sound effects, etc.)

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

75

Surround Sound Mix-down

- Whether to fold surround into LCR mix
- If yes, how?
 - Level
 - EQ
- If no, how to compensate for smearing and other artifacts?
- How to handle LFE channel
- LR vs LCR, etc.

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

76

Spatial Sound Formats

	2/0-Stereo	3/2-Stereo	WFS	Dummy-head
Horizontal direction	+30°...-30°	+30°...-30° surround effects	surround	surround (instable front)
Elevation	not possible	constraints	constraints ?	possible
Near-head distance	not possible	no?	possible ?	possible
Distance, depth	simulated	constraints?	possible	possible
Spatial impression	simulated	possible ?	possible	possible
Envelopment	not possible	constraints?	possible	possible

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

77

Processing Multichannel Sound

- Simple cases
 - EQ, etc.
 - Parallel processing chains
- Complicated cases
 - Dynamic-range processing
 - Spatial processing and reverb
 - Good-quality mixdown
 - N:M panning

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

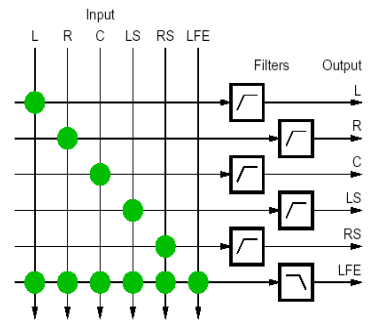
78

Processing Challenges

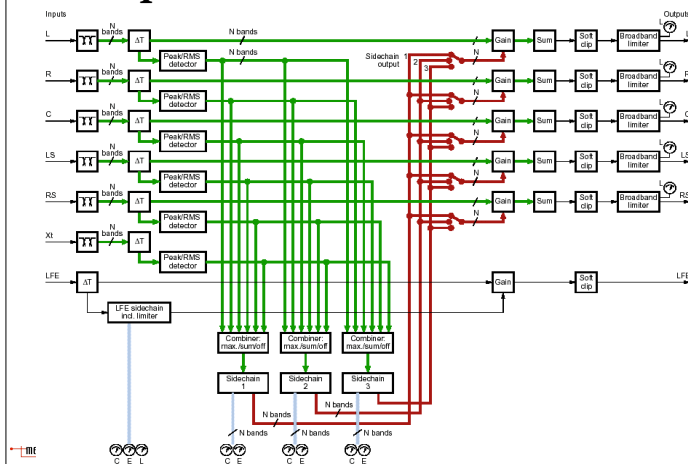
- Level, spectral, transient, sibilance, envelope, (etc.) detectors
 - Dynamic-range processors
 - Noise reduction
- N:M Processing
 - Mix-up and -down
 - Matrix coders
- Existence of mono LFE
 - Mix to full range before detectors
- Signal routing (may get complex)

Simplest Processing Matrix

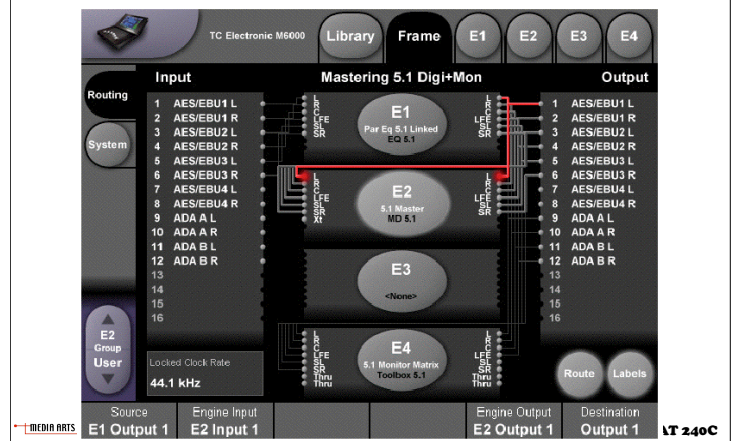
- LFE extraction
(green dots = variable gain summing nodes)



Complex Matrix (from Nielsen)



Routing in the TCE M6000



Review

- Many recording techniques for spatial or surround sound
- Mic layouts for 5.1 optimal
- Mixing with layered reverb for surround
- See Topic 3

MAT 240C Group Projects

- Real-time reverberator
 - FIR (done) and convolution-based versions
- HRTF player
 - Based on convolution-based reverb
- VR “auralizer”
 - Streaming sound I/O
- 5.1-channel output
- AlloSphere projects

Real-time Reverberator

- Streaming Moorer/Loy version
 - Based on existing code in CSL
 - Parameters derived from prior designs
- FIR and convolution- (FFT-) based versions
 - FIR version done in MAT 240B
 - For file-based input and file or streaming output
 - Convolution version based on MAT 240B PVsnd
 - Stand-alone program and VST plug-in needed
 - Low-latency scheduler (à la Bill G.) needed
- Derive/measure some impulse responses
 - CD-ROMs of room impulse responses
 - LLCH measurements

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

85

HRTF player

- Based on (low-latency) FFT-based reverb framework
- Add simple geometry engine and HRTF database
- Geometry assumes fixed source(s) and listener
- HRTF data sources
 - MIT
 - UWM
 - Others?

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

86

VR “Auralizer”

- Streaming or file-based sound I/O
- Geometry messages via OSC or CORBA
- Basic cues
 - IAAD, IATD
- Configurable physical model for reverb
 - Use ray tracer for early reflections
- Use HRTF for spatial filtering
- Integration with DIVE VR system

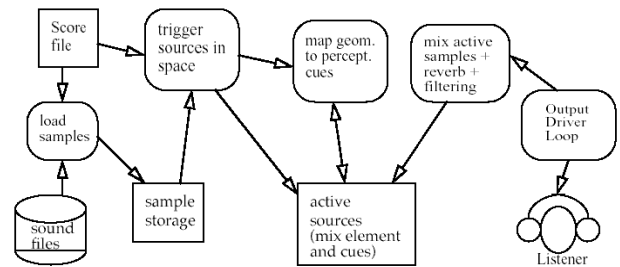
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

87

Old Auralizer Architecture

- 1992-2001: RIP



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

88

Projects: Next Steps

- Continue evolution of flexible play program
- N:M streaming and panning
- Simple format conversions
- New time-domain reverberators
- Convolution framework based on pv
- Start work on HRTF reader
- Start design of new auralizer
- Look at 5.1 & AlloSphere systems

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

89

What's Next?

- Matrix decoding and inter-format conversions
 - Loads of bad examples
 - Can we do better?
- Reverberators
 - Basic designs
 - Schröder and family
 - FIR & convolution-based solutions
- Coding!

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

90

MAT 240C

Digital Audio Programming:

Spatial and Surround Sound

Topic 3: Multichannel Matrix Playback

MAT 240C

91

Topic 3: Multichannel Playback

- Matrix decoding and inter-format conversions
 - Loads of bad examples
 - Can we do better?
- Models
 - Wire-based
 - Phase-based
 - Filter-based

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

92

Topic 3 Readings

- Surround Sound
 - Surround Mixing: History and Techniques
 - Surround Sound: Past, Present, and Future
 - Surround Formats
 - What is the LFE Channel?
 - A Chance for Enhanced Creativity

MAT 240C

93

Stereo “Matrix” Decoding

- Decode information in a 2-channel signal to derive front-back (or even height) information
 - Simple techniques can yield interesting results
 - Certain configurations work well for limited ranges of content
 - Popular in the 1970s with quad
 - Cheap simple hardware (cabling) solutions
 - Popular again now with CDs played on 5.1-channel systems
 - Fancier DSP-based solutions

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

94

Stereo Field Processing

- MS encoding and the field width
- LCR playback
- Relation to filters and delays



MAT 240C

95

Matrix Decoding

- Use features of the stereo signal
 - Simple: decorrelation (L/R difference signal)
 - Fancy: Derive spatial information from stereo (requires DSP)
 - Idea: Turn L/R stereo into M/S stereo and use some phase info in the S signal
 - OK, but how?

MEDIA ARTS & TECHNOLOGY PROGRAM

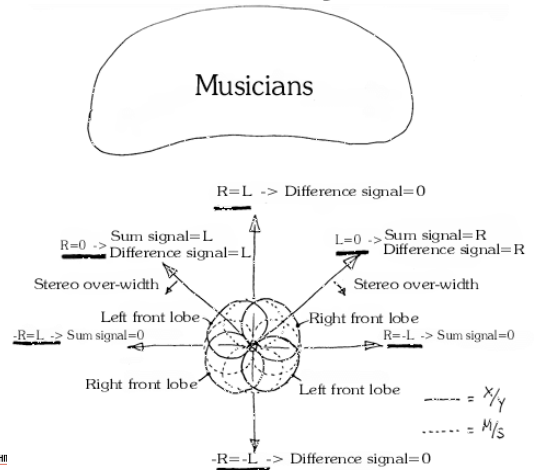
MAT 240C

96

Wire-based Systems

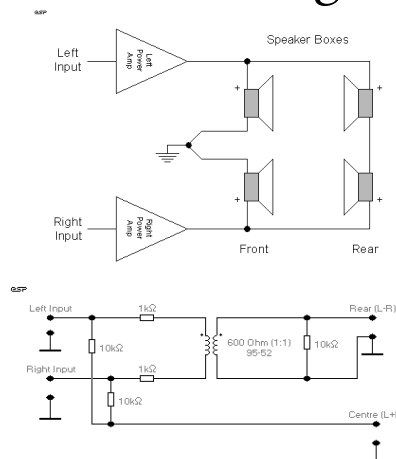
- Simplistic stereo-surround channel derivation
- Use L, R, sum, and difference signals
- No real relation to true spatiality
- Sounds “deep” for many simple stereo signals
- Pathologically bad for binaural and highly decorrelated signals
- Often touted as “4-channel matrix decoding”

Matrix Signals

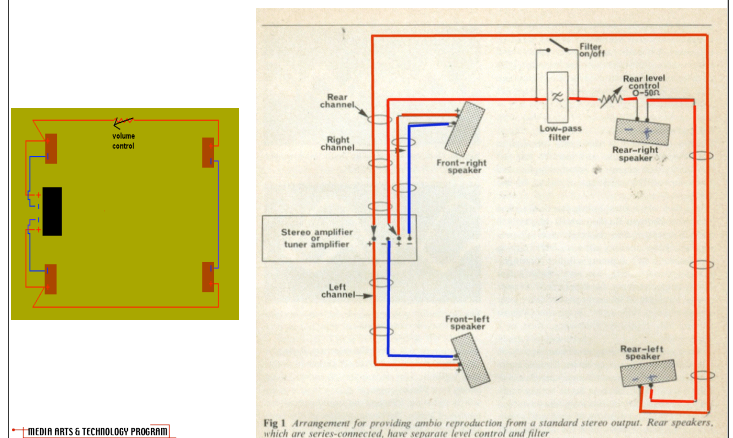


Simplest: Hafler “Decoding”

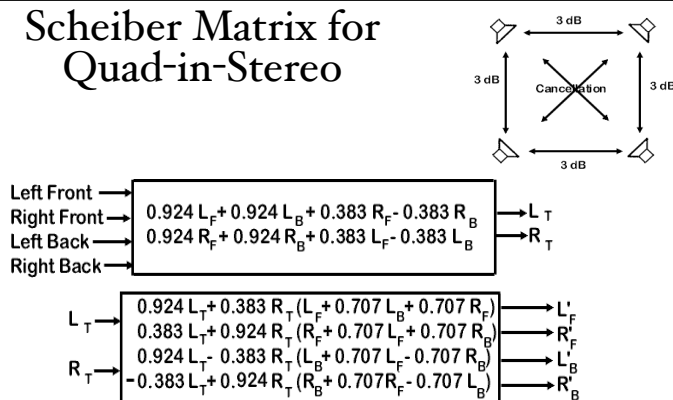
- Sum/Difference signals
- Front/rear out of phase
- Hopelessly cheesy



A Simple Old-fashioned Matrix Setup

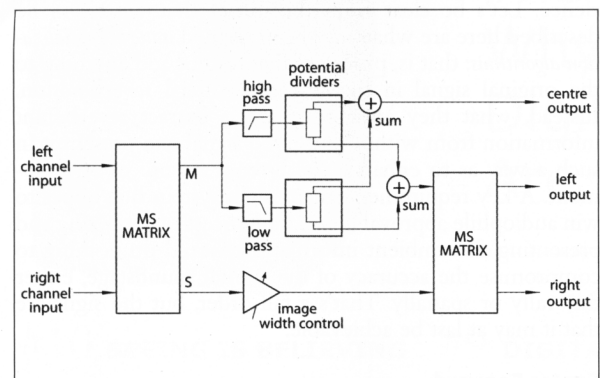


Scheiber Matrix for Quad-in-Stereo

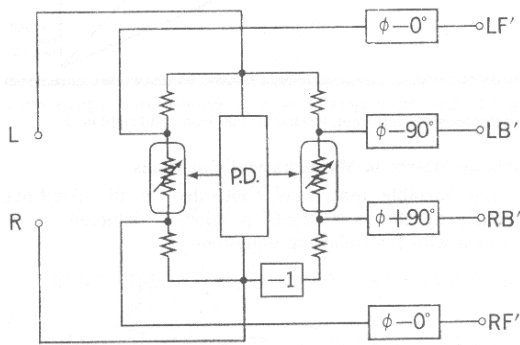


- Equations and separation diagram (why does this even work at all?)

Tri-field LCR Decoder



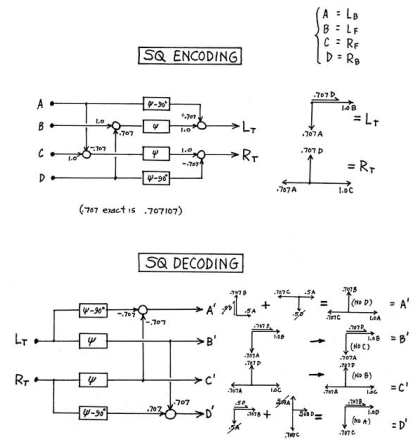
Sansui Phase-Discriminator Based Stereo-to-SQ Decoder



MAT 240C

103

Wendy's SQ Analysis

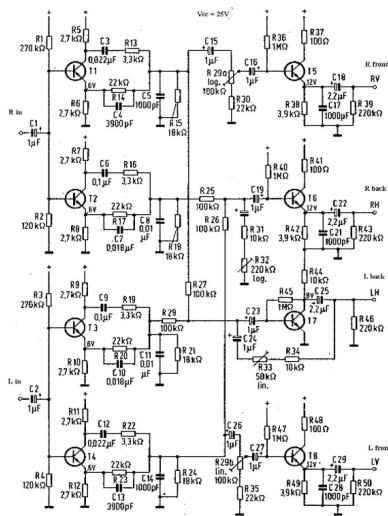


© 1993 Wendy Carter - All Rights Reserved

MAT 240C

104

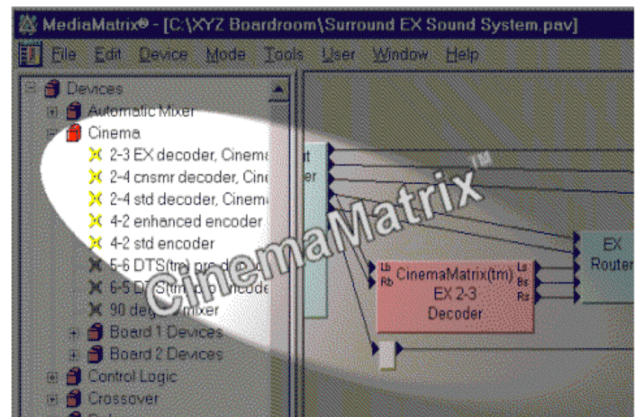
Solved in Analog



MAT 240C

105

CinemaMatrix Decoder

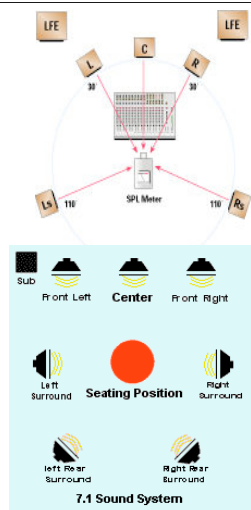


MAT 240C

106

Versions of ITU 5.1

- 5.1 layout, geometry, speaker characteristics
- LFE processing
- Add rear channels: 6.1, 7.1
- Add subs: 7.2
- Add elevation dimension: 10.2
- "Optimal" 14.4

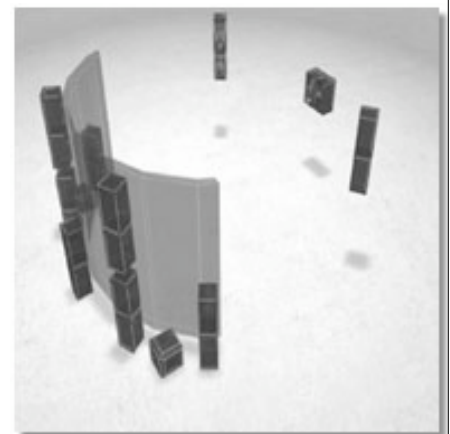


MAT 240C

107

10.2-Channel Surround

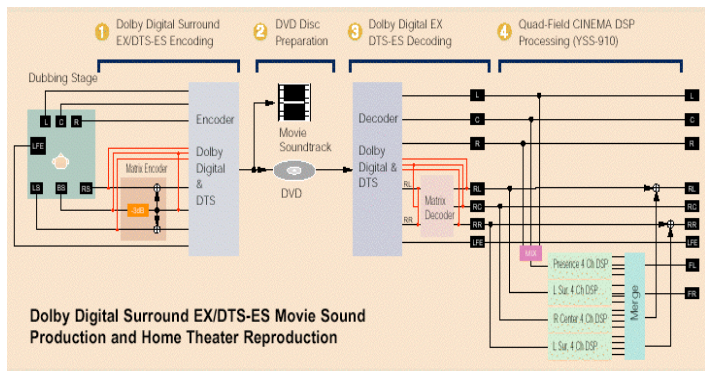
- Adds height dimension
- Front/rear subs



MAT 240C

108

6.1-Channel Dolby Digital Matrix

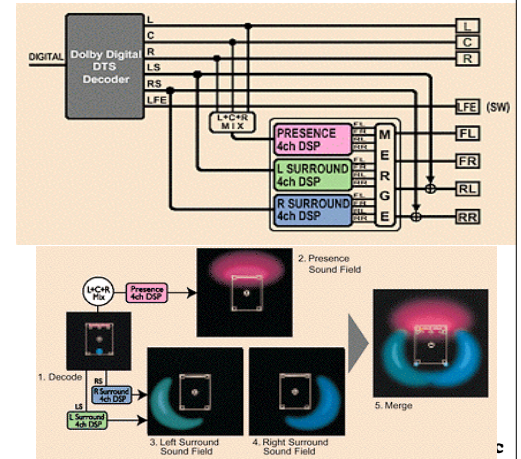


MAT 240C

109

DSP-based Surround Matrix

- Yamaha



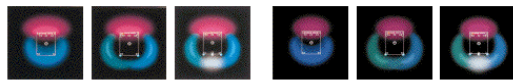
MEDIA ARTS & TECHNOLOGY PROGRAM

110

Surround Matrix Applications

- Examples

Stereo Input (70mm Spectacle) 5.1-Channel Input (Tri-Field CINEMA DSP) 6.1-Channel Input (Quad-Field CINEMA DSP) Stereo Input (70mm Sci-Fi) 5.1-Channel Input (Tri-Field CINEMA DSP) 6.1-Channel Input (Quad-Field CINEMA DSP)



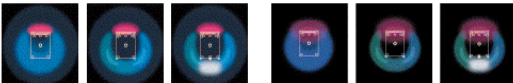
Spectacle

This program transports you into the middle of the scenes you are watching. In a very wide space, every sound, even large sounds, are heard clearly. A new level of sound realism.

Sci-Fi

Reproduces dialogue, music and effects of the latest SF high-tech movie soundtracks with excellent separation. Capable of conveying the impression of a huge space.

Stereo 5.1-Channel Input (Tri-Field CINEMA DSP) 6.1-Channel Input (Quad-Field CINEMA DSP) Stereo 5.1-Channel Input (Tri-Field CINEMA DSP) 6.1-Channel Input (Quad-Field CINEMA DSP)



Pop/Rock

This program is for recreating live music venues. Its presence sound field enhances the action on stage. The surround sound field extends far behind the screen.

DJ

The presence sound field uses opera house data, while surround uses concert hall data. You'll hear the DJ's voice with exceptional clarity and music with rich depth.

MAT 240C

111

Cinema Sound Formats

Comparison of Movie Sound Formats

Surround Format	Dolby Pro Logic	Dolby Digital	Dolby Digital EX	DTS Digital Surround	DTS-ES	DTS 9624
Media	DVD, VCR, TV, LD, CD	DVD, LD	DVD, LD	DVD, LD, CD	DVD	DVD
Reproduction Channels	4 Channels	1 — 5.1 Channels	6.1 Channels	5.1 Channels	6.1 Channels	5.1 (8.1) Channels
Channel Configuration	Left, Center, Right and Surround	Left, Center, Right, Left Surround, Right Surround and LFE	Left, Center, Right, Left Surround, Center Surround and Right Surround and LFE	Left, Center, Right, Left Surround, Right Surround and LFE	Left, Center, Right, Left Surround, Right Surround and LFE	Left, Center, Right, Left Surround, Right Surround and LFE
Encode System	Matrix	AC-3 (Digital)	AC-3 (Digital)	Coherent Acoustics	Coherent Acoustics	Coherent Acoustics
Sound Process	Matrix	Discrete	Discrete and Matrix	Discrete	Discrete and/or Matrix	Discrete (Discrete and Matrix)
Frequency Response (Front)	20–20,000 Hz	20–20,000 Hz	20–20,000 Hz	20–20,000 Hz	20–20,000 Hz	20–40,000 Hz
Frequency Response (Rear)	100–7,000 Hz	20–20,000 Hz	20–20,000 Hz	20–20,000 Hz	20–20,000 Hz	20–40,000 Hz
Resolution Ratio	20 dB	20 dB	20 dB	20 dB	20 dB	24 dB
Sampling Frequency	48 kHz	48 kHz	48 kHz	48 kHz (LD/CD)	48 kHz (LD/CD)	96 kHz
Conversion Rate (CD/LD)	348 kbps (LD)	348 kbps (LD)	1,411 kbps	1,411 kbps	1,411 kbps	1,411 kbps
Conversion Rate (DVD)	448 kbps	448 kbps	1,536 kbps	1,536 kbps	1,536 kbps	1,536 kbps
Remarks			Center surround signal is matrix processing.		Center surround signal is discrete or matrix processing.	

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

112

Performance Spaces

- Paris (IRCAM, GRM)
- Karlsruhe, ZKM
- Belfast, SERC
- Santa Barbara, CNSI
- Birmingham, BEAST
- San Francisco, Audium
- Graz, TU

MAT 240C

113

ZKM Karlsruhe



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

114

BEAST, Birmingham

- Circular system with overhead tweeter array



MAT 240C

115

Audium, SF



MAT 240C

116

Directional Sources

- Model anisotropic radiation pattern of natural sources



MAT 240C

117

What Can We Do?

- 2-to-5.1-channel decoder
- All in software
- Uses:
 - Stereo sum/difference
 - Level of decorrelation
 - Phase shifts via all-pass filters
 - FFT and filters?
 - Added reverb to rear channels

MAT 240C

118

Coding Steps

- STP's WackyPanner example
 - Look at call-back function
- Incorporate Hafler-style decoding
 - Simple code for simple version...
- Add all-pass filter for phase shift
- Add delay for rear channels
- Add reverb (see below) for rear channels
- More to come here...

MAT 240C

119

What's Next?

- Reverberators and convolution
 - Readings
 - Code examples
 - Applications, plug-ins
 - Listening examples

MAT 240C

120

MAT 240C

Digital Audio Programming:

Topic 4: Reverberators and Convolution

MAT 240C

121

Topic 4: Reverberators

- Reverberators
 - Basic models and their parameters
 - Schröder and family
 - Implementations
 - Convolution methods



MAT 240C

122

Topic 4 Readings

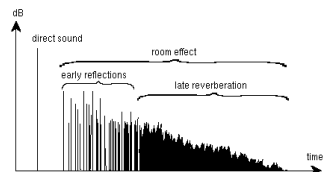
- Reverberation
 - Sweet Sound of Reverb
 - Reverberation, Master Handbook of Acoustics
 - Efficient Models for Reverberation and Distance Rendering
 - Reverb and Room Simulation in the Multichannel Era
 - Efficient Convolution without I/O Delay
 - Room Acoustic Simulator

MAT 240C

123

Reverberation

- Early reflections
 - <50 msec or so
 - Should be spatialized (or at least panned)
 - Tell the listener about the room geometry and source position
 - Are different for each source and listener
- Late reverb “tail”
 - Tells listener about the room’s characteristics
 - Details are less important: T60 time and decay shape



MAT 240C

124

Early Reflections

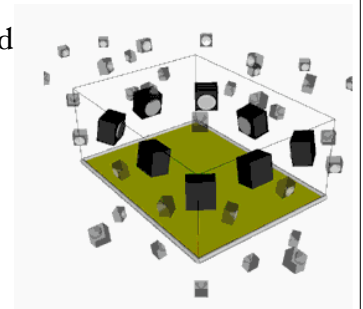
- Parametric
 - Taken from somewhere...
- Physically-informed
 - E.g., random primes with some distribution
- Model-derived
 - Acoustical ray-tracing
 - 10-20 reflections (3rd-order), 100-200 msec
- Measured
 - Impulse response capture

MAT 240C

125

Early Reflections

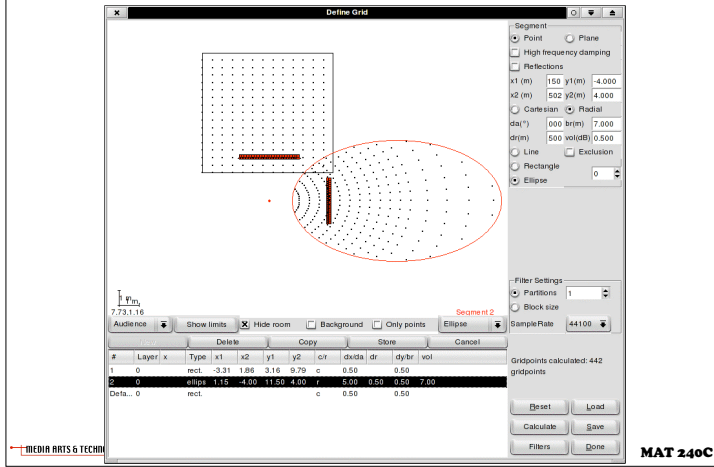
- Given a room model and src/listener position, these can be computed
- Geometry can be interpreted as a set of “phantom” sources



MAT 240C

126

Virtual Sources in SWonder



127

Reverb Tail

- Based on early reflections
 - Could simply be tap output sent to parallel comb filters
- Stochastic
 - Given some characteristics
- Statistical
 - Given some description
- Measured

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

128

Using Reverb

- Settings depend on material!
- What sounds open and clear with one instrument (e.g., spiky) might sound dull and muddy with another (legato)
- This is an important part of mixing/mastering

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

129

Reverb Examples

- Dry, then with various IRs
- Snare
- Rim shot
- Pan Flute
- Futter-tongue

(There are sound files here)

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

130

Reverb Examples

- Snare
 - dry - this is a Ludwig maple-frame snare from about 1968
 - IR1 - processed using Peak's ImpulseVerb with an IR from the Old South Church in Boston
 - IR2 - preset called "Entry Hall-CKSBE-length"
 - IR3 - IR from the Trinity Church in Boston
- RIM TAP
 - dry - Ludwig snare played cross-stick style, as you might hear in a ballad
 - IR1 - IR from the Bethany Church in Montpelier, Vermont
 - IR2 - preset "Entry Hall-CKSDE-width"
 - IR3 - high end reverb processor -- note the high end "sizzle"
- PAN 1
 - dry - hand-carved Peruvian Pan Flute sample
 - IR1 - public domain "Ambience Hall" IR
 - IR2 - IR from the Santa Chiarra Cathedral
 - IR3 - Peak preset IR of the Gloucester, Massachusetts City Hall
- PAN Flutter
 - dry - the same Peruvian Pan Flute played with a "flutter tongue" effect
 - IR1 - Peak IR preset of the Domkyrkan Cathedral
 - IR2 - Peak preset IR file called "Laminated Glass Room"
 - IR3 - Peak preset IR file called "Rough Finish Plaster Room"
 - Verb - "Small Hall" reverb preset from a popular \$3,000 synth

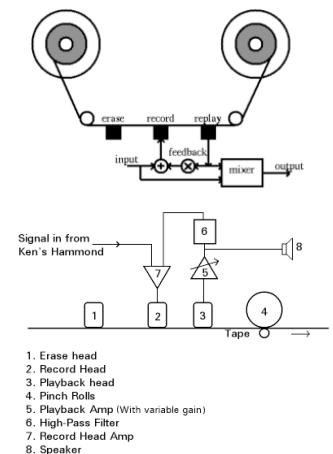
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

131

Mechanical Reverb Systems

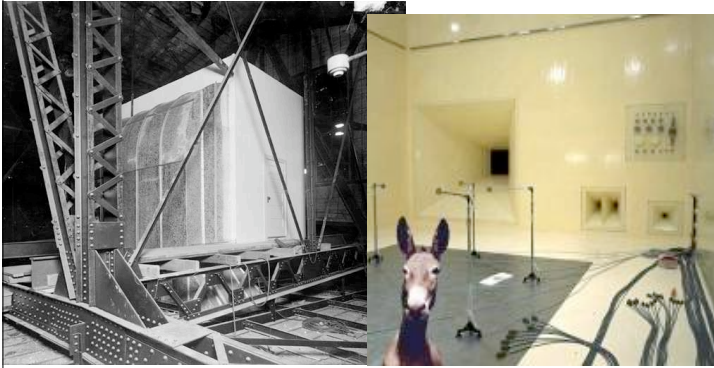
- Box/chamber
- Tape
- Spring
- Plate
- Others



MEDIA ARTS & TECHNOLOGY PROGRAM

132

Echo Chambers



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

133

Tape Loops: EchoPlex (1960s)

- Tape loop with moveable play/feedback head
- Echo
- Doppler
- Flanging

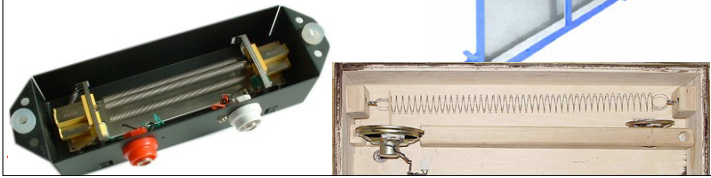


MEDIA ARTS & TECHNOLOGY PROGRAM

134

Springs and Plates

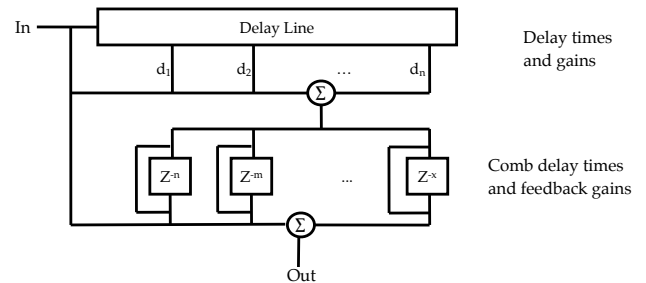
- Metal foil or spring with speaker/mic attached
- Variable physical damping on plate/spring



MEDIA ARTS & TECHNOLOGY PROGRAM

135

Schröder/Moorer/Loy Reverberator



- Multi-tap delay line (early) + bank of parallel recirculating delay lines (late)

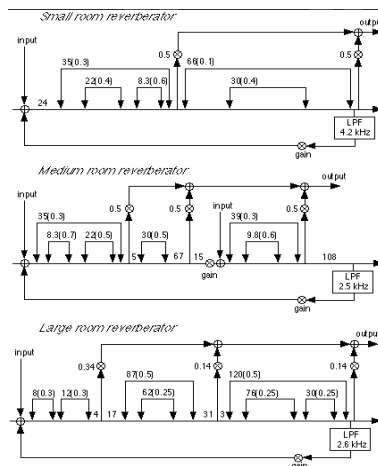
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

136

Physical Models and Reverberation

- Early reflections: based on exact room geometry (acoustical ray tracing) (or based on tweaked magic numbers)
- Late reverb according to general characteristics



MEDIA ARTS & TECHNOLOGY PROGRAM

137

Implementation of Schröder

- Basic multi-tap delay line
 - Single buffer, pointer arithmetic
- Recirculating delay lines (comb filters)
- Low-pass filters (in comb feedback)
- All-pass filters (for phase response)
- System architecture
- Parameter sets (oh yeah...)

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

138

Data Structures (from au)

- Sample buffer
 - `float * buffer = (float *) malloc (size);`
- Tap
 - `*out++ = *tap++;`
 - If (`tap > end`) `tap = start;`
- Comb filter
 - `*out = *tap++;`
 - `*in++ += *out++ * scale_value;`

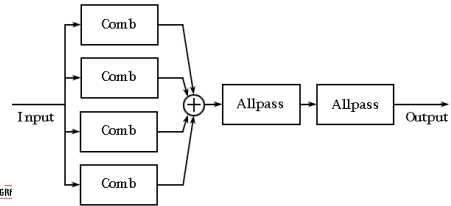
MAT 240C

139

Freeverb



- Modified Schröder/Moorer/Loy
- 8 parallel combs followed by 4 all-pass filters
- Separate L/R combs/times (more computation)
- Carefully tuned parameters and setting mapping
- Available as source, VST plug-in, or in CSL



MAT 240C

140

Coding

- Add reverb function to play program
- Clean up ATON auralizer reverb
- Use LPREV data sets
- Try freeverb data
- Try NC's SC reverb data

MAT 240C

141

Reverberator Parameters

- Basic variables
 - Pre-delay
 - Reflection build-up
 - Early reflection density
 - Tail length
 - Tail early decay rate
 - LP filtering
 - Wet/dry mix
- These can be mapped onto a smaller set...

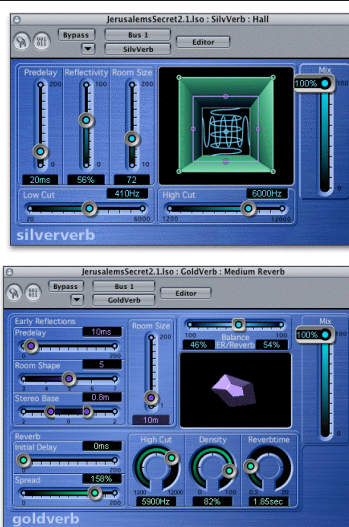


MAT 240C

142

Reverberator Parameters

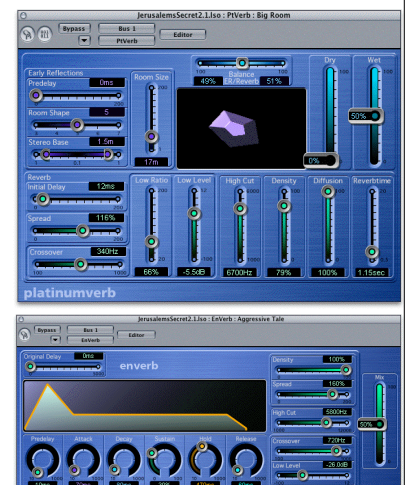
- Example: Logic plug-ins
- AVerb (above)
- SilVerb: adds LP/HP filter (& useless picture)
- GoldVerb: adds room shape, stereo width, "spread," "balance"



143

Advanced Reverb Plug-ins

- Example: Logic
- PlatinumVerb: adds cross-over, diffusion
- EnVerb: adds fine control over reverb envelope



144

Convolution Reverb

- Use recorded or computed IR
- Take its FFT (one long one or in blocks)
- Convolve in the frequency domain with buffers of input
- Simple single-block solution: latency = IR length
- Smarter: store previous input FFTs (1-block latency) loop for IR-len / block-size blocks
- Remember window shape constraint for COLA

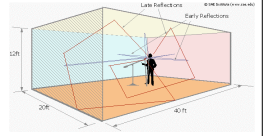
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

145

Impulse Responses for Reverb

- IR measurement
 - Source selection (gun, clapper, speaker)
 - Mic placement (variable)
 - Post-processing (de-convolution)
- Synthetic IR computation
 - Acoustical ray-tracing from physical model
- Stereo decorrelation (increase or decrease)
- Prep. for convolution processing



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

146

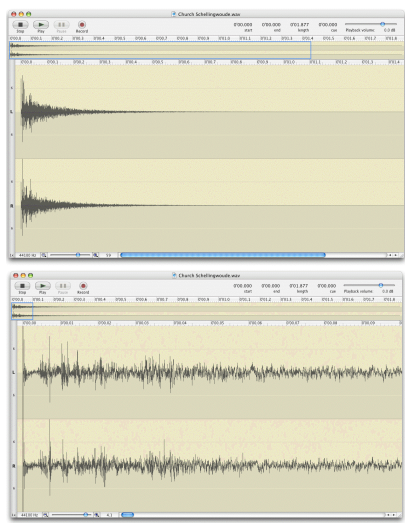
IR Examples

- Small church in Schellingwoude, NL
- 1.5-sec and zoomed-in to first 100 msec



Impulse Response by:
Fokke van Saane

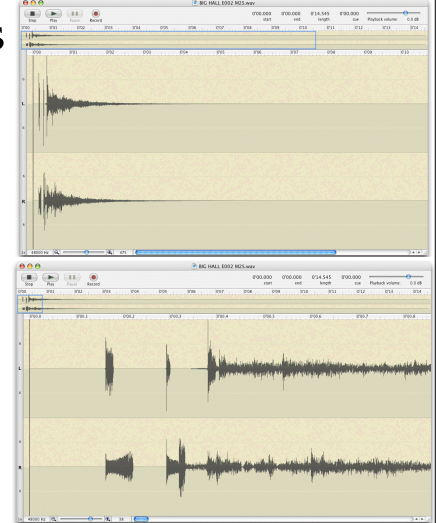
2x Schoeps CMC5 MK5 omni
wide spaced A-B
Postex_PD4 DAT recorder
6mm caliber gun shot



147

IR Examples

- Generic “big room” and stereo attack portion



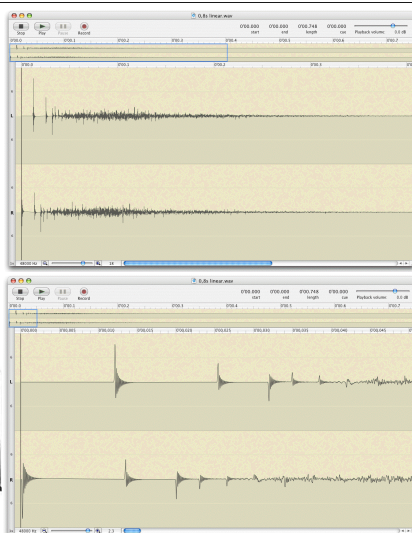
148

IR Examples

- EMT 244 Digital Reverb (recorded)

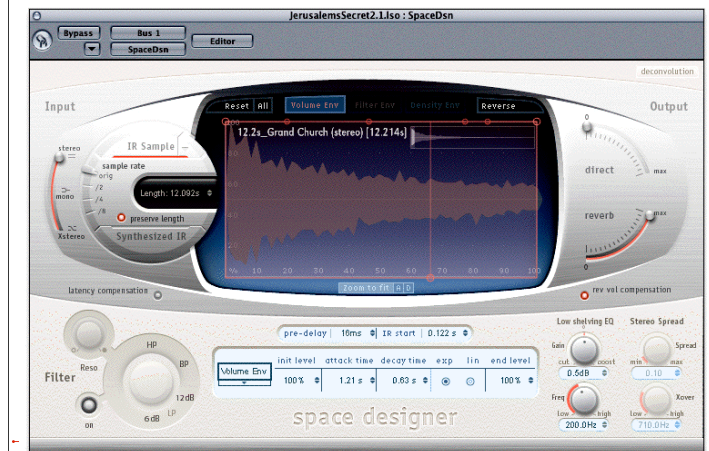


MEDIA ARTS & TECHNOLOGY PROGRAM



149

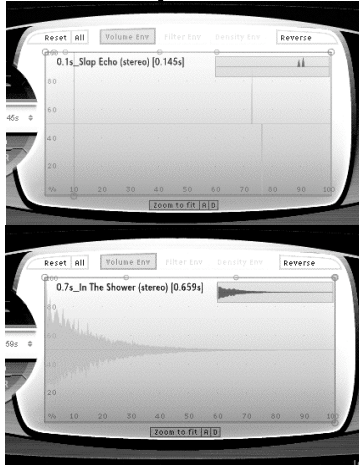
Convolution Reverb GUI



150

Reverb Impulse Responses

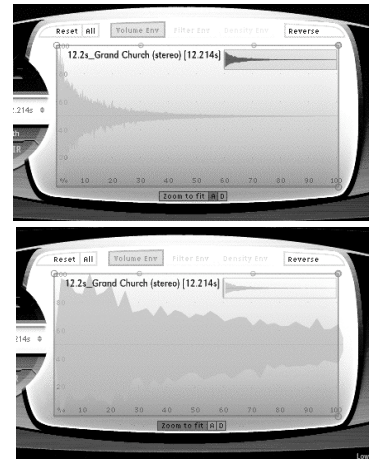
- Slap echo @ 100 msec
- Shower stall (0.7 sec)



151

Reverb Impulse Responses

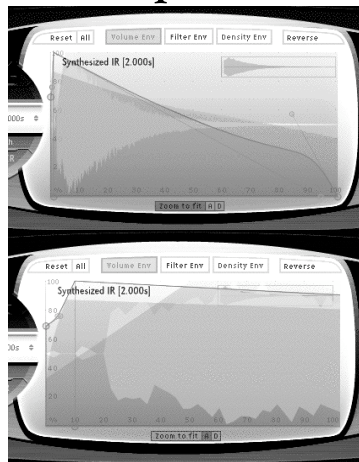
- 12-sec “grand church” IR and its first 100 msec



152

Reverb Impulse Responses

- Synthesized (slow rise) IR and its attack segment



153

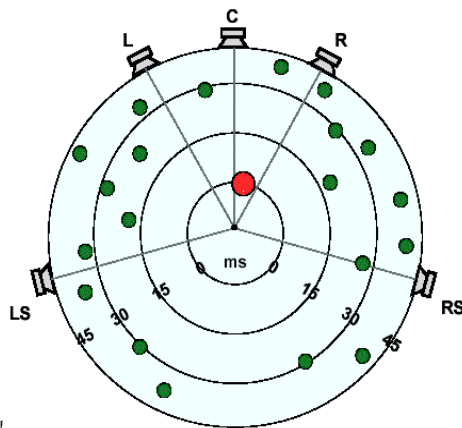
Advanced Reverb

- Mixing in different “rooms”
- Stereo reverb decorrelation
- Reverb for surround-sound mixing
- IR combination
- Surround panning (VBAP or even HRTF) of individual reflections
- Reverb and spatializers

MAT 240C

154

Reverb in 5.1 Channels



MAT 240C

155

Low-Latency Convolution

- Problem: Long IR calculations are expensive to do with direct-form convolution ($O(N^2)$ complexity); but the FFT solution implies long latency (length of the IR itself)
- Solution 1: Single block latency (~ 512 samples)
- Solution 2: very low latency

MAT 240C

156

Partitioned FFT

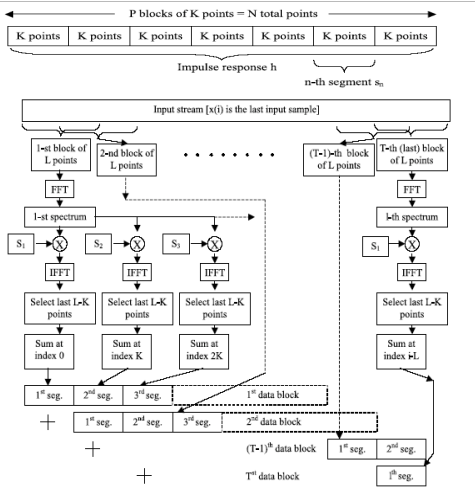


Figure 5: Partitioned convolution.

157

Partitioned Convolution in CSL

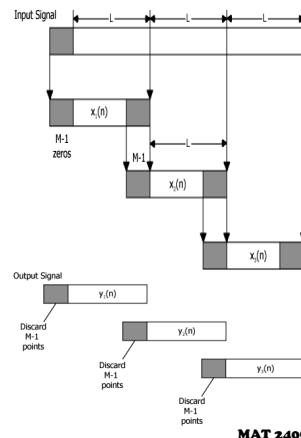
```
void Convolver :: nextBuffer(Buffer & outputBuffer, unsigned channel) throw (Exception) {
    unsigned which = winCount % numBufs; // pseudo ring buffer addressing
    if (mMyInput) {
        Effect::pullInput(outputBuffer); // get a buffer from the input stream
        CSL_FFTW_exec(mForwardPlan); // FFT input from mSampleBuffer into mSpectrumBuffer
        memcpy(minFFTs[which], mSpectrumBuf, bufsize); // copy it into the minFFT[] buffer
    }
    bzero(mSpectrumBuffer, bufsize); // zero out the working FFT buffer before loop
    // loop to multiply/accumulate spectra for convolution
    for (unsigned i = 0; i < numBufs; i++) {
        which = (numBufs + winCount - i) % numBufs; // which input FFT to use
        complex_multiply_accumulate( // sum complex vectors
            mFilterFFTs[i], // mult current IR window
            minFFTs[which], // by past input in reverse order
            mSpectrumBuffer); // sum into mSpectrumBuffer
    }
    CSL_FFTW_exec(mInversePlan); // execute the IFFT for spectrum domain convolution
    memcpy(outputBuffer.mMonoBuffers[0], mSampleBuffer, numFrames * sizeof(sample));
    winCount++;
}
```

MAT 240C

158

Overlap-Save Method (Buckiewicz-Smith)

- For M length filter, we break the signal into N sample blocks, with each block overlapping the last by $M - 1$ samples
- Discard first $M - 1$ points of resulting vector and concatenate output
- Introduces delay of at least N samples



MAT 240C

159

Scalability of P-FFT

- Example (from BruteFIR) - 4 FFTs of 8192 points each on 1 GHZ processor

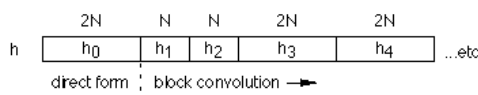
delay in ms	processor load	partition size	number of partitions
3 ms	60%	64 samples	128
6 ms	30%	128 samples	64
12 ms	16%	256 samples	32
24 ms	11%	512 samples	16
47 ms	8%	1024 samples	8

MAT 240C

160

“Zero-latency” Convolution

- Problem: even block-wise convolution suffers with very small block sizes
- Solution: do a small block (e.g., 32 samples) in the time domain, then break the FFT up into pieces (partitioned FFT)



MAT 240C

161

Minimum Cost Method

- Divide the impulse response into sequential blocks h_0, h_1, h_2 , etc. so that we:
 - Don't introduce delay between input and output
 - Minimize computation cost
- To do these we must:
 - Implement h_0 direct multiplication
 - Set the size of convolution blocks to be as large as possible

MAT 240C

162

MAT 240C

- 163

MEDIA ARTS & TECHNOLOGY PROGRAM

- MEDIA ARTS & TECHNOLOGY PROGRAM**

MEDIA ARTS & TECHNOLOGY PROGRAM

MEDIA ARTS & TECHNOLOGY PROGRAM

MEDIA ARTS & TECHNOLOGY PROGRAM

Complexity

	Multiplications	Additions
No prior results	$3N \log N + 6N$	$\frac{1}{2}N \log N + 8N$
Given previous half-sized input spectra	$2N \log N + 7N$	$3N \log N + 7N$
Given input spectrum	$N \log N + 4N$	$\frac{1}{2}N \log N + 3N$

- Per output sample, multiply accumulate will require N multiplications
- Block convolution requires $3\log N + 6$ multiplications
- Block convolution requires fewer multiplications than multiply accumulate starting at $N = 32$
- Cost of block convolution varies depending on how the input spectra must be calculated:

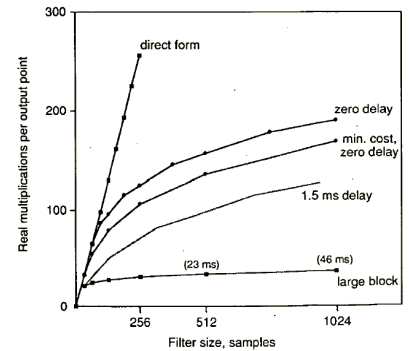
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

169

Cost

- If we remove the FIR filter, we get a delay of 1.5 msec when $N = 32$ at 44.1kHz, with significant cost reduction
- Impulse response of 128,000 samples requires 427 multiplies per output sample with zero-delay (as opposed to 128,000)



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

170

Issues

- It's a bear to code (scheduling, DSP, I/O, cacheing, etc.)
- There are a few unresolved issues (see HRTF section)
- There's that Lake patent (McGrath, 1993) to ignore
- Otherwise, it's ubiquitous (commercial HW/SW IR-based reverberators)
- See Brent Lehman's thesis

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

171

Convolution Implementations

- Basic FFTW-based solutions (see MAT 240B)
- Using FFTW in CSL: Spectral class
- CSL block-partitioned convolver
- BruteFIR and distributed convolution

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

172

Distributed Convolution

- CRAM-managed CSL FFT servers
 - Stream samples using RFS protocol
 - Single sync'ed output server
- BruteFIR
 - Up to 256 inputs and 256 outputs, modular IO
 - Mix/copy channels before and/or after filtering
 - Cascade filters or build complex filter networks
 - Typical filter lengths are in the range 2048 - 262144 taps
 - Reasonable low, fixed I/O-delay (typically 200 ms)
 - Cross-fade for seamless filter coefficient changes.
 - 32 or 64 bit floating point internal resolution.
 - Supports multiple processors

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

173

Review

- Reverberation
 - Architectures
 - IR data
- Convolution
- High-performance Processing
 - Long-tail
 - Low-latency
 - Distributed



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

174

What's Next?

- HRTFs
 - Theory
 - Data sets
 - Coding
 - Controlling

MAT 240C

175

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

Digital Audio Programming:

Spatial and Surround Sound

Topic 5: Head-Related Transfer

Functions

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

176

Topic 5 Readings

- Head-Related Transfer Functions
 - Overview of the Head-Related Transfer Functions (HRTFs)
 - Binaural Audio in the Age of Virtual Reality
 - HRTF Measurements of a Dummy-Head Microphone
 - The CIPIC HRTF Database
 - Wavelet-based Spectral Smoothing For HRTF Filter Design

MAT 240C

177

Head-Related Transfer Functions

- Task: model the direction-specific filtering of the human anatomy
 - Pinna, ear canal, upper body
- Used for spatialization over headphones, rather than speakers
- Easy to measure, very effective
- Complex to implement
- Difficult to generalize



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

178

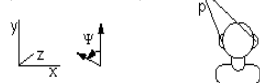
Geometry and Localization

Geometry

(seen from the top)



(seen from the back)



Spatial Cues

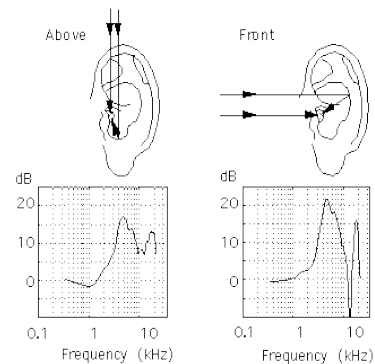
Distance (d) = $(b + a) / 2$ Loudness $\propto d^2$ L/R ratio $\propto (b - a) / e$ Low-pass filter $\propto d^2$ Direct/reverb ratio $\propto d^2$ Spatial low-pass filter ratio $\propto \theta$ Spatial band-reject filter $\propto \psi$ Inter-aural time delay $\propto (a - b) / e$ Initial/decay reverb ratio $\propto \psi$

(many more possible)

MAT 240C

179

Pinna Shape and Reflections



- Individual differences have a huge impact

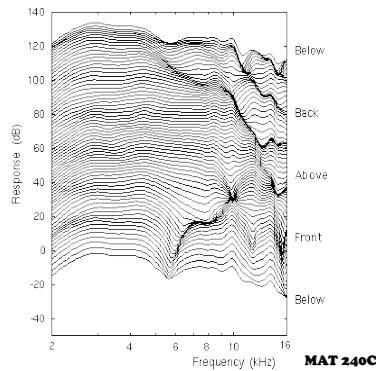
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

180

The HRTF

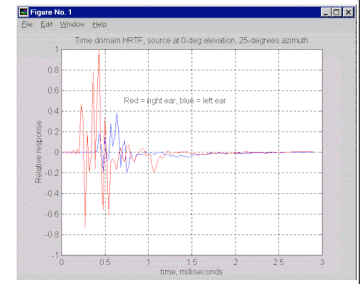
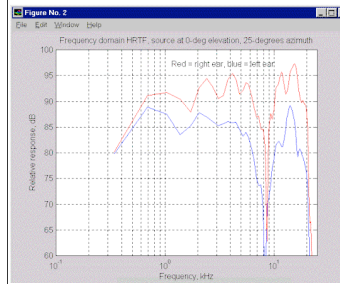
- Benefits: good localization of single sources
- Problems: ...



MAT 240C

181

HRTF and HRIR

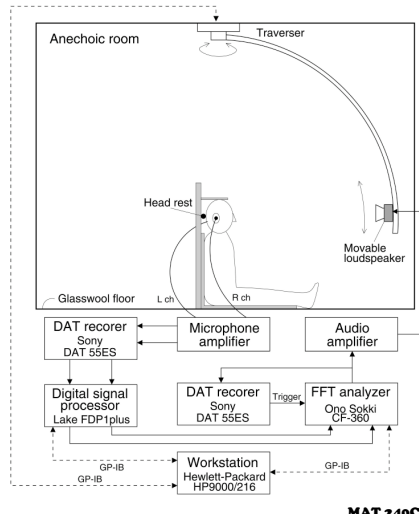
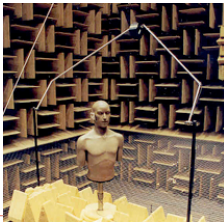
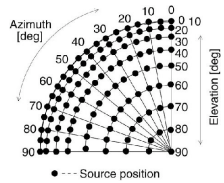


MAT 240C

MAT 240C

182

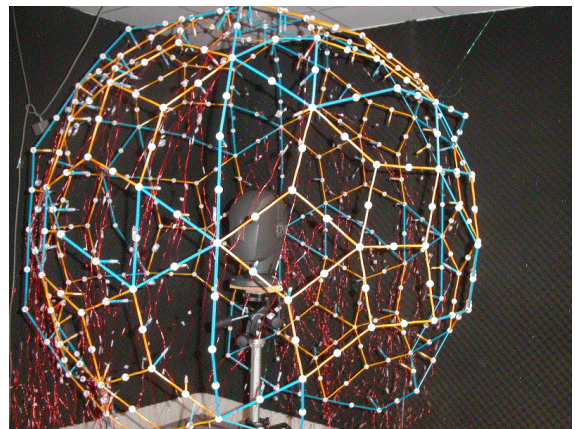
HRTF Measurement



MAT 240C

183

UMD HRTF Mic. Array



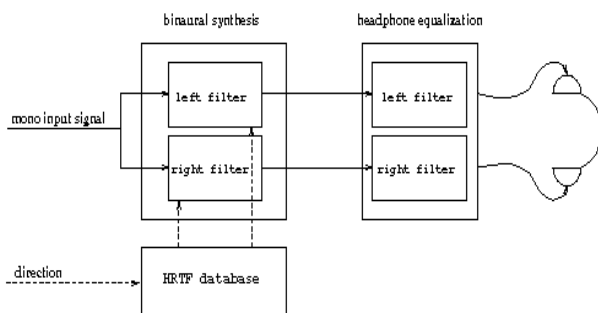
MAT 240C

MAT 240C

184

HRTF Processing

- (Per virtual source)



MAT 240C

185

The Problem with HRTFs

- Mine don't sound good to you
- The average of mine and yours don't sound good to either one of us
- We have many years training with ours
- It's a 4-D function!
- But, if every user could have it measured...

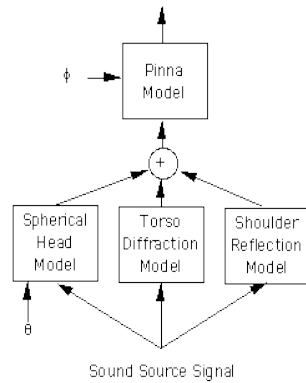
MAT 240C

MAT 240C

186

Composite Models

- Source radiation pattern
- Room and early reflections
- Various body interactions
- That's not even all!



MAT 240C

187

HRTF Data Sets

- Several available HRTF data sets
 - UW, MIT, IRCAM
 - Each has different spatial resolution and format
- Data reduction is an active R&D topic
 - “Generic” HRTF family
 - HRTF data interpolation

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

188

Steps in HRTF Processing

- It's simple:
 - Load the DB of spectra (IR files)
 - Get desired source position (real-time tracking?)
 - Play source through FIR filter (or convolve)
 - See WG's SGI code (@MIT)
 - See the CSL code
- Challenges
 - How to model movement between data positions
 - How to do lots of changing FIR filters

MAT 240C

189

HRTF Code Examples

- See CSL (C++, partial)
- MIT (C, obsolete I/O API)
- CRC 3-D sound player
- Low-latency convolution in CSL
- Others...

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

190

What's Next?

- Playback of Spatial/Surround Sound
 - Vector-based amplitude panning
 - Ambisonic representation and rendering
 - Wavefield synthesis
- Readings
- Code examples

MAT 240C

191

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C Digital Audio Programming: Spatial and Surround Sound Topic 6: Spatialization

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

192

Topic 6: Spatialization

- Spatialized/Auralized Sound
 - Vector-based amplitude panning (VBAP)
 - Ambisonic representation and rendering
 - Wavefield synthesis (WFS)
 - Control, GUIs, and spatialization applications

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

193

Topic 6 Readings

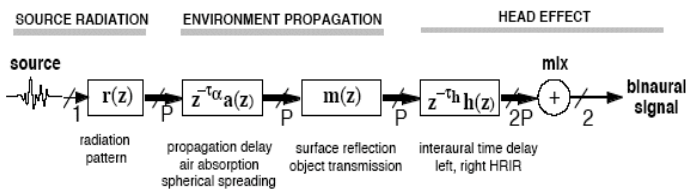
- Spatialization
 - Synthesizing 3-D Sound Scenes...
 - Real-Time Audio Spatialization with Inexpensive Hardware
 - A Beam-Tracing Approach to Acoustic Modeling
 - Original VBAP Paper
 - Ambisonics intro
 - WFS Descriptions

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

194

The Big Picture



- Source (freq.-dependent radiation pattern)
- Filter (air absorption, reflections)
- Receiver (ears, body)

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

195

Playback Options (after McCoy)

- Head Related Transfer Function (HRTF)
 - Most effective for headphones
 - Computationally expensive filters
 - Filter model specific to each user
- Auralization (synthesized room reflections)
 - Computationally expensive
 - Requires geometrical room model
 - Small sweet spot if not on headphones

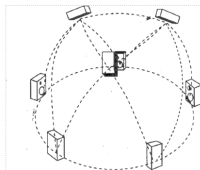
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

196

Playback Options

- Vector Base Amplitude Panning (VBAP)
 - Reformulation of pan-pot stereo panning for 3D
 - As “realistic” as panning
 - Variable # and layout of speakers
 - Can incorporate reverb model
 - Computationally efficient
 - Adaptable to (almost) arbitrary speaker geometry
 - Adaptable to any (large) number of output channels



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

197

Playback Options

- Ambisonics
 - Flexible model and representation
 - Separate encoding & decoding stages
 - Variable complexity and # of outputs
 - Assumes “regular” speaker arrangement
 - Ignores reverb
 - Adaptable to but not ideal for large arrays of speakers or irregular speaker placement

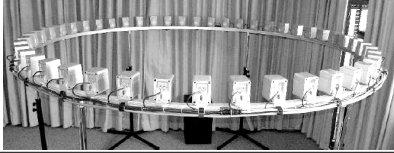
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

198

Playback Options

- Wave Field Synthesis
 - Total sound field reproduction via superposition
 - Potentially (?) very accurate
 - Ignores reverb
 - Computationally demanding
 - Requires large # of regularly spaced speakers



MAT 240C

199

Questions/Criteria

- Source
 - Mono + geometry
 - Mixed N-channel
- Speaker distribution
 - 2D, 3D
 - Headphone playback
- Spatial cues used
 - Direction, distance, room, etc.
- Control, GUI, application

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

200

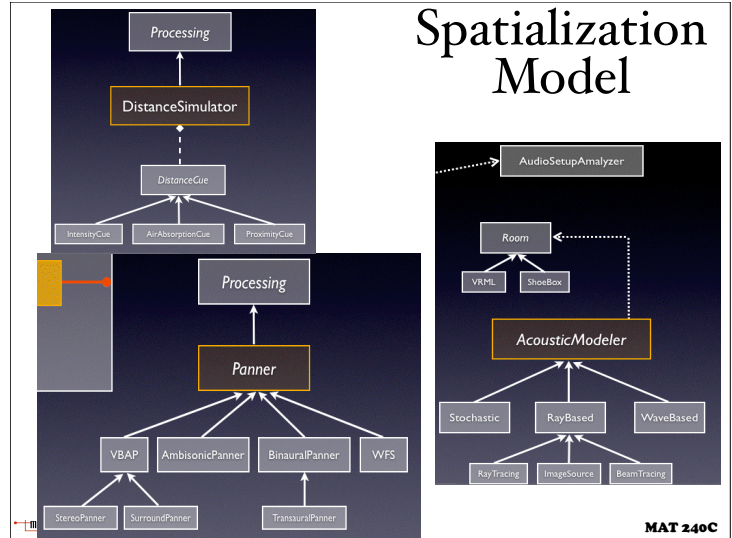
Spatialization Object Model

- Spatialized source (buffer + geometry)
- Speaker layout (position + model)
- Room model (geometry + materials)
- Processors:
 - Panner
 - Reverberator
 - Distance cue
 - Spatializer

MAT 240C

201

Spatialization Model



MAT 240C

202

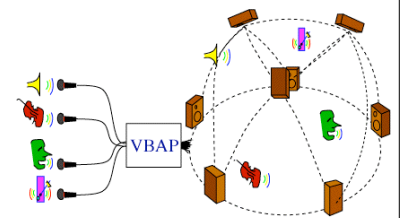
Alternatives

- How are sound and geometry bundled, streamed, synchronized?
- Are panning & spatialization the same?
- How is reverb integrated with spatialization, if at all?
- Is a distance cue built-in the the process?
- What metamodel is used?

MAT 240C

203

Vector-Based Amplitude Panning



- VBAP
 - Assume speakers located on the surface of a sphere
 - Assume monophonic sound sources and 3D positions outside of the sphere
 - Pan sources between the 3 nearest speakers

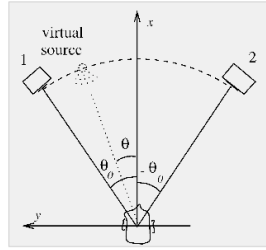
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

204

Traditional Stereo Panning

- Gains applied to each speaker are dependent on angle θ as described by a variety of panning laws, i.e.



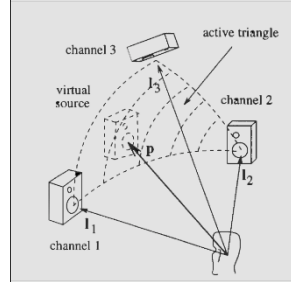
$$\frac{\sin(\theta)}{\sin(\theta_0)} = \frac{g_1 - g_2}{g_1 + g_2}$$

$$\frac{\tan(\theta)}{\tan(\theta_0)} = \frac{g_1 - g_2}{g_1 + g_2}$$

MAT 240C

205

Vector Base Amplitude Panning



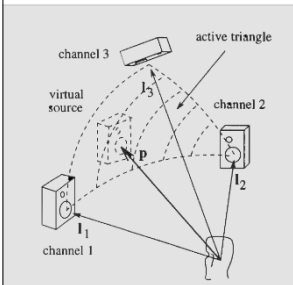
- Vector P points to the position of the virtual source.
- Vectors l_1 , l_2 , and l_3 point to the speakers on channel 1, 2, and 3, respectively.

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

206

Vector Base Amplitude Panning



$$\vec{P} = g_1 \vec{l}_1 + g_2 \vec{l}_2 + g_3 \vec{l}_3$$

- Vector P can be expressed as a linear combination of the vectors l_1 , l_2 , and l_3
- We can solve for vector g whose coordinates are the gain factors to be applied to each loudspeaker.

MAT 240C

207

Vector Base Amplitude Panning

- The previous equation can be expressed in vector form.

$$\vec{g} = \vec{P}^T L_{123}^{-1}$$

$$\vec{P}^T = \vec{g} L_{123}$$

$$\vec{g} = [g_1 \ g_2 \ g_3]$$

- where

$$L_{123} = \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix}$$

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

208

Vector Base Amplitude Panning

- For systems with more than one triplet of speakers
 - Proper triplet must be identified for each source before VBAP calculation
 - Proper triplet found by searching through triplets and attempting to calculate gain (fancy techniques available for this)
 - Triplet that offers solution with no negative gains is the proper triplet

MAT 240C

209

Limitations of VBAP

- VBAP comes with inherent limitations
 - Assumes unit vectors as source position
 - All sources confined to the surface of a sphere defined by loudspeaker array
 - Distance attenuation can be added for sources outside the sphere
 - Sources inside the sphere need some other cue to render closeness
 - The solution to this problem requires significant work beyond what is currently implemented

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

210

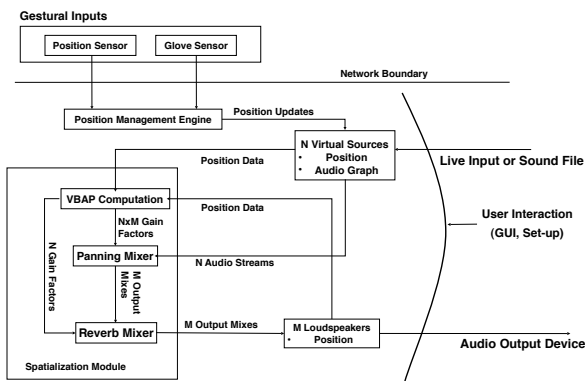
Limitations of VBAP

- Number of outputs needed for effective spatial rendering
 - Practical lower bound
 - VBAP can only render sources in the area spanned by the loudspeakers
 - Need enough channels to surround the listener

The VBAP Process

- Given a source stream, desired position, and speaker position data:
 - Evaluate which sector of loudspeakers the source is within
 - Compute gains for each channel according to source position
 - Gains used by audio system to mix each source accordingly into correct output channel
- VBAP in CSL code: Source and VBAP classes + CSL wrappers

Ventriloquist Architecture



Using VBAP in the Real World

- Must generate distance cues separately
 - Wet/dry reverb ratio, distance filtering, doppler shift
- No way to place sources inside the sphere
 - Spread source over > 3 speakers?
- Assumes identical speakers equidistant from the listener with regular density pattern
 - Separate weight table?
- Integrate with spatial reverberator; apply VBAP to all early reflections

Scaling/Distributing VBAP

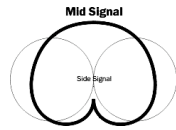
- Many speakers: geometrical search requires fancier techniques (solved)
- Many sources: each source maps to 3 neighboring speakers (in theory)
- Lots of motion: recompute weighting vectors on source (or listener) motion
- Distribution: easy, based on geometrical grouping
- Integration with spatial reverb explodes number of sources to pan

VBAP Implementation

- Several open-source implementations
 - Max, SuperCollider, C, etc.
- Doug McCoys MAT thesis: VBAP in CSL
 - CSL UGen class that takes inputs and positions
 - Uses fixed format speaker configuration file
 - App includes interactive control with data glove
- Refined/extended/integrated by Jorge C.

Ambisonics

- Primarily a representation for spatial sound based on spherical geometry
- Coupled with a playback mapping technique
- Loosely coupled to various recording/processing techniques (M/S)
- Very powerful and flexible (basic manipulations are simple)



MAT 240C

217

The Basics

- Represent a spatialized signal as a mono signal and 3 directional ones (x/y/z)
- Relationship to M/S stereo
- Basic (B format) equations
 - Source at azimuth A elevation B

$$W = \text{input signal} * 0.707,$$

$$X = \text{input signal} * \cos A * \cos B,$$

$$Y = \text{input signal} * \sin A * \cos B,$$

$$Z = \text{input signal} * \sin B,$$
- These correspond to the first-order spherical harmonics

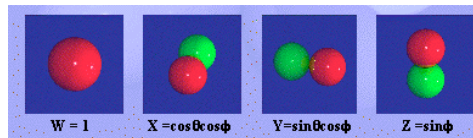
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

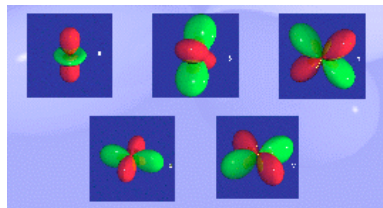
218

Spherical Harmonics

- First order



- Second order

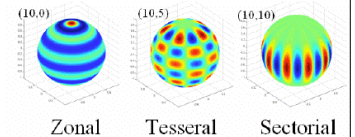


MAT 240C

219

Channel and Order

- Example 10th-order harmonics
 - Zonal
 - Tesseral
 - Sectorial
- In general, $L \gg N = (M + 1)^2$
 - where:
 - M = Ambisonic order
 - N = number of channels in Mth-order
 - L = number of meaningful output channels



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

220

Ambisonic Playback

- For an arbitrary (2D or 3D) loudspeaker configuration, take the 4 B-format signals and create the individual outputs, which is relatively trivial.
- It's much better if the layout is regular.
- There are HW decoders that do this
- There are SW...

MAT 240C

221

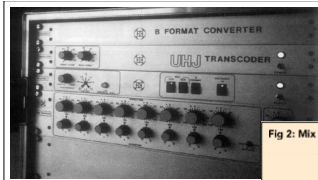
Comparison with Surround Sound

Ambisonics	5.1
Only requires four channels.	Needs six channels.
Includes height information.	No height information.
Only one mix required for DVD.	At least two mixes required for DVD.
Lower data density: lossless compression can be used.	Higher data density: lossy compression needed.
Imaging equally accurate in all directions.	Imaging only accurate across front stage.
Imaging possible between the speakers.	Almost non-existent inter-speaker imaging.
Sounds can be placed within the array.	Sounds appear around the edge of the array.
Speakers can be placed almost anywhere.	Speakers must be in special positions.
'Sweet spot' large enough to enjoy image outside the array.	'Sweet spot' only covers a small area: image changes as you move

MEDIA ARTS & TECHNOLOGY PROGRAM

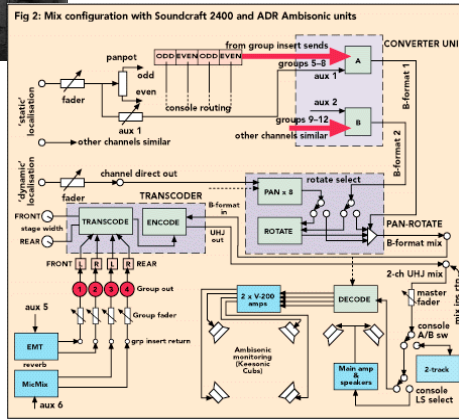
MAT 240C

222



Ambisonic HW

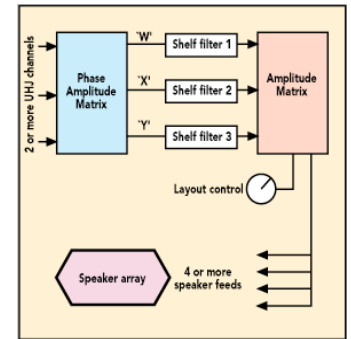
- Hardware for encoding, mixing, processing



223

Ambisonic “Matrix” Encoding

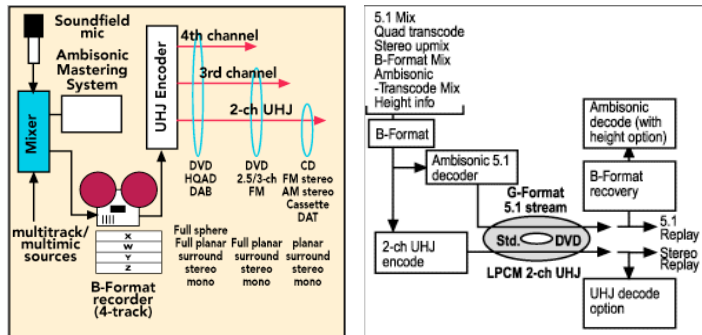
- Many format options based on B-format signals
- Look familiar?
 - FM radio
 - Matrix quad



MAT 240C

224

Ambisonics on DVD/SACD



- More formatting/matrix options

MAT 240C

225

Ambisonic Recording

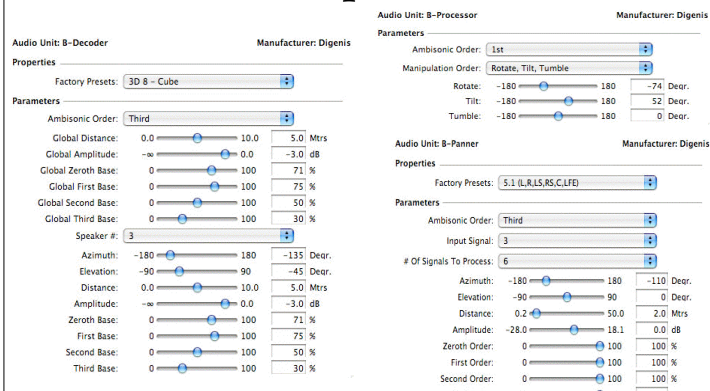
- Soundfield Microphone
- Many other techniques to derive B-format from multi-microphone recordings



MAT 240C

226

Software Implementations



- AU Plugins (also in VST, Csound, SC, etc.)

MAT 240C

227

Ambisonics Implementations

- Open-source in Csound, SuperCollider
- Ambisonics in CSL by Wakefield et al.


```
HOA_Encoder encoder1(source1, 3, 0.0, 0.0);
HOA_Encoder encoder2(source2, 3, 0.0, 0.0);
HOA_Mixer ambiMix(encoder1);
ambiMix.addInput(encoder2);
HOA_Rotator rotator(ambiMix);
rotator.setRotate(0.f);
HOA_SpeakerLayout speakerlayout;
HOA_Decoder decoder(rotator, speakerlayout, HOA_PROJECTION);
gIO->set_root(decoder);
```

MAT 240C

228

Using Ambisonics

- Again, no special handling of distance cues
- Encoding is independent of decoding
 - Intermediate format has $(M+1)^2$ channels
- Some hardware decoders in use
- Assumes regular speaker configuration
- Very dense arrays benefit from higher-order encoding
 - e.g., 10th-order [121 channels] decoded out to 500 or more speakers

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

229

Evaluation

- Encodes sound geometry rather than speaker signals
- Can be extended to higher-order signals (improves separation and size of sweet spot)
- Spatial processing possible
- Supports different output formats and integration with other formats and techniques

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

230

Scaling/Distributing Ambisonics

- Many speakers: decoders are (relatively) simple
- Many sources: encoders can be separated, stream $(M+1)^2$ channels to decoders
- Lots of motion: no added cost for motion
- Distribution: both encoders and decoders can easily be distributed, but must stream all channels between one another (no locality of output)

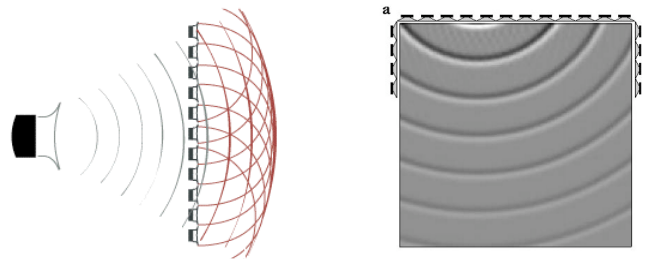
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

231

Wave-field Synthesis (WFS)

- Huygens principle (superposition): many closely spaced speakers can create a coherent wavefront for an arbitrary source position



MAT 240C

232

The WFS Promise

- Wavefront reconstruction allows for arbitrary source placement
- Sources can be within the volume
- There's no sweet spot!
- Limited-range (inexpensive) speakers assumed
- Arbitrary source movement at no extra cost
- See <http://www.iosono-sound.de/applications.html>

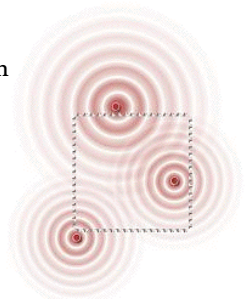
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

233

WFS Assumptions

- Theory: model sound pressure and velocity at every point in a volume
- Simplify:
 - Volume extends to ∞ in 1 direction
 - Simplifies sphere to a plane
 - No sources within volume
 - No elevation cues
 - Simplifies plane to a line
 - Only need velocity or pressure
 - 2.5-Dimensional WFS



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

234

To make WFS work...

- Compute delay time, phase shift, and amplitude scale for each speaker: solve Kirchoff/Helmholtz integral

$$P_A = \frac{1}{4\pi} \oint_S \left[\left(P \frac{1 + jk\Delta r}{\Delta r} \cos\phi \frac{\exp(-jk\Delta r)}{\Delta r} \right) + \left(j\omega\rho_0 V_n \frac{\exp(-jk\Delta r)}{\Delta r} \right) \right] dS$$

- Possibly correct spectrum for room effects
- Implementations are complex, but scalable (up to 48 channels on a single CPU)

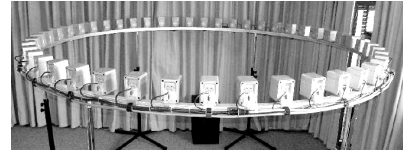
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

235

WFS Speaker Density

- Wavefront creation depends on spatial aliasing
- Speaker spacing determines highest frequency that can be accurately placed (< 20 cm desirable)
- Range between the limit of spatial hearing (200-300 Hz) and the speaker density spatial alias freq. (>= 2 octaves good)
- 1.25 cm (5 inches) leads to upper freq. of ~1.4 kHz
- Wider spacing \Rightarrow lower upper bound



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

236

Speaker Layout for WFS

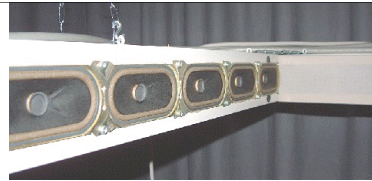
- WFS assumes dense placement of identical speakers in the horizontal plane
- Assumes identical point-source speakers
- Can be altered to bi-polar speakers
- 3D WFS has not been attempted
 - Solve 3D KH integral using fast multipole methods as at UMD

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

237

WFS Spaces



- Small set-ups: 32-64 channels on 1 CPU
- 198-channel cinema in Ilmenau, Germany
- IOSONO systems



MEDIA

240C

238

IOSONO WFS System



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

239

MAPs for WFS

- Multi-actuator (8-32) panels
- Require DSP to handle resonances and interactions



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

240

WFS Implementation

- Configuration, set-up
 - Speaker layouts, sources, room characteristics
- Real-time convolution
 - Distributed FFTW via BruteFIR
 - Scene/score playback and sample mgmnt.
 - Streaming I/O over network
 - Output server(s)

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

241

WFS Scalability & Distribution

- Many speakers: desired, players scale well
- Many sources: linear computational load
- Lots of motion: no extra load
- Distribution: very good using BruteFIR or similar package to plan and distribute massive amounts of FFT-based convolution

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

242

Spatializer Control and GUIs

- Spatial sound applications
- Spatializer control
 - Trajectory files and scheduler
 - Interactive mixing and positioning
 - Stateful and modal interfaces
- GUIs for spatial sources
 - Ventriloquist, Zirkonium, SWonder, IOSONO
- Spatial sound scores and editors



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

243

Spatializer Control

- VR/VE systems
 - User head-tracking, headphone output
- Trajectory files and scheduler
 - Event = time, source-ID, ampl
 - posX/Y/Z or posR/Theta/Psi
- Interactive mixing and positioning
 - Faders, gloves, trackers
- Stateful and modal interfaces

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

244

GUIs and Spatializer Apps

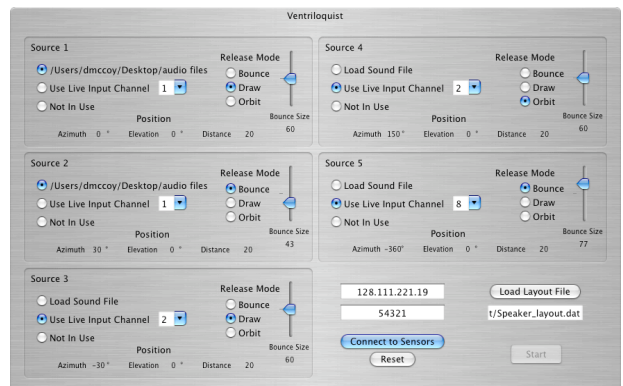
- Ventriloquist
 - Doug McCoy, CREATE
 - VBAP App with dataglove/FOB input
- Zirkonium
 - C. Ramakrishnan (ex-MAT), ZKM
 - Room editor
 - Spatial score editor
- Wonder
 - M. Baalman, TUBerlin
 - WFS scene editor



MEDIA ARTS & TECHNOLOGY PROGRAM

245

Ventriloquist GUI

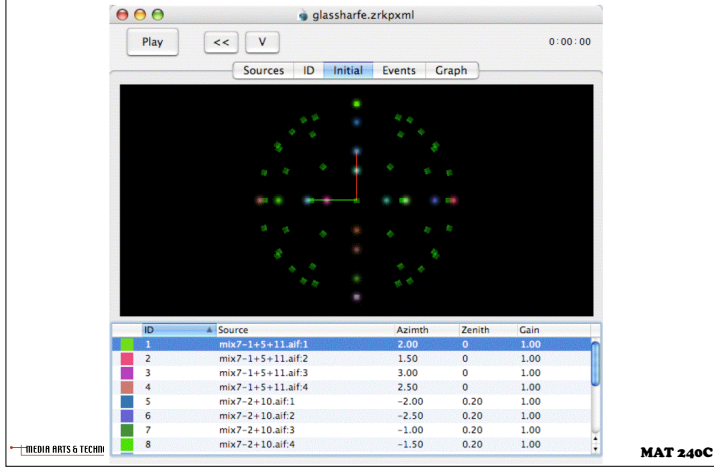


MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

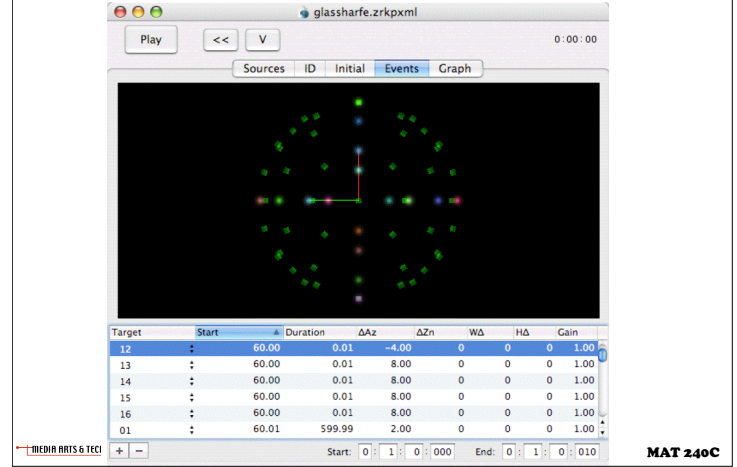
246

Zirkonium Config



247

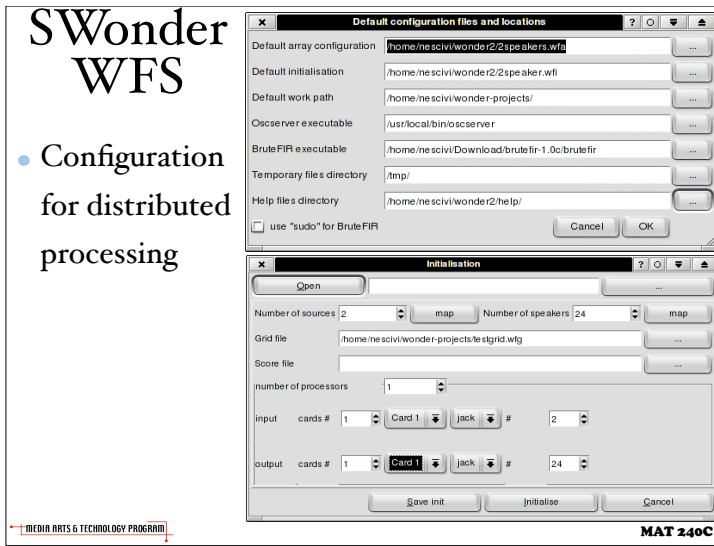
Zirkonium Score



248

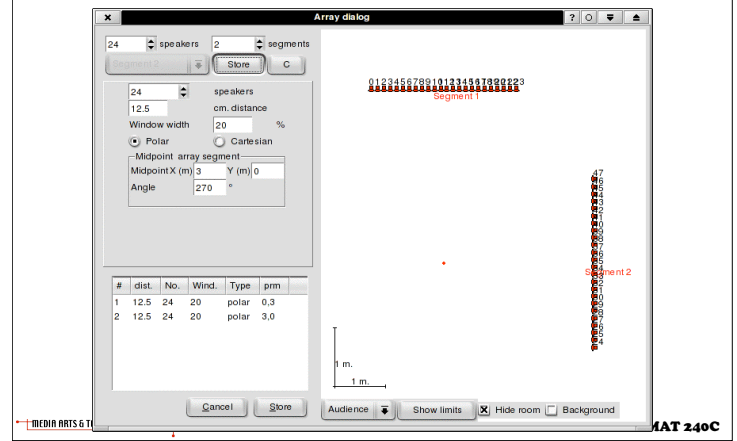
SWonder WFS

- Configuration for distributed processing



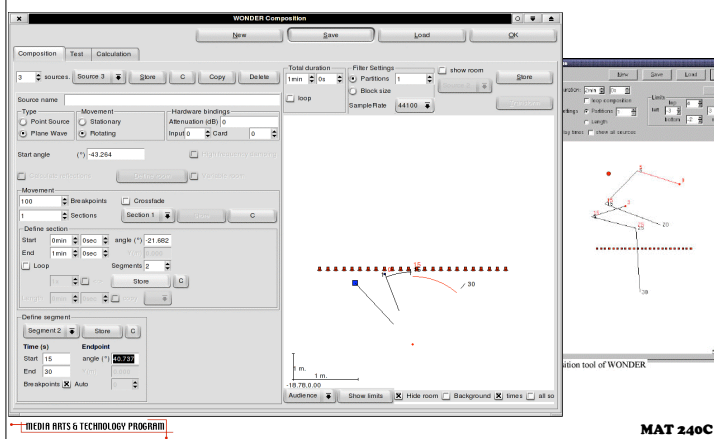
249

SWonder Layout Editor



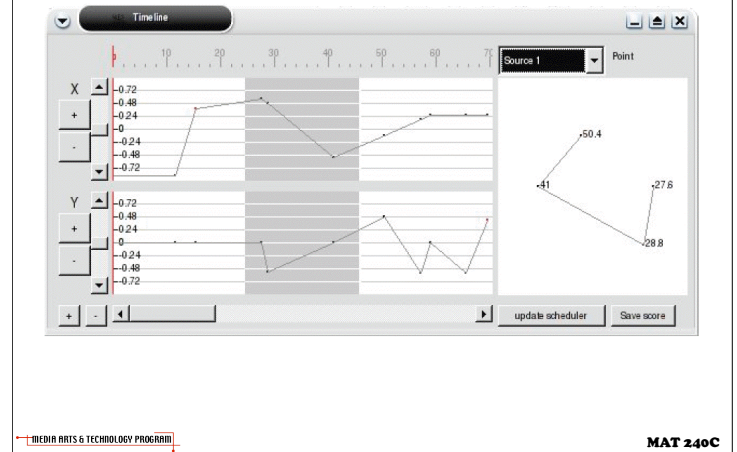
250

SWonder Composition



251

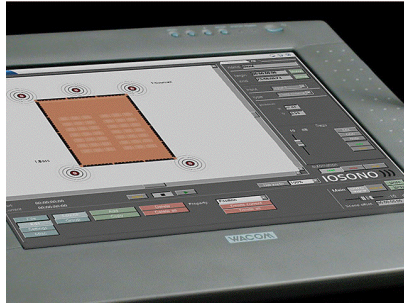
SWonder Editors



252

IOSONO Editor & Control

- Configuration
- Spatial score editor
- Scenes and playback



253

IOSONO HW

- DSP units
- MADI routing
- Server farm
- Speaker panels
- Control GUI



254

Review

- Spatialized/Auralized Sound
 - Vector-based amplitude panning
 - Ambisonic representation and rendering
 - Wavefield synthesis
- Spatializer apps and control

MAT 240C

255

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C Digital Audio Programming: Spatial and Surround Sound Topic 7: Applications

MAT 240C

256

Topic 7: Applications

- Playback systems
- Reverberators
- Tracking and control
- Plug-ins
- The AlloSphere

MAT 240C

257

Topic 7 Readings

- Applications and the AlloSphere
 - AlloSphere Audio Design
 - Zirkonium
 - Ventriloquist
 - Wonder

MAT 240C

258

Applications

- See project descriptions above
- Matrix playback SW
- 2-to-5.1 up-mix app.
- Reverberator plug-ins
- HRTF for headphones
- Spatializers
- CNSI AlloSphere introduction

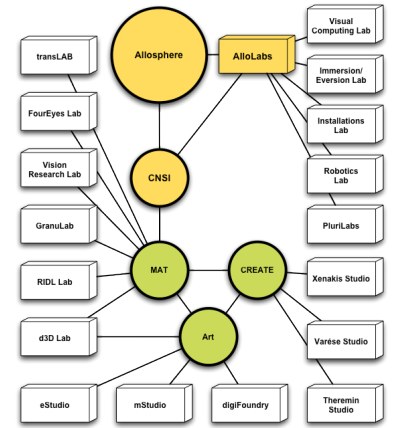
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

259

CNSI at UCSB

- Building facilities: clean rooms, wet labs, materials labs, etc.
- MAT labs in CNSI: sound and image synthesis, processing, rendering, media networking, digital classroom, media post production, robotics, installation space, etc.

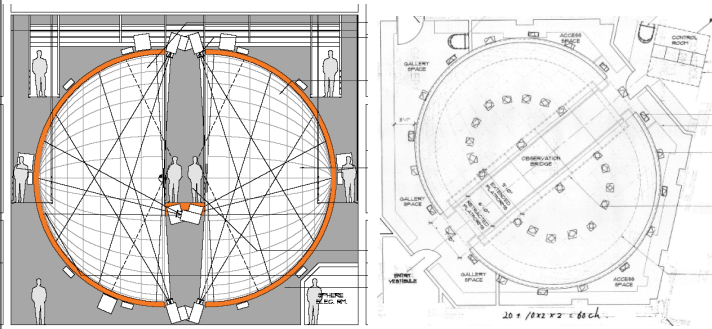


MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

260

The UCSB AlloSphere



- 14 video projectors • 400+-channel surround sound
- Multi-camera video capture • Spatial microphone array

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

261

The AlloSphere



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

262

Applications

- Immersive interactive interfaces to numerical simulations
- Multi-user virtual environments with multi-modal user sensing/tracking
- Data mining with flexible rendering
- Scientific visualization and sonification
- Sound/video projector development
- Immersive multimedia art works

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

263

Requirements

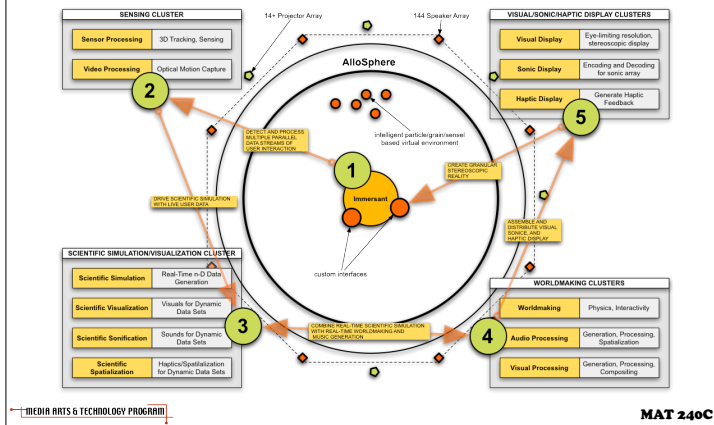
- 360-degree high-resolution video projection onto the inside of a sphere (eye-limited resolution)
- Many-channel (≥ 400) immersive 3D surround sound playback (ear-limited resolution)
- Computer vision-based sensing input
- Spatial microphone array
- Various mechanical/magnetic/audio input options
- Useable by 10-20 people at a time

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

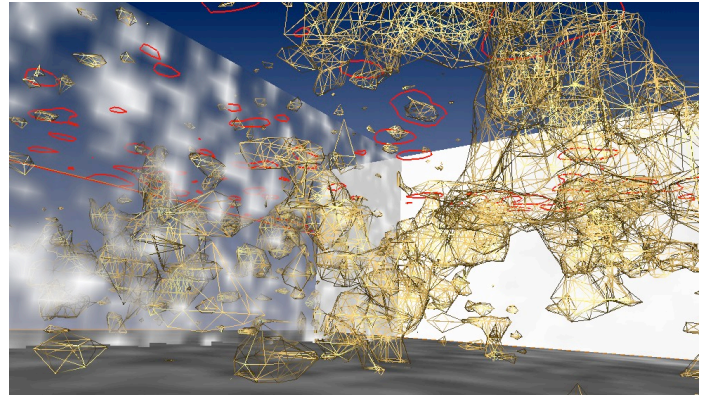
264

Application Flow



265

Data Mining and Navigation



266

CNSI Computational Infrastructure

- Traditional vector supercomputer
 - Runs (older FORTRAN) numeric/simulation applications
 - MPI software framework for parallel computation
- Large Linux cluster
 - Runs (modern cluster-oriented) scientific applications
 - MPI and other cluster/grid application managers
- Multimedia processing cluster (media IO farms)
 - Input sensing, feature extraction, gesture/voice recognition
 - Media rendering
 - Media output to sphere
- CRAM distributed application manager

MAT 240C

267

Media Rendering and Data I/O

- Video
 - 9- to 14-channel overlapping video output on inside of sphere = ~140 Mpixels (eye-limited resolution)
 - Frame-sync warped projection
- Audio
 - 400-500-channel high-resolution sound output
 - HiRes on FireWire or 1000BaseT Ethernet
 - Bandwidth: $256 * 24/96 \approx 620$ MBit/sec
- Sensing and Input
 - Camera-based video input sensing and analysis
 - Audio input via spatial microphone array
 - Mechanical and magnetic sensors on user bridge

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

268

AlloSphere Infrastructure

- AlloSphere Media servers
 - GestureSensor matrix: computer vision and sensor mapping
 - OpenGL renderer stack: output of complex immersive models
 - Sonification/Spatialization servers: sound synthesis and spatial processing
 - Content data stores: media databases
- Run-time applications managed by the CRAM distributed application manager

MAT 240C

269

SW Development, Deployment

- Use off-the-shelf tools where possible:
 - OpenSceneGraph for 3D models, world-building, and interaction models
 - OpenGL for low-level graphics rendering
 - CSL for audio synthesis and spatialization
 - VRPN/GestureSensor frameworks for sensor input management
 - OSC protocol for control networking
 - CRAM for application description, system mgmnt, load-balancing, and fault-tolerance

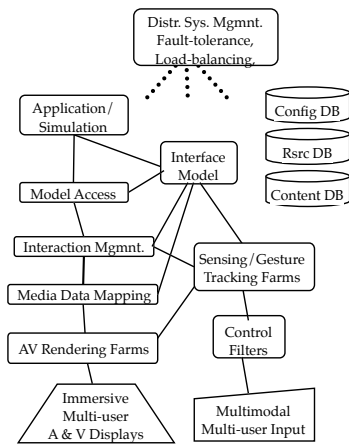
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

270

AlloSphere Software Management

- Distributed Sensing, Computation, and Projection = DSCP (MVC++)
- Back-end application models are scientific/numerical/simulation
- Multimodal multiuser sensing/control and tracking/mapping farms
- Application = sensing/tracking policies + output data mappings
- Presentation/interaction via HoloSphere, LAN/WAN streaming
- DBs for configurations, resources, and media content (renderers)

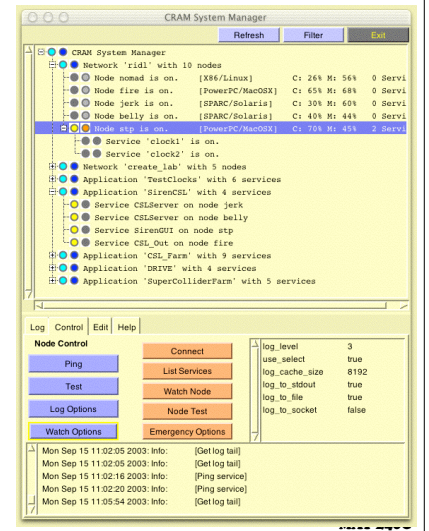


MAT 240C

271

CRAM Manager

- Network/Node
- Node/Service
- Application/Service
- Log/control pane



272

A Typical AlloSphere Application

- Input sensing (camera, sensor, microphone)
- Gesture recognition/mapping
- Application model (numerical, simulation, arts content, data mining/navigation)
- Back-end processing (data/content accessing)
- Output media mapping (visualization and/or sonification)
- Rendering (A/V)

MAT 240C

273

AlloSphere Audio Requirements

- “Ear-limited” audio rendering
 - Freq. response: 20 -18000 Hz
 - Dynamic range: 0 - 120 dB
 - Signal-to-noise ratio: > 80 dB
 - T60 time: < 0.75 sec.
 - Spatial accuracy > 30 in the horizontal plane and 100 in elevation
- Support for mainstream audio synthesis/processing software (SuperCollider, CSL, CLAM, Max, etc.)

MAT 240C

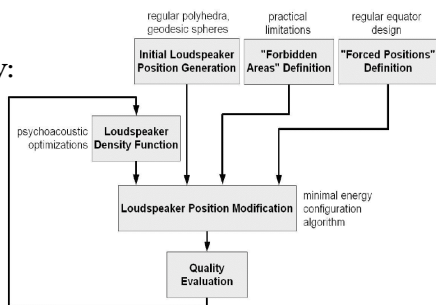
274

AlloSphere Speaker Configuration

- Determining the number of speakers required
- Designing a speaker layout

Sphere geometry:

- 4.75m radius
- 35m circumference
- 364 m2 area



MAT 240C

275

How many speakers do we need?

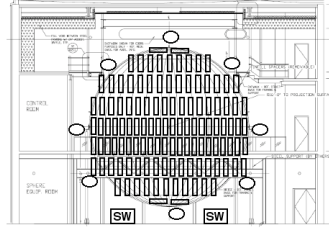
- VBAP: to get 100 resolution ~ 364 speakers on 1m centers
- Higher-order Ambisonics: > 256 speakers with a smooth vertical density function
- 2D WFS: ~ 150-200 speakers around the equator
- 3D WFS: 15cm spacing (1 kHz upper range) = 9100 speakers

MAT 240C

276

How to arrange them?

- Symmetry around the vertical axis
- Door issues and other forbidden zones
- Vertical density function
- Equator rings
- Upper/lower rings
- Subwoofers?



MAT 240C

277

Loudspeaker Technologies

- Speaker design:
 - Configuration (n-way)
 - Driver technology (dynamic, electrostatic, ribbon, piezo-electric, etc.) for one or more elements
 - Cross-over filter(s)
 - Driver mount, motor board
 - Speaker enclosure
- Almost all possible options are found on the market!

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

278

AlloSphere Requirements

- Directionality desired -- easier to handle absorption of "stray" energy and back-wave
- Weight/vibration issues -- cannot attach the sphere screen mounting, must hang elements
- Speaker as network element -- avoid runs of analog wiring, make stand-alone network node for each speaker or group of speakers

MAT 240C

279

Driver Technology Options

- Dynamic drivers
 - ++ cheap, wide range, easy to drive
 - -- heavy, omnidirectional, fragile
- Electrostatic drivers
 - ++ lightweight, thin, bi-directional, robust
 - -- require hi-V bias, large surface to get low frequency extension
- Others...



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

280

Speaker System Designs

- Electrostats, dynamic tubes, ribbons + tweeter arrays

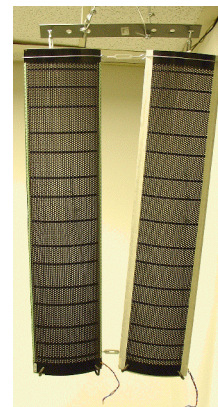


MAT 240C

281

Speaker Mounting

- Example: hanging ESL panels behind the sphere's surface



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

282

Network Interface Box

- 1000BaseT (and/or FireWire1600) interface
- Control logic (FPGA), input buffering, demux, etc.
- Each with 16-128 channels of:
 - Digital-to-analog convertor (or digital amp)
 - Crossover/compensation filter (optional)
 - 50 watt (at 2 Ω) Class D amplifier
 - Step-up transformer (for ESL voltages)

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

283

Audio Output

- 500-channel spatializer in CSL
 - Map monophonic sources + geometric placement models (as well as stereo, 5.1, etc.), onto sphere array using any of our favorite techniques (VBAP, Ambisonics, WFS, etc.)
 - Allow dynamic switching between diffusion, distance, room models
 - Flexible GUI for pan, image spread, diffusion
- Output distribution servers
 - 512-channel FireWire/100BaseT streaming
 - 2 or more servers using S. Buttner's EtherSync cards (90 nsec clock sync)

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

284

Current Sphere Setup

- (Subject to change)
- Intel PowerMac servers
- MOTU FireWire outputs
- Mackie speaker ring
- Plurisonic Lab
- PluriLab studio

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

285

Audio Input Processing

- Source localization from microphone array
- Speaker ID (voiceprints)
- Feature extraction for speech understanding
- Melodic analysis for "query by humming"
- Capture/map input from hardware control surface
- Hardware input/user sensing via touch screens, trackers, MIDI/OSC, reacTable

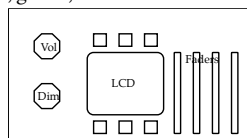
MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

286

Sphere Sensing Details

- Camera Array
 - 4 pan/tilt/zoom cameras mounted in sphere
 - Stream to computer vision feature extraction
- Microphone Array
 - 8 * 4 matrix in sphere surface front/above
 - Stream to sound/speech audio analysis/tracking
- Sensor Inputs
 - OSC and MIDI for sensors, control surface, gloves, etc.
 - ASCII keyboards
- Control Surface
 - Light dimmer and volume knobs, faders
 - LCD with buttons for main menus



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

287

Unanswered Questions

- Distance cues for VBAP
- Sources inside the sphere for VBAP
- Distance cues for Ambisonics
- Scaling-up of VBAP, HOA *
- 3D WFS implementation using FMM math *
- Streaming protocols for 500 channels (+ video) over 1000BaseT, FireWire, etc. *
- Design/construction of network element/interface
- Audiophile-grade digital amplifier
- System integration, CRAM manager (LB, FT) *

MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240C

288

Where do we go from here?

- Projects
 - Development
 - Integration
- Applications
 - Delivery
- AlloSphere
 - Design and Deployment

Review

- Applications
 - Development
 - Dissemination
- AlloSphere
 - Design
 - Infrastructure
 - Application development

MAT 240C Topics

- 1: Spatial Sound, Spatial Hearing
- 2: Recording and Representation
- 3: Matrix Playback Processors
- 4: Reverberation and Convolution
- 5: Head-Related Transfer Functions
- 6: Spatialization
 - VBAP, Ambisonics, WFS, systems and control
- 7: Applications and the UCSB AlloSphere

MAT 240C: Digital Audio Programming: **Spatial and Surround Sound**

The End

MAT 240D - Digital Audio Programming: Sound Synthesis Techniques (Fall, 2007)

The MAT 240 course sequence is a six-part (two-year) practical programming course; it consists of hands-on software development devoted to digital audio and multimedia applications. Students read a selection of papers from the literature, with the emphasis on learning to use and extend the current state-of-the-art programming methods, tools, and programming interfaces. Class assignments involve C/C++/Java programming on Linux, Macintosh, MS-Windows, various plug-in APIs, and other platforms.

In MAT 240D, we will implement a variety of software sound synthesis techniques, starting from traditional additive, subtractive, to non-linear synthesis. Our focus, however, will be on more contemporary techniques such as physical models and granular synthesis. We will explore the internals of several existing synthesis packages and write our own versions of selected techniques in the CSL framework in C++ (<http://create.ucsb.edu/CSL>).

Students are expected to know the basics of digital audio signal representation and processing, and to be proficient in C, C++, or Java (Smalltalk, SuperCollider or LISP are a plus). Grading will be on the basis of in-class participation and programming projects.

Course Outline

- Sound synthesis and processing background
- APIs and frameworks for digital audio synthesis
- Additive synthesis and the FFT
- Wavetable oscillators and optimizations
- Subtractive synthesis and dynamic filters
- Nonlinear techniques: FM and wave-shaping
- Sample-based synthesis and processing
- Physical models: waveguides and simulations
- Chaos and novel techniques
- Building applications and GUIs

Instructor

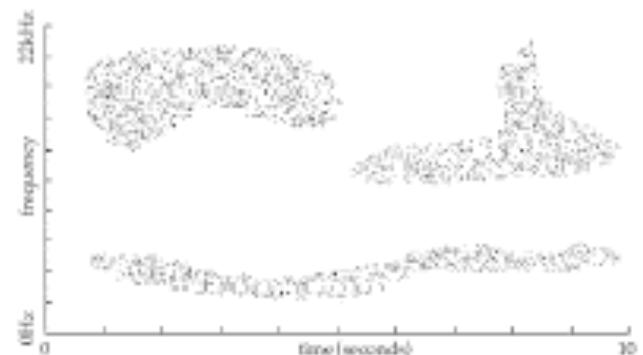
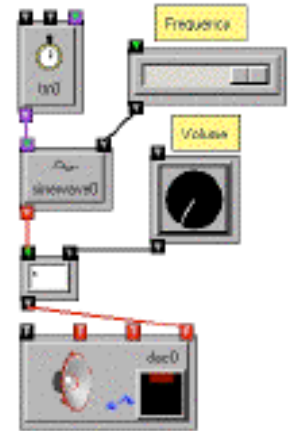
- Stephen T. Pope (stp@mat.ucsb.edu)

Meeting time and place

- T/Th 11:00 AM - 1:00 PM, Music 2215

Electronic Resources

- Course Web Site
See <http://create.ucsb.edu/240>
- Email Mailing List
See <http://www.mat.ucsb.edu/mailman/listinfo/240> to join



MAT 240D Reader Contents

Signal Processing Aspects of Computer Music: A Survey, J. A. Moorer (DASP book)
Chapter 7: Physical Modeling and Formant Synthesis. Curtis Roads (CM Tutorial book)
Granular Synthesis. Curtis Roads (Foundations book)
Digital Waveguide Modeling of Musical Instruments J. O. Smith (www)
Synthesis without Lookup Tables, James McCartney (CMJ)

SoftSynth Catalog pages, Full Compass Fall 2007 Catalog

Clear, Efficient Audio Signal Processing in ANSI C. Adrian Freed (www)
JSyn Programmer's Guide. Phil Burk (www)

The CREATE Signal Library ("Sizzle"): Design, Issues, and Applications. Stephen Travis Pope and Chandrasekhar Ramakrishnan (ICMC 2003)
Metamodels and Design Patterns in CSL4, Stephen Travis Pope, Xavier Amatriain, Lance Putnam, Jorge Castellanos, and Ryan Avery (ICMC 2006)

Wavetable Synthesis 101: A Fundamental Perspective, Robert Bristow-Johnson
Effect Design: Part 3: Oscillators, Jon Dattorro (JAES)
Physically-Based Parameteric Sound Synthesis and Control, Perry Cook
CSE393: Computer Music & Sound Synthesis, Tom Cortina, SUNYB

Optional Readings

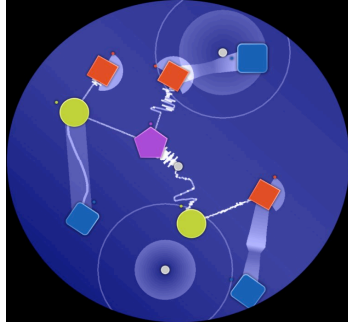
Granular Synthesis of Musical Signals. Sergio Cavaliere and Aldo Piccialli (MSP book)
PulsarGenerator Documentation. Curtis Roads and Alberto de Campo (www)
The Computer Music Tutorial. Curtis Roads.
The CSound Book. R. Boulanger
Three Languages for Software Sound synthesis, S. T. Pope (CMJ)



MEDIA ARTS & TECHNOLOGY PROGRAM

MAT 240D: Digital Audio Programming: Synthesis Techniques

Stephen T. Pope
stp@mat.ucsb.edu
Fall, 2007



MEDIA ARTS & TECHNOLOGY PROGRAM

1

Lecture 1 Outline

- * MAT 240 course series: digital audio programming (in C/C++/Java)
- * MAT 240D Goals
- * Course logistics
- * Written materials
- * Course overview
- * Topic 1
 - * Review: digital audio programming
 - * Basic synthesis techniques



MEDIA ARTS & TECHNOLOGY PROGRAM

2

MAT 240 Series

- * Hands-on programming courses using (primarily) C, C++, and Java for digital audio application development
- * Six-quarter (two-year) course series
- * Students use a variety of software development tools on MS-Windows, Linux/UNIX, and the Macintosh

MEDIA ARTS & TECHNOLOGY PROGRAM

3

MAT 240 Topics (6 quarters)

- * A: File I/O and Streaming Media
- * B: Spectral Transformations
- * C: Spatial Processing of Sound
- * D: Sound Synthesis Techniques
- * E: Control and Multi-rate Processing
- * F: Music Information Retrieval

MEDIA ARTS & TECHNOLOGY PROGRAM

4

MAT 240D Course Goals

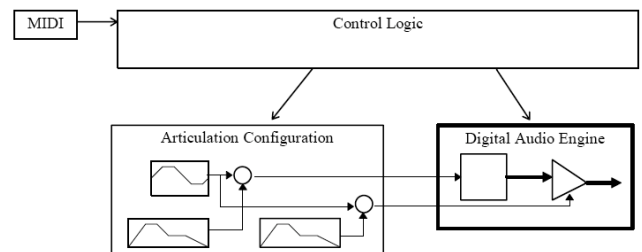
- * Implement all common sound synthesis techniques in C/C++/Java
- * Explore the design of object-oriented software frameworks for digital audio synthesis and processing
- * Build integrated applications



MEDIA ARTS & TECHNOLOGY PROGRAM

5

In Pictures



MEDIA ARTS & TECHNOLOGY PROGRAM

6

Course Logistics

- * **Lectures**
 - * Tues/Thurs 11:00 AM - 1:00 PM
 - * Music 2215
- * **Lab, work groups TBD**
- * **Grading**
 - * Class participation
 - * 2-3 in-class presentations
 - * Final “review”



MEDIA ARTS & TECHNOLOGY PROGRAM

7

Course Materials

- * **Presentation slides**
- * **Readings:** reader available in MAT class room, based on C. Roads Computer Music Tutorial, articles from Computer Music Journal and J. O. Smith's publications
- * **MAT 240D web site and links**
- * **Example code archive**
- * **Available software development tools**

MEDIA ARTS & TECHNOLOGY PROGRAM

8

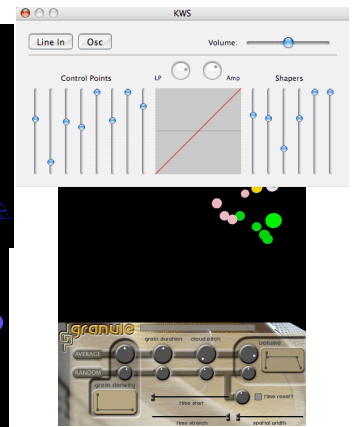
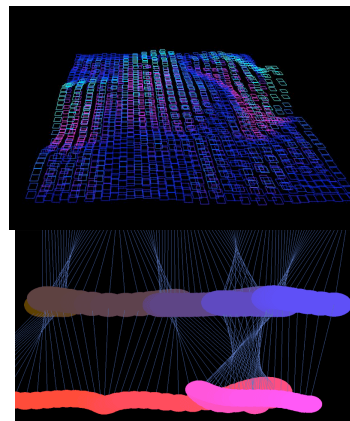
Course Topics

- * **Wavetables and additive synthesis**
- * **Vector synthesis and extended techniques**
- * **Introduction to DASP frameworks**
- * **FM and wave-shaping synthesis**
- * **Subtractive and filter-based techniques**
- * **Sample-based synthesis and processing**
- * **Granular synthesis, time-domain summation**
- * **Physical models, waveguides & modes**
- * **Chaos and novel techniques**
- * **Integrated applications**

MEDIA ARTS & TECHNOLOGY PROGRAM

9

MAT 240D Student Projects



MEDIA ARTS & TECHNOLOGY PROGRAM

10

Course Code Archive

- * **Example code**
- * **Class code**
- * **Frameworks**
- * **Updates**
 - * CSL
 - * CLAM
 - * CLM

[amber-1.1.2](#)
[CAST](#)
[CLAM-0.8.0](#)
[clm-3](#)
[CO_1.0](#)
[it++-3.8.1](#)
[jmusic-1.5](#)
[JSampler-0.1a](#)
[jsyn142_mac_osx_sdk](#)
[libgig-2.0.1](#)
[linuxsampler-0.3.3](#)
[loris-1.2.0](#)
[mxv-LATEST-source](#)
[NewCode](#)
[portaudio_18b](#)
[qsampler-0.1.2](#)
[SamplePlayer](#)
[Samples](#)
[stk-4.2.0](#)
[String0.2](#)
[wire10demo_mac_osx](#)
[240D_Tests.cpp](#)
[js-classic.jar](#)
[SamplePlayer.zip](#)
[saw_adsr.c](#)
[sharc.tar.gz](#)

MEDIA ARTS & TECHNOLOGY PROGRAM

11

Review: Digital Audio Programming

- * **Digital audio data**
 - * Sampling and quantization
 - * Data types for audio data
- * **Audio I/O APIs**
 - * See MAT 240A slides, web site, code
 - * Call-back functions and buffer handling
 - * Platform-specific: DirectX, CoreAudio, ALSA
 - * Cross-platform: PortAudio, SNDLIB, SDL, OpenAL, LibSndFile
 - * Language-specific: JavaMF, Siren, CLM

MEDIA ARTS & TECHNOLOGY PROGRAM

12

Sound Synthesis Background

- * Physics and acoustics
 - * Source/filter model
 - * Fourier theorem
 - * Sampling and quantization
- * Musical instrument design
 - * Sources and resonators
 - * Families of instruments
 - * Historical Western instruments
 - * Control and sound production
 - * Triggers and continuous control
 - * Multiplicity of control

MEDIA ARTS & TECHNOLOGY PROGRAM

13

Sound Models and Representation

- * Sound Properties
 - * Time-domain
 - * Sound waveforms and the time-domain signal
 - * Time-varying envelope and control functions
 - * Frequency-domain
 - * Spectrum and timbre
 - * Dynamicity and transients
 - * Spatial-domain
 - * Source/room model and its limitations

MEDIA ARTS & TECHNOLOGY PROGRAM

14

Taxonomies of Sound Synthesis

- * ...so many to choose from
- * Underlying signal model
- * Analysis/synthesis technique pairs
- * Physical model family
- * Analog synthesizer models
- * See:
 - * Roads (CMT)
 - * Loy (RepMusSig)
 - * Pope (MusProc)
 - * Others...

MEDIA ARTS & TECHNOLOGY PROGRAM

15

Sound Synthesis History

- * 1945ff: analog synthesis/processing
 - * Option 1: *Elektronische Musik*
 - * Option 2: *Musique Concrète*

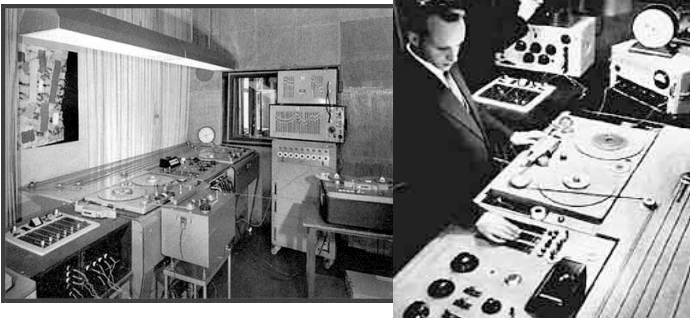


MEDIA ARTS & TECHNOLOGY PROGRAM

16

Ah, the 1950s

- * WDR, Köln



MEDIA ARTS & TECHNOLOGY PROGRAM

17

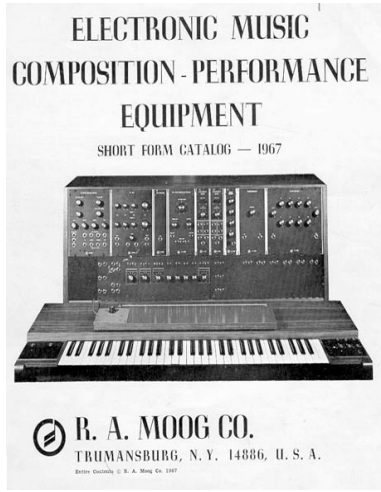
1960s: Voltage-Controlled Analog

- * Moog, ARP, Buchla, EMS, etc.
- * VC Sources
 - * Multi-waveform oscillators, noise sources
- * VC Processors
 - * Resonant LP filters, amplifiers
- * Control
 - * Envelope Generators
 - * Control inputs
- * Issues of connectivity and patching

MEDIA ARTS & TECHNOLOGY PROGRAM

18

Early Analog Synths (Moog, 1967)



MEDIA ARTS & TECHNOLOGY PROGRAM

19

Moog 55 (1973)



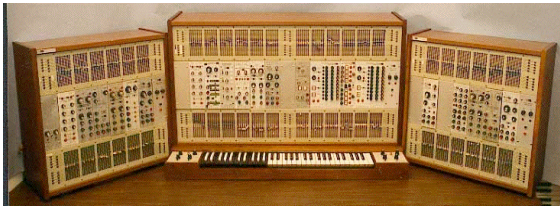
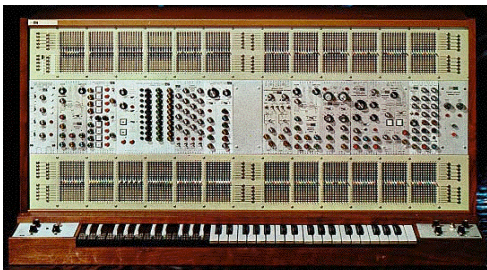
MEDIA ARTS & TECHNOLOGY PROGRAM

20

ARP 2500 (1970)

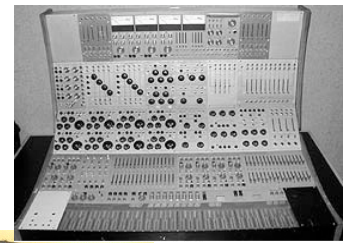
Sliders for
patching

Multi-unit
modules



21

EMS VCS3, Roland 700, Buchla 200



MEDIA ARTS & TECHNOLOGY PROGRAM

22

EMS Synthesi 100

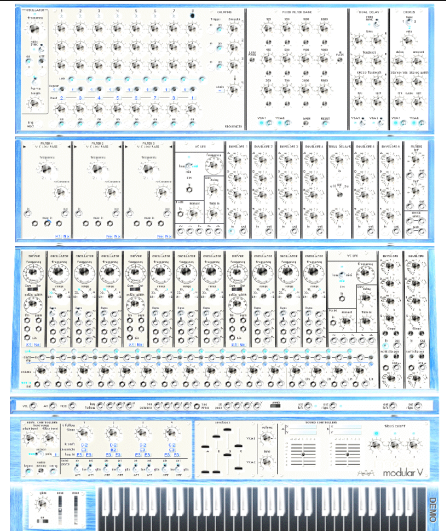


MEDIA ARTS & TECHNOLOGY PROGRAM

23

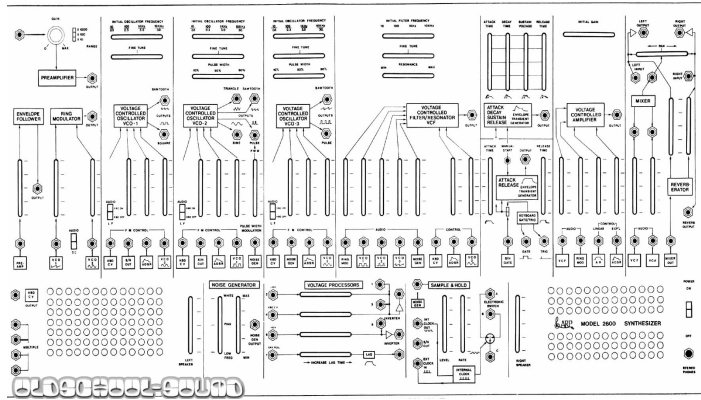
Moog Series III

* As
emulated
by Arturia
plug-in
soft-synth



24

ARP 2600 (1971)

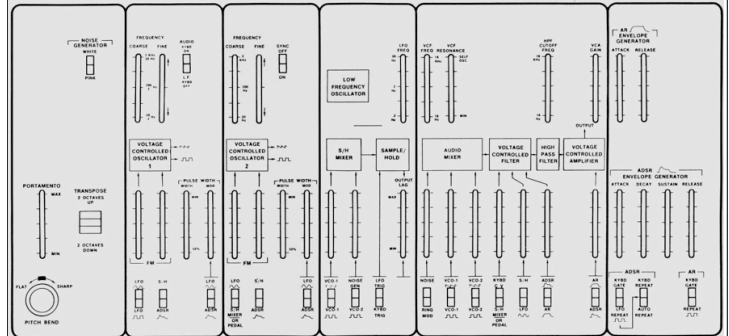


Default patch: 5 srcs, mxr, VCF, VCA, A/D/S/R

— MEDIA ARTS & TECHNOLOGY PROGRAM

25

ARP Odyssey (1973)



ODYSSEY
Facsimile Pad 25 Sheets
Use for a written record of your built sounds. To reorder,
see your Dealer or contact ARP Instruments, Inc., 320
Needham Street, Newton, Mass. 02164. (617) 965-9700

Fixed/switchable patch
Patch Title:
2VCOs, noise/RM, VCF/VCA
Comments:

— MEDIA ARTS & TECHNOLOGY PROGRAM

26

Computer Sound Synthesis

- * 1950s: “acoustical compiler” project at Bell Labs
- * 1960s: Music-N languages
 - * Model of unit generators and buffers
 - * Music-V is portable (written in FORTRAN)
 - * Published in MVM’s 1969 book
 - * Various source and processor UGens
 - * No real-time interaction
 - * “Composition routines” as compiler 1st pass
- * 1970s: Music-N family on mainframes and UNIX

— MEDIA ARTS & TECHNOLOGY PROGRAM

27

SWSS Today

- * Sound compilers
 - * Csound, SuperCollider, Chuck
- * DASP APIs
 - * CSL, CRAM, JSyn, Cmix, CLM, Siren
- * Soft-synths (VSTi...)
 - * Samplers
 - * Analog-like
- * Max/MSP/Pd

— MEDIA ARTS & TECHNOLOGY PROGRAM

28

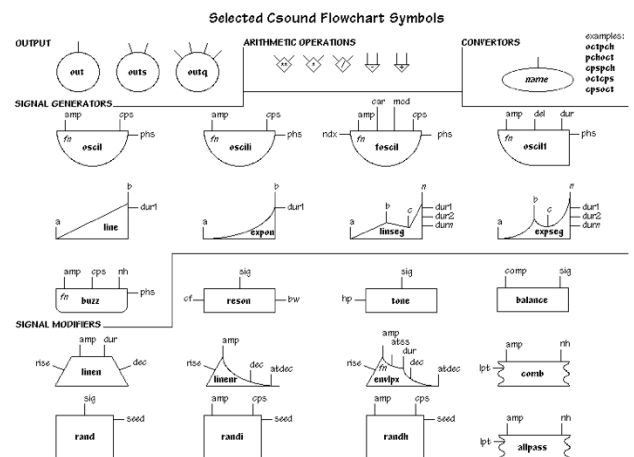
Categories of Synthesis Techniques

- * Representations and models
 - * Additive synthesis and Fourier analysis
 - * Extended additive: vectors and samples
 - * Subtractive synthesis, source/filter models
 - * Non-linear: FM and wave-shaping
 - * Time-domain methods: grains and samples
- * Families of physical models
- * Hybrid/mixed-mode models (SMS)

— MEDIA ARTS & TECHNOLOGY PROGRAM

29

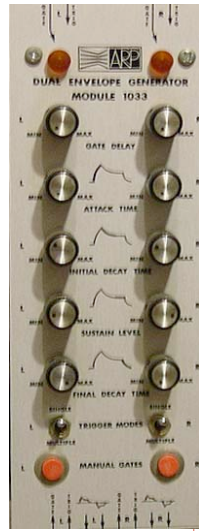
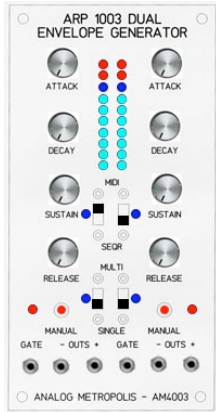
Csound Unit Generators



— MEDIA ARTS & TECHNOLOGY PROGRAM

30

Topic: Programming



31

The PortAudio Framework

- * PA API and “pull” model
- * Call-back functions and streams

```
#include "portaudio.h"
// call-back function signature
typedef int (PortAudioCallback)(
    void *inputBuffer, void *outputBuffer,
    unsigned long framesPerBuffer,
    PaTimestamp outTime, void *userData );

// initialization example
err = Pa_Initialize();
if( err != paNoError )
    printf("PA error: %s\n", Pa_GetErrorText(err));
```

MEDIA ARTS & TECHNOLOGY PROGRAM

32

PortAudio Callback: Sawtooth

```
static int paSawtoothCallback( void *inputBuffer, void *outputBuffer,
    unsigned long framesPerBuffer, PaTimestamp outTime,
    void *userData ) {
    paTestData *data = (paTestData*) userData;    // cast data ptr
    float *out = (float*) outputBuffer;           // cast output buf

    // sample computation loop
    for ( unsigned i = 0; i < framesPerBuffer; i++ ) {
        *out++ = data->phase;    // output = phase value
        data->phase += phase_incr; // increment phase
                                // wrap around
        if ( data->phase >= 1.0f ) data->phase -= 2.0f;
    }
    return 0;    // return no-error value
}
```

MEDIA ARTS & TECHNOLOGY PROGRAM

33

PortAudio Streams

```
err = Pa_OpenDefaultStream(
    &stream,    /* passes back stream pointer */
    0,          /* no input channels */
    2,          /* stereo output */
    paFloat32,  /* 32 bit floating point output */
    44100,      /* sample rate */
    256,        /* frames per buffer */
    0,          /* # buffers, 0 => use minimum */
    paSawtoothCallback, /* specify our custom callback */
    &data );    /* pass our data to callback */

err = Pa_StartStream( stream );
```

MEDIA ARTS & TECHNOLOGY PROGRAM

34

Extensions

- * Global control of freq/ampl
 - * Phase increment and max/min as params
 - * Set/updated from another thread
- * Envelope and triggers
 - * Amplitude decays over time
 - * Triggered from another thread
- * PA callback calls mixer obj: vector of inputs
- * This is enough to write a simple score reader and sequencer

MEDIA ARTS & TECHNOLOGY PROGRAM

35

Data Structures

- * Voice/player
 - * Sound properties
 - * Envelope
- * Mixer, inputs, callback loop
- * Note
 - * Pitch, duration, amplitude
- * Score
 - * List of notes
 - * Schedule

MEDIA ARTS & TECHNOLOGY PROGRAM

36

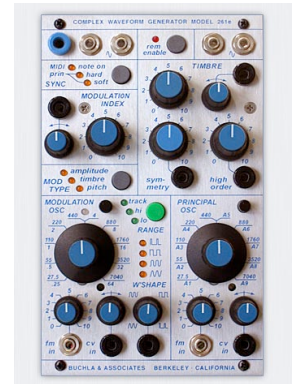
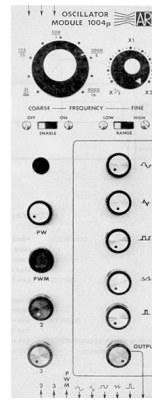
Exercises

- * Install development tools (i.e., > 1) on your platform
- * Work on PortAudio sawtooth, tests
- * Extend as described above
- * Write the same in Java, C++, SC, python, Lua, Smalltalk
- * Write the same using other sound IO APIs (see example callback signatures)

MEDIA ARTS & TECHNOLOGY PROGRAM

37

Topic: Oscillators



MEDIA ARTS & TECHNOLOGY PROGRAM

38

Wavetables and Additive Synthesis

- * Simple fixed-waveform oscillators
 - * Analog paradigm
- * Additive synthesis
 - * True sum-of-sines
 - * IFFT-based synthesis
- * Computed functions vs. wave tables
 - * Speed/space trade-off and distortion
 - * Alternatives (large tables, ringing filters, etc.)

MEDIA ARTS & TECHNOLOGY PROGRAM

39

The Simplest Sine Oscillator

- * Computed Sine Osc:
- * Given a buffer to fill
- * Given F_s , F_w , amplitude
- * In the PA callback, do
 - $\text{phase_incr} = f_w / f_s; \quad // * 2\pi$
 - $*\text{out}++ = \sin(\text{phase}) * \text{amplitude};$
 - $\text{phase} += \text{phase_incr};$
- * What are the design issues here?

MEDIA ARTS & TECHNOLOGY PROGRAM

40

Table Look-up Oscillator

- * In the old world
 - * Computed $\sin()$ was very slow on old machines
 - * Caches were moot (non-existent or slow)
 - * So, store an array in memory with one cycle of a sine wave
- * In the callback function
 - * $*\text{out}++ = \text{table}[(\text{int})(\text{phase} + 0.5f)];$
 - * $\text{phase} += \text{phaseInc};$
 - * $\text{if}(\text{phase} > (\text{tableSize} - 1))$
 - * $\text{phase} -= \text{tableSize};$

0	0.0000
1	0.0123
2	0.0245
...	...
127	0.9999
128	1.0000
129	0.9999
...	...
319	-0.9999
320	-1.0000
321	-0.9999
...	...
509	-0.0368
510	-0.0245
511	-0.0123



MEDIA ARTS & TECHNOLOGY PROGRAM

41

Table Look-up Osc Details

- * Table creation
 - * Case of sine: simple loop
 - * Other cases (SoS, sample playback, etc.)
- * Size vs. cache performance: limit on table size
 - * 1/4-waveform tables & indexing
- * Table indexing
 - * Index math: phase_incr as float; index as int
 - * Size vs. distortion trade-off (index quant. noise)
- * Interpolation in tables - linear, cubic, all-pass
- * Examples (see CSL)

MEDIA ARTS & TECHNOLOGY PROGRAM

42

Sine Osc Performance

- * **Benchmark parameters**
 - * **Computed vs. table look-up**
 - * Easy to measure
 - * **Flavors of interpolation**
 - * Linear, cubic, all-pass
 - * May be more expensive than computed sin (for simple tables)
 - * **Tables larger than the cache**
 - * **Table optimization methods**
 - * **Applicability to non-sine waveforms**

MEDIA ARTS & TECHNOLOGY PROGRAM

43

The CREATE Signal Library

- * **The CSL Framework**
 - * **Buffer** -- holds sample**
 - * **UnitGenerator** -- does nextBuffer()
 - * **Controllable, Scalable, Effect, Phased**
 - * Special handling of control ports
- * **CSL_Types.h** -- core typedefs
- * **CSL_Exceptions.h** -- DASP exceptions
- * **CSL_Core.h** -- Central classes
- * **CGestalt.h** -- globals

MEDIA ARTS & TECHNOLOGY PROGRAM

44

CSL Kernel

- * **Core Classes**
 - * **Buffer** -- the multi-channel sample buffer class (passed around between generators and IO guys)
 - * **Port** -- used to represent signal and control inputs and outputs in named maps; holds a UnitGenerator and its buffer
 - * **UnitGenerator** -- an object that can fill a buffer with samples -- the central abstraction of CSL DSP operations
 - * Mix-in classes (added to UnitGenerator)
 - * **Controllable** -- superclass of the mix-ins that add control or signal inputs (held in maps)
 - * **Effect** -- A (controllable) UnitGenerator subclass that process an input port (e.g., filters, panners). All effects inherit from me.
 - * **Scalable** -- A (controllable) mix-in that adds scale (multiplicative) and offset (additive) inputs (used by most common UGens)
 - * **Phased** -- a (controllable) mix-in for generators with frequency inputs and persistent phase accumulators

MEDIA ARTS & TECHNOLOGY PROGRAM

45

CSL Kernel 2

- * All of these mix-in classes add macros for handling their special named control ports, as in DECLARE_PHASED_CONTROLS, LOAD_PHASED_CONTROLS, and UPDATE_PHASED_CONTROLS
- * **Writeable** -- a mix-in for generators that one can write into
- * **Seekable** -- a mix-in for generators that one can position as a stream
- * **Cacheable** -- for generators that can cache their past output values
- * **Channel/Buffer processing**
- * **FanOut** -- 1-in n-out fan-out object (now built in to UnitGenerator)
- * **Splitter** -- splits a stream into multiple 1-channel outputs
- * **Joiner** -- joins multiple 1-channel inputs into a single stream
- * **Interleaver** -- general interleave/de-interleave
- * **I/O**
- * **IO** -- the input/output stream/driver, its utility functions and virtual constructors, PAIO, CAIO, VSTIO, JackIO, NullIO, RemoteIO

MEDIA ARTS & TECHNOLOGY PROGRAM

46

CSL Oscillator Classes

- * **class Oscillator : public UnitGenerator, public Phased, public Scalable...**
- * **class WavetableOscillator : public Oscillator**

```
WavetableOscillator(Buffer & wave);
WavetableOscillator(Buffer & wave, float frequency, float phase);
void setWaveform(Buffer & wave); // plug in waveforms
void setInterpolate(InterpolationPolicy whether);
// get the next buffer of samples
virtual void nextBuffer(Buffer & outputBuffer, unsigned outBufNum)
    throw (CException);
```
- * **class Sine : public Oscillator**

```
Sine(float frequency, float ampl, float offset, float phase);
void nextBuffer(Buffer & outputBuffer, unsigned outBufNum)
    throw (CException);
```

MEDIA ARTS & TECHNOLOGY PROGRAM

47

CSL Oscillator Details

- * **Abstractions**
 - * **WavetableOsc**
 - * **CompOrCacheOsc**
 - * **SumOfSines**
 - * **SquareBL...**
- * **Versions of Sine**
 - * **Straightforward**
 - * **As a Phased, as a Scalable**
- * **Computed vs. look-up sine**
- * **Details and alternatives (lots of each)**

MEDIA ARTS & TECHNOLOGY PROGRAM

48

Advanced Sine Oscillators

- * Filtered square waves
 - * Common in analog designs
- * Series expansion (11 steps > 16-bit resolution)
- * Ringing filter sine waves
 - * Simple filter is easy as 1 multiply and 2 adds
 - * Very low distortion
 - * Performance great, for fixed frequency
 - * Recomputation of coefficients on frequency change is expensive

$$c = 2.0 * \cos(f) - 2.0$$

$$v = y[1] - (c + 1.0) * y[0]$$
- * Code example and references

MEDIA ARTS & TECHNOLOGY PROGRAM

49

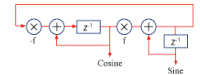
Filter-as-Sine Code

- * Simplified state-variable filter
- * Code


```
// initialize oscillator
sinZ = 0.0;
cosZ = amp;

// Compute "f" on freq change
f = 2 * pi * Fc / Fs

// iterate oscillator
sinZ = sinZ + f * cosZ;
cosZ = cosZ - f * sinZ;
```



Csound Code

```
while (nn--){
  *ar++ = x;
  v += c * x;
  x += v;
}
```

MEDIA ARTS & TECHNOLOGY PROGRAM

50

Sum-Of-Sines Oscillator

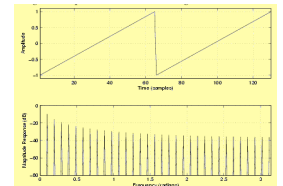
- * Osc banks and additive synth
- * Data structure for spectral slice (fr/am/ph)
- * Loop through overtones summing sines
 - * S...l...o...w...
- * Alternatives
 - * Compute wavetable
 - * Not possible for inharmonic spectra
 - * Use IFFT
 - * Need to maintain phase for non-bin-center frequencies (the normal case)

MEDIA ARTS & TECHNOLOGY PROGRAM

51

Band-limited Waveforms

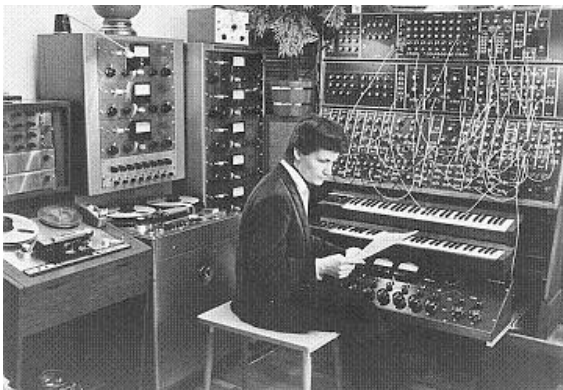
- * Problem: our example sawtooth produces obvious aliasing at the reset point (see Figure below)
- * Question: how to best generate band-limited approximations to canonical non-BL waveforms (saw, square, impulse, etc.)
- * On-the-fly summation
 - * May be quite slow
- * Cached wavetable
 - * Limited freq range
- * Filter a non-BL version
 - * Filter may be expensive
- * http://www.softsynth.com/jsyn/examples/tj_seosc.html



MEDIA ARTS & TECHNOLOGY PROGRAM

52

Topic: Vector Synthesis



MEDIA ARTS & TECHNOLOGY PROGRAM

53

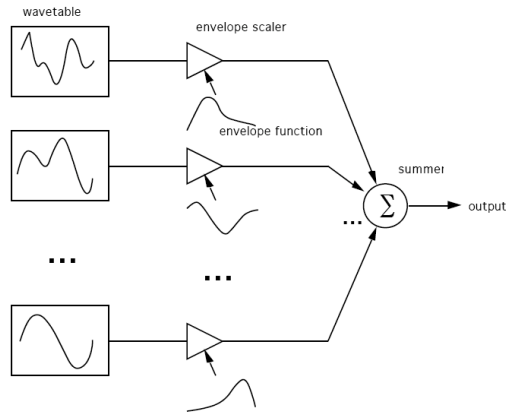
Vector Synthesis

- * Cross-fade between several wave tables
 - * (attack, steady state, decay) (formants)
 - (even/odd harmonics) (harmonic/inharmonic overtones) (deterministic/stochastic components)
- * Real-time control of timbre (envelopes)
- * Timbral vibrato
- * Implementation depends on representation of basic waveforms

MEDIA ARTS & TECHNOLOGY PROGRAM

54

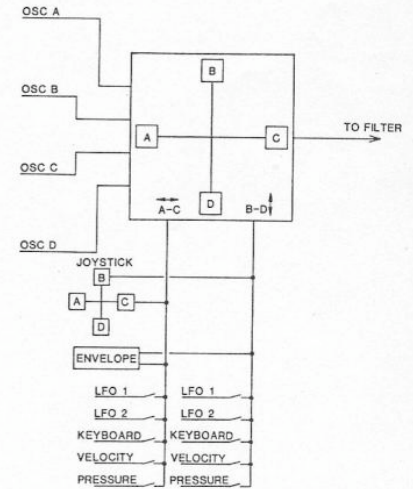
Vector Synthesis Example



MEDIA ARTS & TECHNOLOGY PROGRAM

55

Vector Synthesis Control



MEDIA ARTS & TECHNOLOGY PROGRAM

56

The SHARC Database

- * Spectral averages of standard orchestral instruments
- * Many subtleties (i.e., transients) lost
- * Simple text files for spectrum, instrument, and library
- * Useful for vector synthesis
- * See CSL code examples
 - * SOS or IFFT

MEDIA ARTS & TECHNOLOGY PROGRAM

57

FFT/IFFT Synthesis

- * Synthesis techniques with associated analysis techniques
- * Properties of DFT
- * Performance of FFT
- * Analysis details: MQ analysis, Loris model, stoch+det decomposition (SMS)
- * Manipulating FFT data
- * IFFT and OLA for synthesis
- * Applications (vocoders, 240B code)

MEDIA ARTS & TECHNOLOGY PROGRAM

58

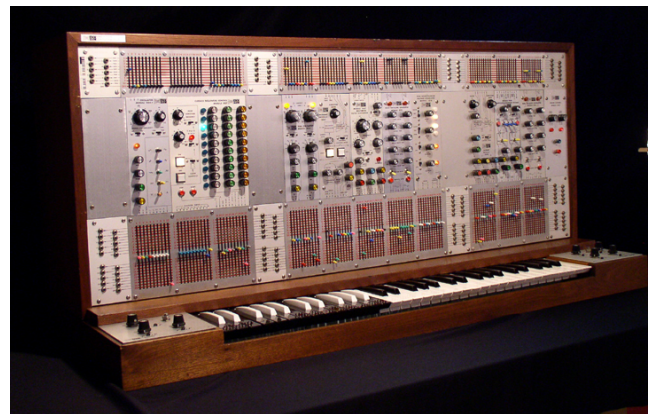
Coding Exercises

- * Basic Oscillators
 - * Computed, cached
 - * Band-limited
- * SumOfSines
- * Vector
- * IFFT
- * Frameworks
 - * CSL, Csound, SuperCollider, JSyn

MEDIA ARTS & TECHNOLOGY PROGRAM

59

Topic: DASP Frameworks



MEDIA ARTS & TECHNOLOGY PROGRAM

60

Introduction to DASP Frameworks

- * **Concepts of OO Modeling**
 - * Inheritance & generalization hierarchies
 - * Abstract and concrete classes
 - * Goal: most code lives high in the trees (and is widely reused)
 - * Leaves are defined by their differences
 - * Core concepts are high-level classes
 - * Models are arrived at through analysis of concrete scenarios, iteration

MEDIA ARTS & TECHNOLOGY PROGRAM

61

Models for DASP Classes

- * **Sound data models**
 - * Sample buffer, stream, signal, none/weak
- * **Source/filter/sink classes**
 - * Unit generator (many options), stream, processor/filter, processing node
 - * Are graphs classes?
- * **Other models**
 - * Ports, plugs, flows
 - * Fcns. of time, control, envelopes

MEDIA ARTS & TECHNOLOGY PROGRAM

62

DASP Modeling Options

- * Who is active/passive?
- * How are connections represented?
- * How about plugs or dynamic variables?
- * Are control signals special?
- * Do control signals exist at all?
- * Is there a control rate?
- * Handling of multi-channel signals
- * How is IO handled?
- * How are graphs activated?

MEDIA ARTS & TECHNOLOGY PROGRAM

63

Selected Model Examples

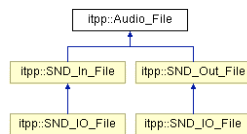
- * IT++
- * Jsyn
- * MXV
- * CSL
- * CLAM
- * Siren
- * SuperCollider
- * Max/MSP, PD
- * Others (analyze your own)...

MEDIA ARTS & TECHNOLOGY PROGRAM

64

IT++ System

- * C++ framework for DASP aimed at telecomm; developed at Chalmers U in Sweden
- * Models of vectors and matrices as C++ template classes
- * Weak (flat) hierarchy of signals and sources/filters
- * Basic file I/O
- * DASP algorithms for coding, modulation, statistics, etc.

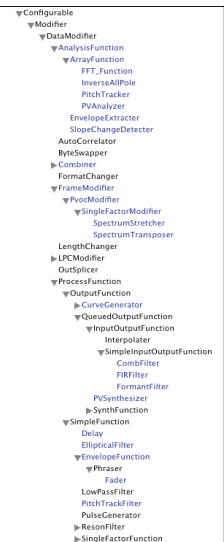


MEDIA ARTS & TECHNOLOGY PROGRAM

65

MXV Design

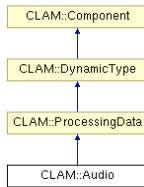
- * Rich OO models of audio data and operations for processing, analysis/resynthesis, editing
- * Basic data objects: array, range, scale, data file, processor, IO, data editor, etc.
- * Data is passive (double*) buffers
- * Good code for FFT, LPC, p-tracking...
- * GUI uses Interviews
- * Lacks good tool support (makefiles)
- * Very good DASP tutorial



66

CLAM Classes

- * Sophisticated design with hierarchies of data components, processes, applications, exceptions
- * Data: sampled, spectral, and other
- * Processing class with state machine and virtual do() method; described by config. classes
- * MVC support classes for DASP
- * Applications for SMS, patching, etc.

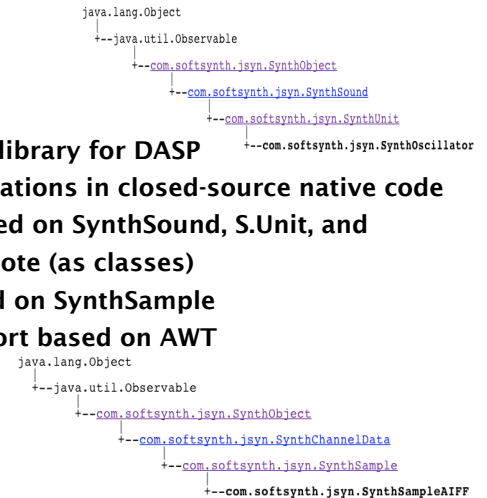


MEDIA ARTS & TECHNOLOGY PROGRAM

67

JSyn

- * Java class library for DASP
- * Implementations in closed-source native code
- * Model based on SynthSound, S.Unit, and S.Circuit/Note (as classes)
- * Data based on SynthSample
- * MVC support based on AWT



MEDIA ARTS & TECHNOLOGY PROGRAM

68

CSL Design

- * Designed for simplicity, flexibility, mainly for synthesis, plug-ins, servers, procedural flavor
- * Multi-channel buffer class (few subclasses)
- * UnitGenerator as main src/filter abstraction
 - * Behavior is nextBuffer callback
 - * CB fcn. gets out buffer
- * Mix-in classes: processor (input UG), phased (accum), writeable, cacheable, etc.
- * Models of static/dynamic variables (plugs/ports), envelopes, IO

MEDIA ARTS & TECHNOLOGY PROGRAM

69

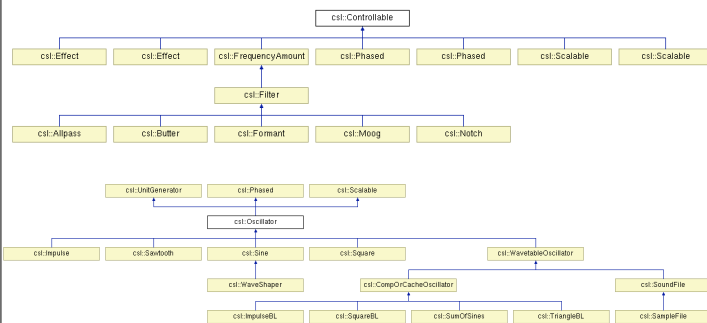
CSL Support

- * UGen models
 - * All standard UGens
- * DASP support
 - * Envelopes in several flavors
 - * Operators on buffers
 - * Variables (static/dynamic)
 - * RingBuffers (circular)
 - * BlockResizer (subgraphs at different blk size)
 - * ThreadedFrameStream (in different thread)
 - * RemoteStream (on different machine)

MEDIA ARTS & TECHNOLOGY PROGRAM

70

CSL 4.1 Models



MEDIA ARTS & TECHNOLOGY PROGRAM

71

Framework Implementations

- * Packaging
 - * C++/Java namespaces
 - * How to extend, embed, distribute
- * Platform/compiler independence (and readability)
 - * Typedefs
 - * Style: naming, formatting

MEDIA ARTS & TECHNOLOGY PROGRAM

72

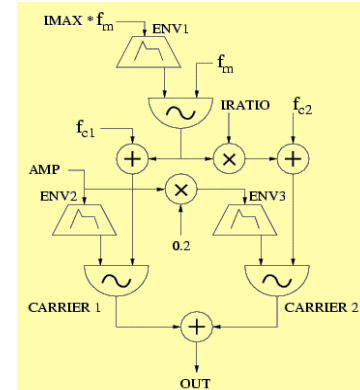
CSL Examples

- * Core classes and class hierarchy
 - * UnitGenerators
 - * Envelopes
- * Building DSP graphs
- * Using IO objects
 - * PAIO, CAIO, JackIO, VSTIO
- * Example main() functions and demos
- * Control and triggering: MIDI and OSC
- * Analyzing the CSL Oscillators
- * Extending the UnitGenerator hierarchy

MEDIA ARTS & TECHNOLOGY PROGRAM

73

Topic: Non-linear Methods



MEDIA ARTS & TECHNOLOGY PROGRAM

74

Wave-shaping Synthesis

- * Signal transfer functions and non-linear processing
- * Wave-shaping insight and Chebyshev polynomials

$$f(ax) = d_0 + d_1 ax + d_2 a^2 x^2 + \dots + d_N a^N x^N$$
- * The envelope problem
- * Wave-shaping implementation options
- * Other non-linear processing

MEDIA ARTS & TECHNOLOGY PROGRAM

75

Wave-shaping Example

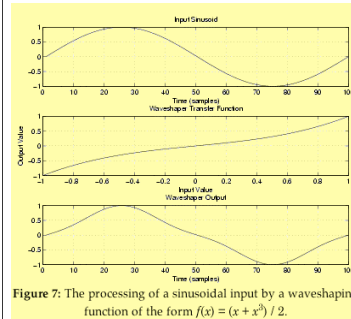


Figure 7: The processing of a sinusoidal input by a waveshaping function of the form $f(x) = (x + x^3) / 2$.

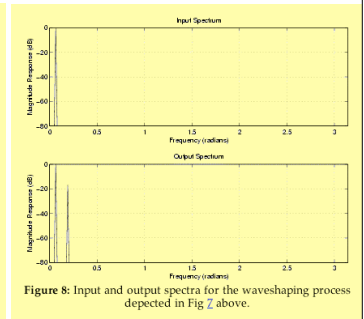
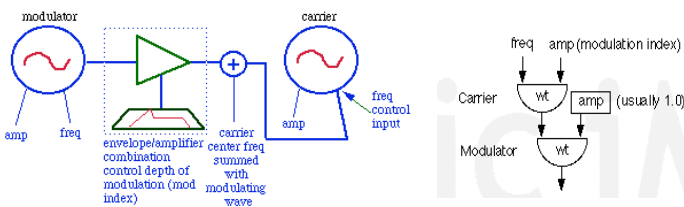


Figure 8: Input and output spectra for the waveshaping process depicted in Fig 7 above.

MEDIA ARTS & TECHNOLOGY PROGRAM

76

Frequency Modulation

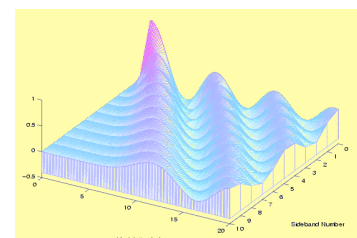
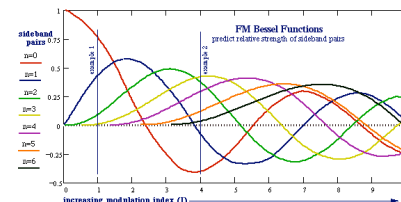


- * Adding vibrato to an oscillator
- * Adding envelopes to the vibrato
- * Increasing the vibrato rate into the audio range (e.g., equal to Fmod)

MEDIA ARTS & TECHNOLOGY PROGRAM

77

Bessel Functions

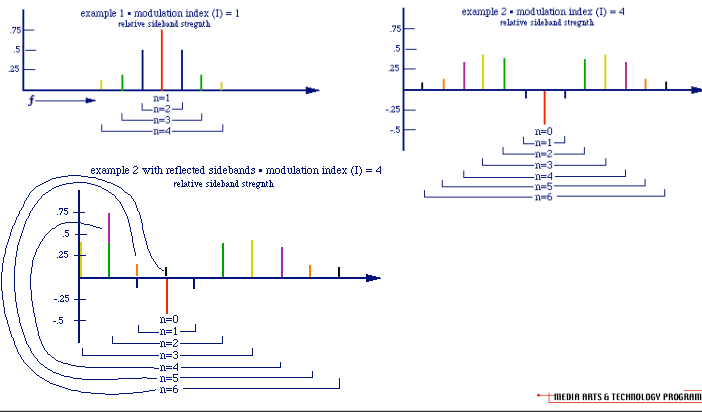


MEDIA ARTS & TECHNOLOGY PROGRAM

78

FM Spectral Sidebands

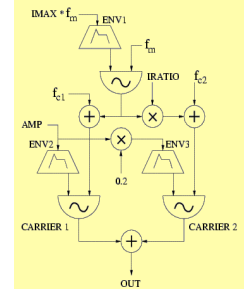
* Aliasing at 0 Hz & $F_s/2$



79

FM Details

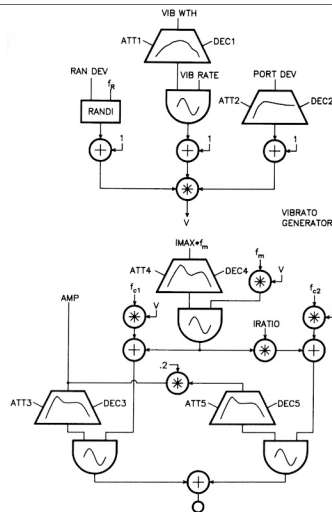
- * Separate amplitude and mod. index envelopes
- * C:M frequency ratio and spectrum
- * Multi-carrier FM
- * M-modulator FM
- * Self-FM
- * See Yamaha patches
- * Canonical FM models and sounds



80

FM Scaled Up

- * Trumpet model by Dexter Morrill



81

FM Parameter Setting

- * C:M ratio
 - * Spacing of harmonics
- * Mod index (envelope)
 - * Spectral richness/brightness
- * Simplest models
 - * "Brass-like" -- small ind. att, low ind.sust.
 - * "Drum-like" -- $\sqrt{2}$ c:m, perc. envs
 - * "String-like" -- env sust. = 1; slow ind. att.

MEDIA ARTS & TECHNOLOGY PROGRAM

82

Advanced FM

- * FM + vibrato
- * Kinds of envelopes
- * Formants with FM
- * Complex FM
- * D. Barstow instruments
- * J. Chowning voices

MEDIA ARTS & TECHNOLOGY PROGRAM

83

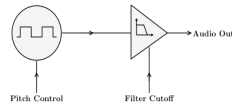
Topic: Subtractive Synthesis



MEDIA ARTS & TECHNOLOGY PROGRAM

84

Subtractive Synthesis

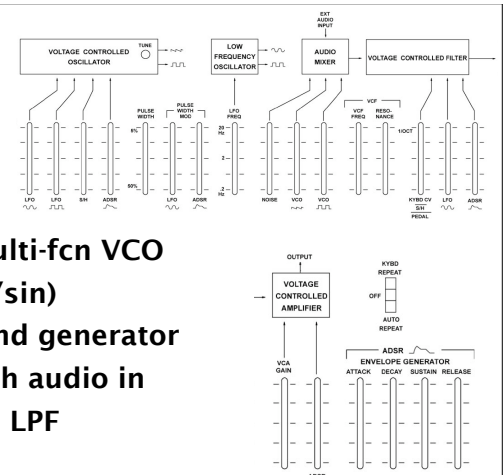


- * **Analog source-filter model**
 - * **Rich source: sum of saw/square/etc**
 - * Oscillators: see above
 - * Other srcs: noise, samples, chaos, etc.
 - * **Dynamic filter: resonant LPF + envelope**
 - * Filter issues
 - * **Other processing**
 - * RingMod, phaser/flanger, reverb
- * **Example: MiniMoog (monsta) above**

MEDIA ARTS & TECHNOLOGY PROGRAM

85

ARP AXXE



- * **Single multi-fcn VCO**
- * **LFO (sqr/sin)**
- * **Noise/rand generator**
- * **Mixer with audio in**
- * **Resonant LPF**
- * **VCA**
- * **Single ADSR**

MEDIA ARTS & TECHNOLOGY PROGRAM

86

Digital Subtractive Synthesis

- * **Source issues**
 - * Band-limited canonical waveforms
 - * Band-limited noise
- * **Filter issues**
 - * **Dynamic filters (computation of coefficients per-sample = expensive)**
 - * **Digital filters**
 - * Canonical form, state variable, FIR, IIR
 - * “Analog sound” filters

MEDIA ARTS & TECHNOLOGY PROGRAM

87

Dynamic Filters that Cheat

- * **Possible solution to the coefficient recomputation problem**
 - * Compute 2 filters per call-back (or at the control rate) and cross-fade
 - * Each callback function determines start and end filter coefficients
 - * In the sample loop: compute both values and cross-fade between them
 - * Some potential zippering (relevant?)

MEDIA ARTS & TECHNOLOGY PROGRAM

88

Synthesizing Noise

- * **Stream of random numbers**
 - * Generally not band-limited
 - * May be filtered to get BL or arbitrary spectral slope
 - * May use random walk, MLS, etc.
 - * Hard to estimate RMS value
- * **Noise colors**
 - * White: equal power per frequency
 - * Pink: equal power per octave (1/f)
- * **See CSL code and references**

MEDIA ARTS & TECHNOLOGY PROGRAM

89

Using Subtractive Synthesis

- * **Source: see above techniques**
- * **Filters**
 - * Simple vs. FIR
 - * See MAT 240B
- * **Control parameters**
 - * Source features, source mix
 - * Filter Fc, bw, feedback
- * **Filter implementation**
 - * Dynamic coefficients vs cheating

MEDIA ARTS & TECHNOLOGY PROGRAM

90

Review

- * Intro, DASP
- * Analog synthesis
- * Digital synthesis background
- * Basic Synthesis Techniques
 - * Additive
 - * Subtractive
 - * Non-linear
 - * Wave-shaping
 - * FM

MEDIA ARTS & TECHNOLOGY PROGRAM

91

Coding Examples

- * Using DASP Frameworks
 - * MXV
 - * CSL
 - * SuperCollider
 - * JSyn
- * Non-linear synthesis
 - * Wave-shaping
 - * FM in all its variants
- * Subtractive synthesis
 - * Sources & filters

MEDIA ARTS & TECHNOLOGY PROGRAM

92

Topic: Samplers



MEDIA ARTS & TECHNOLOGY PROGRAM

93

Sample-based Synthesis

- * Sample playback
 - * Hardware
 - * Software
- * Sample representation
- * Sample banks
- * Tools for sample preparation and transfer
- * Applications

MEDIA ARTS & TECHNOLOGY PROGRAM

94

Sample Playback

- * Simple: store sample in a wavetable
- * May want to change length and pitch independent of one another
- * Need loop points
 - * What part of sound to extend
 - * Not always simple to locate
- * Need transposition function
 - * Interpolating look-up, IFFT
- * Many options (see sampler literature)

MEDIA ARTS & TECHNOLOGY PROGRAM

95

Sampler Hardware

- * History: RAM-based samplers
- * Limited RAM
- * 12-bit (or so) samples
- * MIDI control
- * Sample transfer via MIDI or diskette
- * Proprietary formats



MEDIA ARTS & TECHNOLOGY PROGRAM

96

Soft-Samplers

- * Ubiquitous, built in to DAWs and stand-alone (Kontakt, GigaStudio)



97

E-Mu Soft-Sampler



98

Sampling in Code

- * Store short samples: single notes
 - * Identify steady-state section
 - * Loop it to extend note
- * Transposition/harmonization
 - * Read back sample faster/slower (like a TLU oscillator)
 - * Limits on transposition range
- * Sample/bank storage: many options

MEDIA ARTS & TECHNOLOGY PROGRAM

99

Soft-Sampling

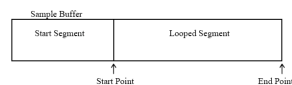
- * Load sample banks
- * Map in-coming commands to:
 - * Bank selection (MIDI channel)
 - * Sample selection (MIDI key)
 - * Sample processing (MIDI key & velocity)
- * Schedule multiple concurrent samples
 - * Loop as necessary (MIDI noteOn/Off)
- * Storage and DSP issues abound

MEDIA ARTS & TECHNOLOGY PROGRAM

100

Sample Representation

- * Single sample: use sound file format with:
 - * Loop points
 - * Base frequency, MIDI key #
 - * Useable frequency range
 - * Amplitude range, map
- * See MOD format
- * See SoundFont format



MEDIA ARTS & TECHNOLOGY PROGRAM

101

Sample Bank Representation

- * Sample bank as list/table of samples
 - * Different pitches
 - * Different amplitudes
 - * Different performance techniques
 - * Can get quite large (many GB)
- * Commercial sample banks
 - * Include proprietary tools for proprietary bank format

MEDIA ARTS & TECHNOLOGY PROGRAM

102

Bank, Instrument, Region

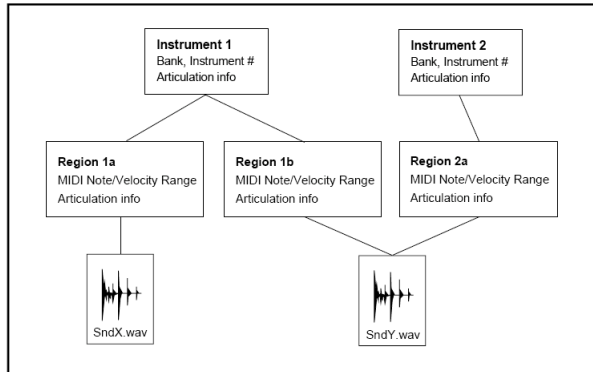


Figure 17. DLS File Structure

MEDIA ARTS & TECHNOLOGY PROGRAM

103

Examples: AKAI S900 Files

* Diskette and file formats

A S900 sample file has a header of 60 bytes as follows:

Length	Format	Description
10	ASCII	Filename
6	0	
4	unsigned	Number of sample words
2	unsigned	Sample rate (Hz)
2	unsigned	Tuning (16ths of a semitone, C3=960)
2	0	
1	ASCII	Loop mode (0=one-shot, L=loop, A=all)
1	0	
4	unsigned	End marker
4	unsigned	Start marker
4	unsigned	Loop length
20		140, 185, 0, 78, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 224, 43, 38, 0, 0, 0

Track 1, side 1 starts with 64 entries of 24 bytes as follows:

Length	Format	Description
10	ASCII	Filename
6	0	
1	ASCII	File type: 'S'=sample, 'P'=program, etc.
3	unsigned	File length in bytes
2	unsigned	Starting block on disk
2		S900 ID = (0,0)

MEDIA ARTS & TECHNOLOGY PROGRAM

104

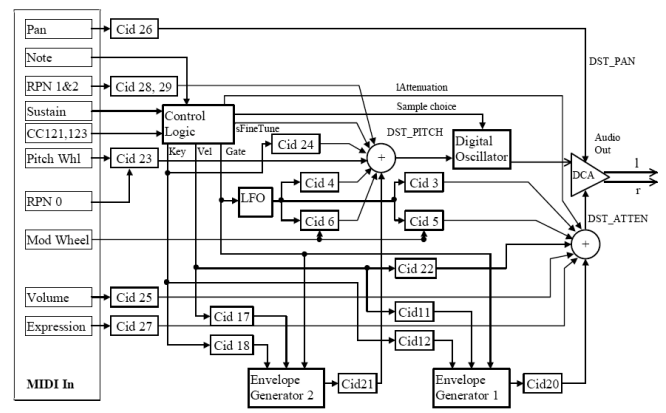
Sample Processing

- * Selection, look-up
 - * Sample bank accessing
- * Playback processing
 - * Transposition, looping
 - * Given loop points, treat loop section as a wavetable
 - * Calculate phase increment and oscillate until note-off
- * Amplitude scaling
 - * Sample may be scaled by a non-linear map to MIDI key velocity

MEDIA ARTS & TECHNOLOGY PROGRAM

105

SoundFont Synth Model



MEDIA ARTS & TECHNOLOGY PROGRAM

106

Using COTS Sample Banks

- * Loading sets of AIFF/WAV files
- * Importing MOD files
- * Loading SoundFont sample banks
- * Other formats
- * Sampler APIs & tools

MEDIA ARTS & TECHNOLOGY PROGRAM

107

Vienna Symphonic Library

▶ 01 Violin ensemble - 14	▶ Folder	717.2 MB
▶ 02 Viola ensemble - 10	▶ Folder	709.2 MB
▶ 03 Cello ensemble - 8	▶ Folder	782.8 MB
▶ 04 Double Bass ensemble - 6	▶ Folder	602.2 MB
▶ 05 Harp	▶ Folder	285 MB
▶ 06 Piccolo Flute	▶ Folder	111.7 MB
▶ 07 Flute	▶ Folder	394.2 MB
▶ 08 Oboe	▶ Folder	139 MB
▶ 09 Clarinet	▶ Folder	339.9 MB
▶ 10 Bassoon	▶ Folder	258.1 MB
▶ 11 Horn solo	▶ Folder	188.6 MB
▶ 12 Horn ensemble - 4	▶ Folder	321.9 MB
▶ 13 Trumpet solo	▶ Folder	144.9 MB
▶ 14 Trumpet ensemble - 3	▶ Folder	130 MB
▶ 15 Trombone solo	▶ Folder	163.5 MB
▶ 16 Trombone ensemble - 3	▶ Folder	229 MB
▶ 17 Bass Tuba	▶ Folder	193.5 MB
▶ 18 French Oboe	▶ Folder	232.2 MB
▶ 19 english Horn	▶ Folder	155.8 MB
▶ 20 Cymbals	▶ Folder	164.7 MB
▶ 21 Timpani	▶ Folder	768 MB
▶ 22 Drums	▶ Folder	56.5 MB
▶ 23 Mallets & Bells	▶ Folder	464.2 MB
▶ 24 Percussion	▶ Folder	94.2 MB

MEDIA ARTS & TECHNOLOGY PROGRAM

108

109



MEDIA ARTS & TECHNOLOGY PROGRAM

MEDIA ARTS & TECHNOLOGY PROGRAM

111

111

112



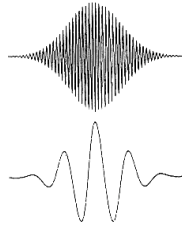
113

495



Granular Synthesis and Time-domain Summation

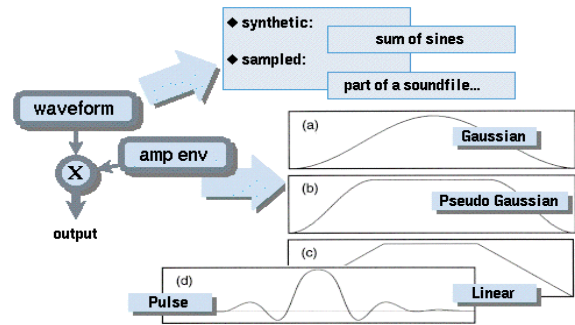
- * Short-time time-domain representation of sound
- * Sound quanta ("grains") last 2-50 msec in general (phonon)
- * Grain clouds are described by their statistical features
- * Granular processing of live sound or sound files
- * Granular pitch/time processing



MEDIA ARTS & TECHNOLOGY PROGRAM

115

Grain Processing



MEDIA ARTS & TECHNOLOGY PROGRAM

116

Grain Synthesis

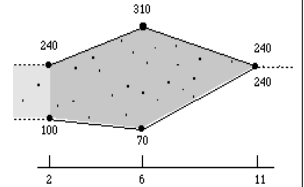
- * Control over single grain parameters
 - * Duration, base freq, spectrum, envelope
- * E.g., SC2 code
 - Create the grain unit generator
 - snd offset rate
 - grain = Acpgrain(sgdrs, tpt, pch, dur, amp, ...);
- * Options
 - * Envelope
 - * Linear, Gaussian, raised cosine, pulse, triangular, sinc
 - * Waveform
 - * Sine, SumOfSines, noise, from-file, VOSIM pulse

MEDIA ARTS & TECHNOLOGY PROGRAM

117

Managing Clouds

- * Tendency masks
- * Control functions
- * Description of clouds:
 - * Density, grain dur, base freq, freq range, ...
- * Csound clouds
 - * ar grain xamp, xpitch, xdens, kampoff, kpitchoff, kgdur
- * CSL Granulator
 - * Granulator(FrameStream & wave, FrameStream & env, FrameStream & rate, FrameStream & amp, FrameStream & freq, FrameStream & dur, FrameStream & pan);



MEDIA ARTS & TECHNOLOGY PROGRAM

118

Flavors of Granular Synthesis

- * Relationship between periods of base or formant freq and inter-grain-delay
- * Fourier/wavelet grids: STFT spectral component as grain
- * Pitch-synchronous: overlap-add; grain rate as fundamental
- * Asynchronous: all parameters as potential probability distribution masks
- * Can be described as base pitch + formants

MEDIA ARTS & TECHNOLOGY PROGRAM

119

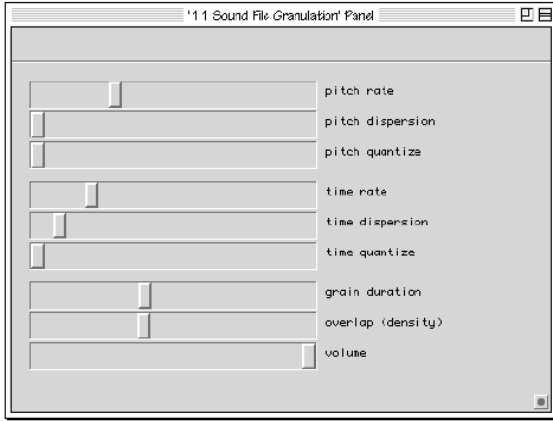
Granular Sound Processing

- * Time granulation for pitch/time processing (stretch, compress, transpose)
- * Grain reordering statistical processing
- * Grain reverb, spatialization
- * GUIs for file granulation

MEDIA ARTS & TECHNOLOGY PROGRAM

120

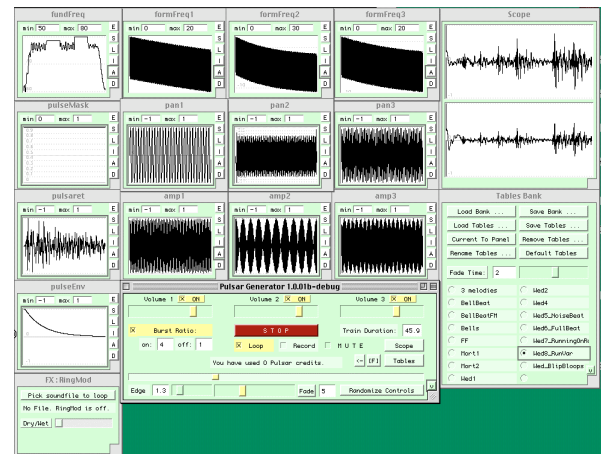
SC2 Granular Processor



MEDIA ARTS & TECHNOLOGY PROGRAM

121

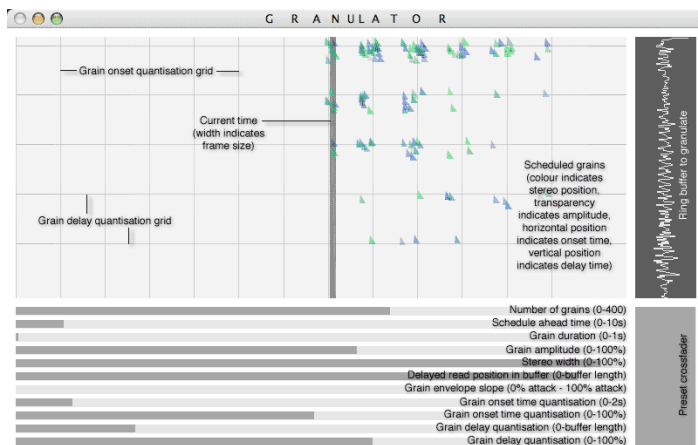
PulsarGenerator



MEDIA ARTS & TECHNOLOGY PROGRAM

122

GrahamW Granulator



MEDIA ARTS & TECHNOLOGY PROGRAM

123

Granular Synthesis Code

- * Doing it the simple way
 - * Grain struct/object
 - * Waveform, dur, envelope, etc.
 - * Function to create a grain
 - * `void write_grain(float *buffer, unsigned len);`
 - * Function to control granular clouds
 - * `void play_grains(...)`
 - * Use fixed parameters (base + rand. range)
 - * Use breakpoint tendency masks

MEDIA ARTS & TECHNOLOGY PROGRAM

124

Efficiency Issues

- * Avoid per-grain function-call (?)
 - * Use array of structs (fPtrs? functors?)
- * Simplify envelope computation
 - * Wavetable for envelope
 - * Linear/triangle envelopes
- * Simplify cloud management
 - * Pre-compute ranges
- * Is this all necessary?

MEDIA ARTS & TECHNOLOGY PROGRAM

125

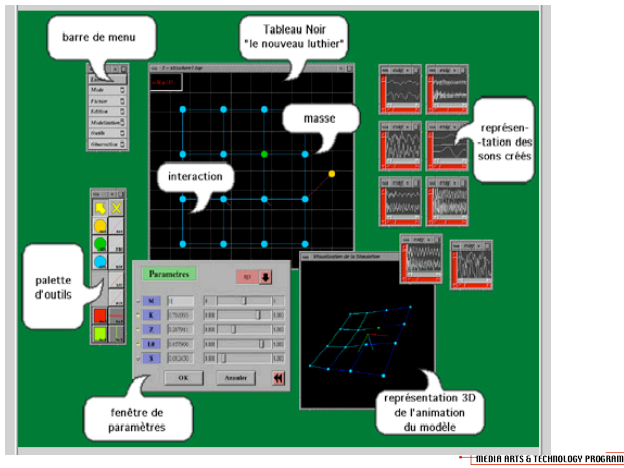
Coding Examples

- * STK C++
- * CSL C++
- * MINT/Synz
- * jMusic Java
- * SuperCollider
- * Others?

MEDIA ARTS & TECHNOLOGY PROGRAM

126

Topic: Physical Modeling



127

Physical Modeling Sound Synthesis

- Not a single technique, rather a family of model types
- Physics of musical instrument acoustics
- "Pure" PM: simulation of a model of instrument acoustics
- "Physical-informed" PM: spectral, time-domain, etc. modeling
- Families of P. models
- Many figures taken from Julius O. Smith, *Physical Audio Signal Processing*, August 2004 and Perry Cook, *Physically-Based Parametric Sound Synthesis and Control*, 2000

128

Musical Instrument Physics

- Vibrating strings with resonators
 - Excitation (bow, strike, sympathetic)
 - Traveling waves and string terminations
 - Resonators and string/resonator transfer functions
- Membranes and plates
 - Excitation, decay, damping, resonance
- Columns of air
 - Excitation, bore characteristics, termination

129

Families of Models

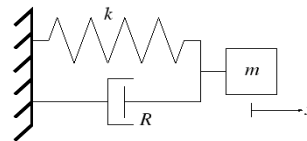


Figure 2: An ideal mass-spring-damper system.

$$\ddot{y} + 2 \sum_{k=0}^M q_k \frac{\partial^{2k+1} y}{\partial x^{2k} \partial t} + \sum_{k=0}^N r_k \frac{\partial^{2k} y}{\partial x^{2k}}$$

- Dynamic Systems
 - Discrete, finite element simulation
 - Continuous differential equations
- Mass/spring/damper
- Traveling waves
- Modal-domain

130

Mass-and-Spring Oscillators

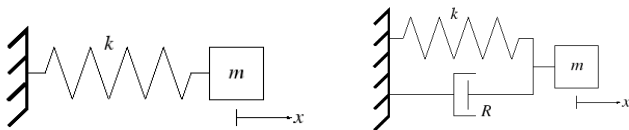


Figure 2: An ideal mass-spring-damper system.

By Newton's Second Law: $F = ma = m \frac{d^2x}{dt^2} = m \frac{d^2x}{dt^2}$

System equation: $m \frac{d^2x}{dt^2} + R \frac{dx}{dt} + kx = 0$

$$x = e^{-\alpha t} A \cos(\omega_d t + \phi).$$

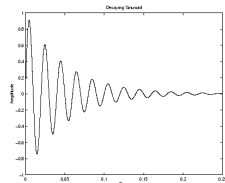
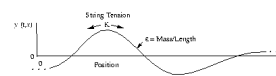


Figure 3: A decaying sinusoid.

- <http://links.math.rpi.edu/applets/appindex/springmass.html>

131

The Physics of Strings



$$y(t, x) = y_r \left(t - \frac{x}{c} \right) + y_l \left(t + \frac{x}{c} \right)$$

$$K y'' = \epsilon \ddot{y}$$

where:

$$\begin{array}{ll} K \triangleq & \text{string tension} \\ \epsilon \triangleq & \text{linear mass density} \\ y \triangleq & \text{string displacement} \end{array} \quad \begin{array}{ll} y \triangleq & y(t, x) \\ \ddot{y} \triangleq & \frac{\partial^2 y}{\partial t^2}(t, x) \\ y' \triangleq & \frac{\partial y}{\partial x}(t, x) \end{array}$$

- Simple terminated, taught, stiff string
- Excitation between the end and the mid-point creates 2 counter-traveling waves
- These reinforce/cancel each other, leading to a constant frequency of resonance (depends on tension, stiffness, length, and weight) as the excitation dissipates

132

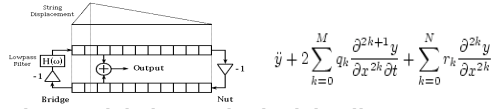
Models of Plucked Strings

* “Pure”

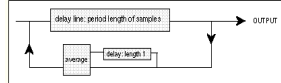
- * Each section modeled as a single delay line
- * Termination reflections are transfer functions
- * May include resonant body and string/resonator transfer function

* “Cheap” (Karplus, Strong, Smith, Jaffe)

- * Inject a noise burst into a recirculating delay line
- * Delay line length gives the “resonant” frequency
- * LPF in the recirculation path determines decay time and damping
- * Sounds great!, many issues



$$\ddot{y} + 2 \sum_{k=0}^M q_k \frac{\partial^{2k+1} y}{\partial x^{2k} \partial t} + \sum_{k=0}^N r_k \frac{\partial^{2k} y}{\partial x^{2k}} = 0$$



133

Coding

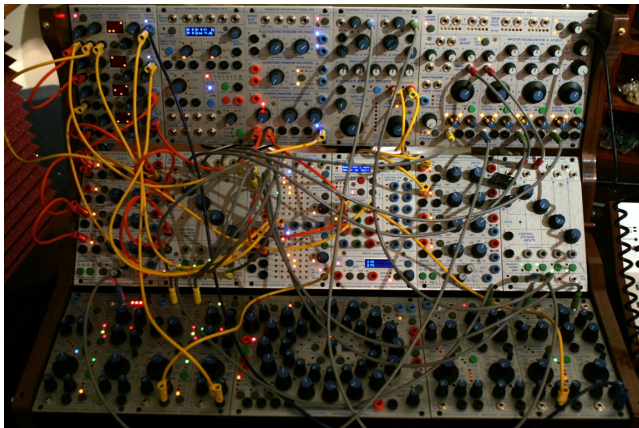
* KS String in CSL

- * VSTi version
- * With controls
- * CSL FDN
- * STK, Chuck
- * FE Membrane models
- * Editor
- * Simulator

MEDIA ARTS & TECHNOLOGY PROGRAM

134

Topic: Waveguides

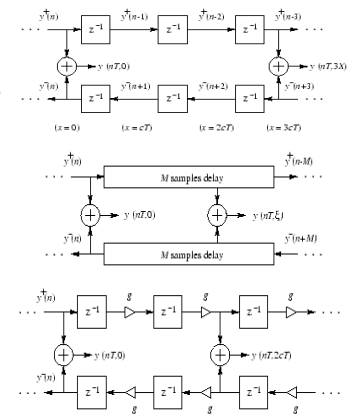


MEDIA ARTS & TECHNOLOGY PROGRAM

135

Digital Waveguides

- * Ideal lossless
- * Commuted delays
- * Ideal lossy



MEDIA ARTS & TECHNOLOGY PROGRAM

136

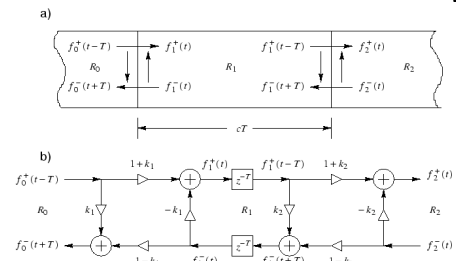
Implementing Waveguides

- * Recirculating delay line cluster as set of ring buffers with tap read/write functions
- * Non-integer delays (all-pass?)
- * Class design for waveguide meshes and feedback delay networks
- * Optimizations for speed and space profiles

MEDIA ARTS & TECHNOLOGY PROGRAM

137

Junctions and Scattering

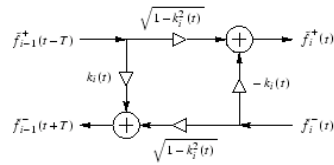


- * Think optics: scattering at any impedance change
- * Guess what: modeled as a (freq-dependent) transfer function

MEDIA ARTS & TECHNOLOGY PROGRAM

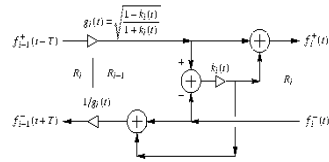
138

Scattering Junction Implementation



Uses 3 multiplies

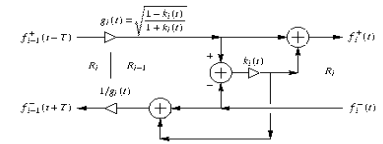
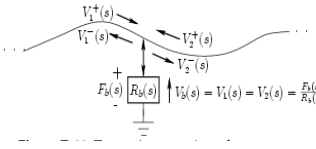
- * Normalized
- * Optimized



MEDIA ARTS & TECHNOLOGY PROGRAM

139

Example: Coupled Strings

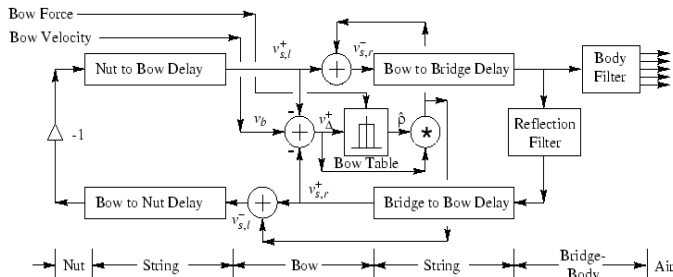


- * Equal-Z strings coupled by a common bridge filter

MEDIA ARTS & TECHNOLOGY PROGRAM

140

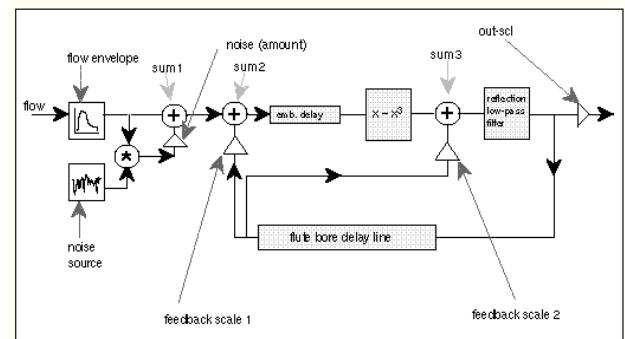
Refined Bowed String Model



MEDIA ARTS & TECHNOLOGY PROGRAM

141

Example: PM Flute 1

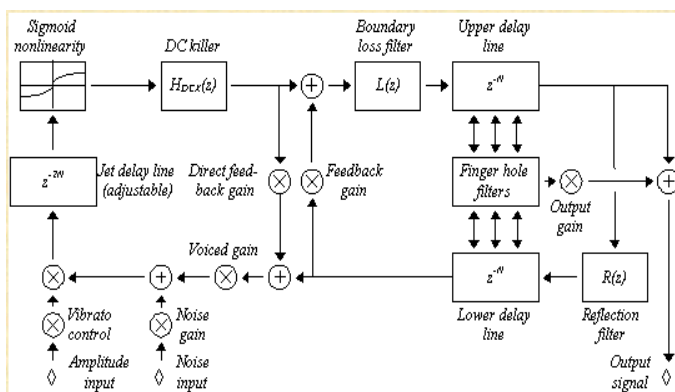


- * P. Cook's flute

MEDIA ARTS & TECHNOLOGY PROGRAM

142

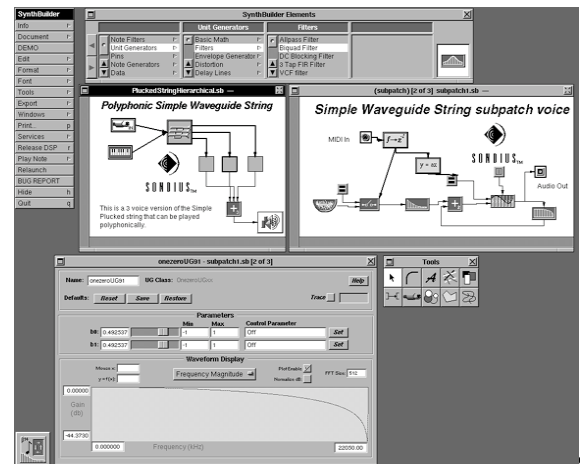
Refined PM Flute



MEDIA ARTS & TECHNOLOGY PROGRAM

143

Example: CCRMA/Staccato SynthBuilder

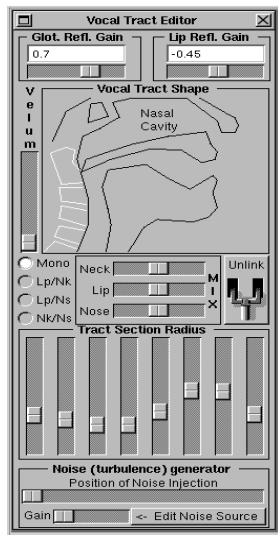


MEDIA ARTS & TECHNOLOGY PROGRAM

144

Example: PM Voice

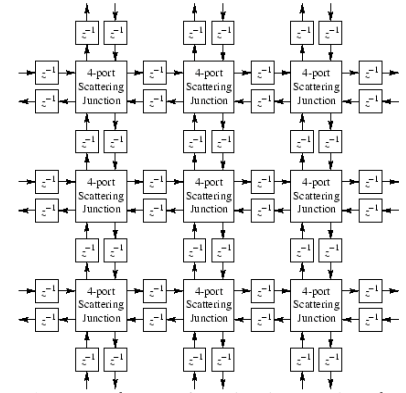
- * Vocal tract seen as a varying-cross-section tube driven by glottal pulses (P. Cook NeXT app.)



145

Waveguide Meshes

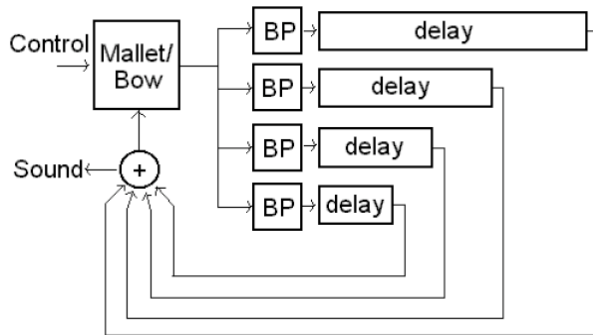
- * PM of surfaces
- * Stiff 2D vibration



MEDIA ARTS & TECHNOLOGY PROGRAM

146

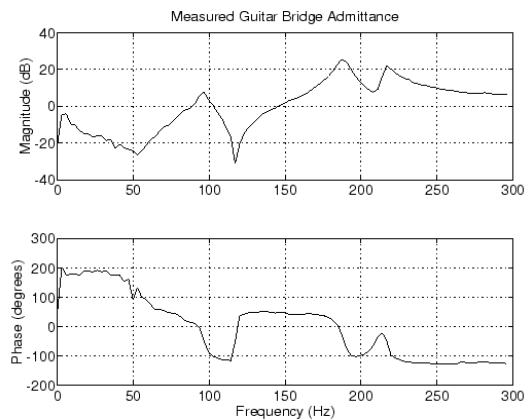
Banded Waveguide Systems



MEDIA ARTS & TECHNOLOGY PROGRAM

147

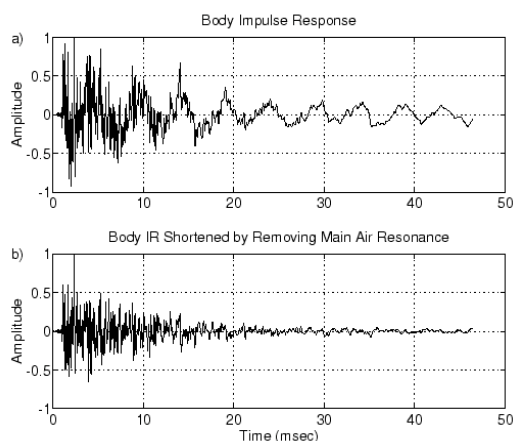
Measurement of Physical Objects



MEDIA ARTS & TECHNOLOGY PROGRAM

148

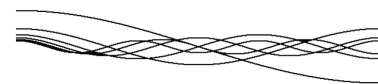
Violin Body Impulse Response



TECHNOLOGY PROGRAM

149

Modeling Modal Vibration



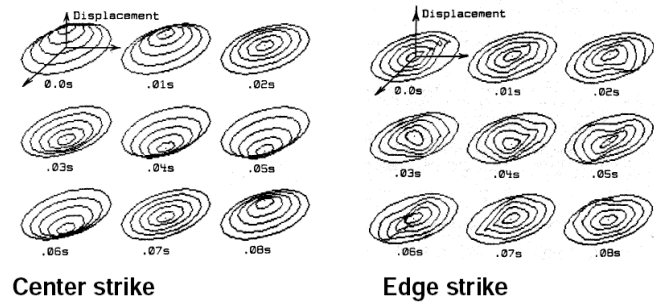
- * Describe the continuous Eigen-modes of oscillation for fixed- and free-coupled materials
- * Excite resonances of a solid body

MEDIA ARTS & TECHNOLOGY PROGRAM

150

Round Membranes

Modes of Plates: inharmonic (round = Bessel)

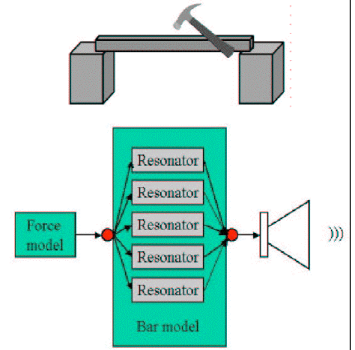


MEDIA ARTS & TECHNOLOGY PROGRAM

151

Implementing Modal Synthesis

- * Excitation/filter
- * Source characteristics
- * Resonator bank
- * Dynamic filters
- * Pick-up, radiation, sensing

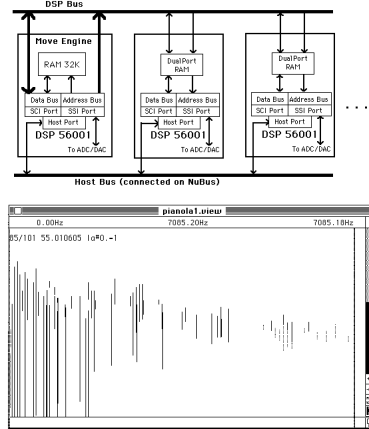


MEDIA ARTS & TECHNOLOGY PROGRAM

152

A. Freed's Reson8

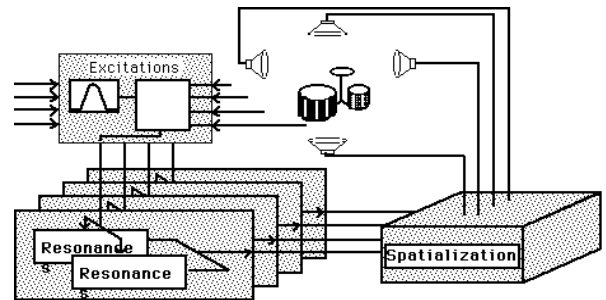
- * Parallel HW implementation of large n-pole resonator banks
- * Stream loads of parameter changes
- * Editors for "ringing pole" schedules



MEDIA ARTS & TECHNOLOGY PROGRAM

153

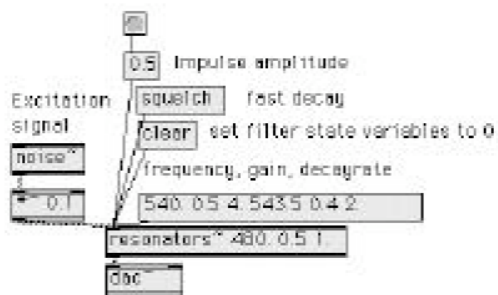
Reson8 in Use



MEDIA ARTS & TECHNOLOGY PROGRAM

154

Resonators~ in Max/MSP



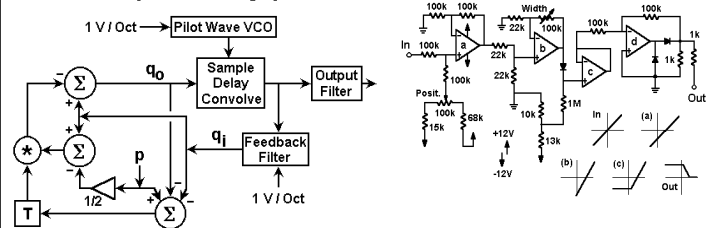
- * Bank of [freq, gain, decayrate] filters

MEDIA ARTS & TECHNOLOGY PROGRAM

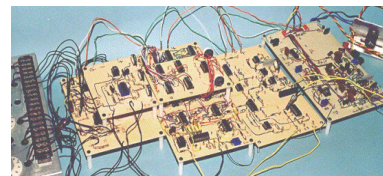
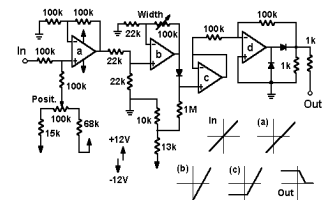
155

Ian Fritz's Analog Modal Clarinet

Clarinet Physical-Modeling Synthesizer



Reed Circuit (T)



MEDIA ARTS & TECHNOLOGY PROGRAM

156

Other PM Paradigms

- * Particle synthesis
 - * “Grain” model
 - * Clustering statistics
 - * Container model
- * PM interpretation of granular synthesis
- * PM interpretation of wavelet synthesis
- * Spectral synthesis interpretation of PM

MEDIA ARTS & TECHNOLOGY PROGRAM

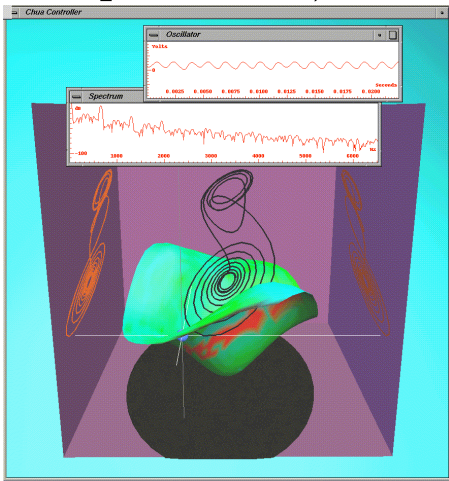
157

Yamaha VL1 PM Synth Module

Keyboard setting

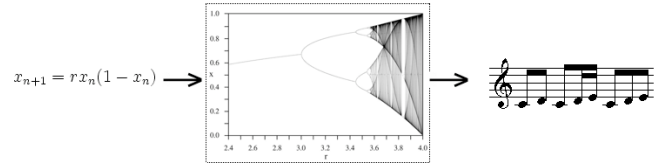
Transposition:

Topic: Chaos, etc.



163

Chaos and Novel Techniques

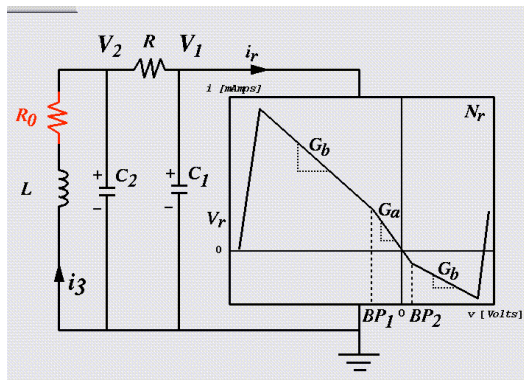


- * **Attractors as signals**
- * **Control and monitoring**
- * **Interaction**
- * **Examples**
 - * Chua's oscillator
 - * Lorenz attractors

MEDIA ARTS & TECHNOLOGY PROGRAM

164

Chua's Circuit



- * **Quasi-multistable analog systems**

MEDIA ARTS & TECHNOLOGY PROGRAM

165

Other Synthesis Methods

- * **Hybrid methods**
 - * Sampled vectors
 - * Physical samples
 - * Spectral morphing
 - * Sample-based subtractive (Morpheus)
- * **Others tried and rejected**
 - * Summation methods (JAM)
 - * Sample-level processing (PILE)

MEDIA ARTS & TECHNOLOGY PROGRAM

166

Coding

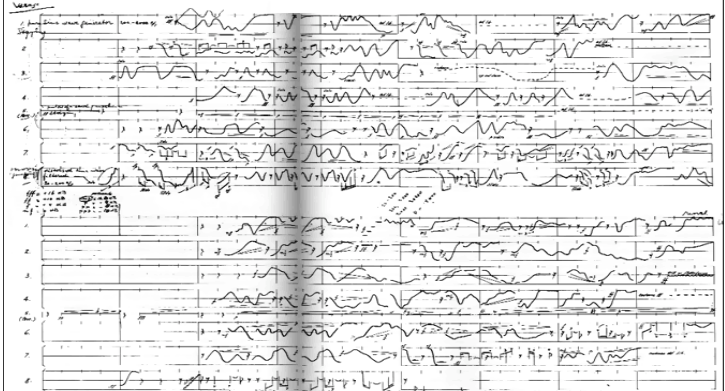
- * **Advanced PM software**
- * **Chaotic score techniques**
 - * SC, Siren
- * **Chaotic oscillators for control**
 - * As Max/PD object
- * **Chaotic sound sources**
 - * Lorenz in CSL

MEDIA ARTS & TECHNOLOGY PROGRAM

167

Topic: System Integration

- * **Control of synthesis techniques**
- * **Players and instruments**



168

Projects

- * Integration & control of synthesis
 - * Soft-synth plug-ins
 - * Scores and control mappers
 - * Player programs and instruments
- * Platforms and DASP environments
 - * Tools in CSL, CLAM, ...
 - * Instruments: VSTi, AUI, JACK
- * Integration with 240[A-C] projects

MEDIA ARTS & TECHNOLOGY PROGRAM

169

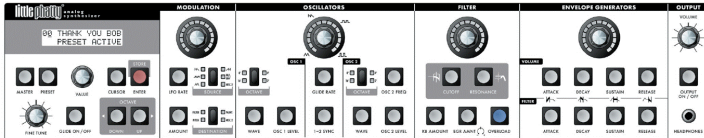
Synthesis & Processing

- * Taxonomy of synthesis techniques
- * Processing
 - * Time-domain
 - * Frequency-domain
 - * Effects
 - * Vintage effects
- * Control, synthesis, processing, spatialization, output

MEDIA ARTS & TECHNOLOGY PROGRAM

170

Review: MAT 240D



- * Wavetables and additive synthesis
- * Vector synthesis
- * DASP frameworks
- * FM and wave-shaping
- * Subtractive and filter-based
- * Sample-based
- * Granular synthesis, time-domain
- * Physical models
- * Chaos and novel techniques
- * Integrated applications

MEDIA ARTS & TECHNOLOGY PROGRAM

171

Where do we go from here?

- * MAT 240E: Control/Networking
 - * Control IO: MIDI, OSB, USB/HID, CUI
 - * Networking: SDIF, RFS
 - * Distributed Systems: CRAM
- * CSL/Synz/MINT groups
- * Integrated systems
- * The CNSI AlloSphere

MEDIA ARTS & TECHNOLOGY PROGRAM

172

Ah, Analog...

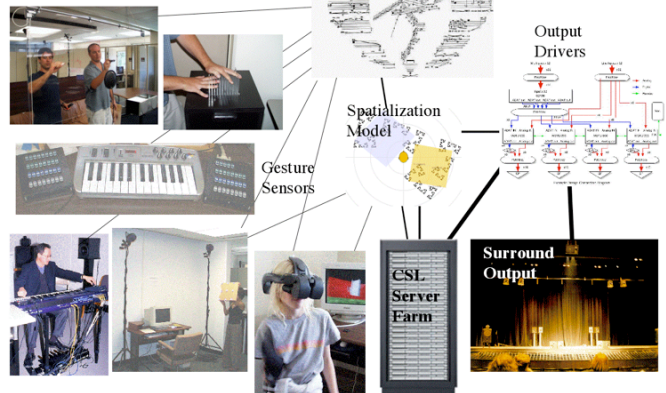


MEDIA ARTS & TECHNOLOGY PROGRAM

173

Thank You

OSC + CSL in Pictures



174

MAT 240E - Digital Audio Programming: Synthesis Control & Streaming (Winter, 2008)

The MAT 240 course sequence is a six-part (two-year) practical programming course; it consists of hands-on software development devoted to digital audio and multimedia applications. Students read a selection of papers from the literature, with the emphasis on learning to use and extend the current state-of-the-art programming methods, tools, and programming interfaces. Class assignments involve C/C++/Java programming on Linux, Macintosh, MS-Windows, various plug-in APIs, and other platforms.

The MAT 240E course focusses on (1) protocols for real-time control of sound synthesis and processing, primarily the Musical Instrument Digital Interface (MIDI), the OpenSoundControl (OSC) protocol, and the USB human interface device (HID), and (2) network streaming of real-time audio using protocols such as UDP and RTP. The plan is to write programs that map in-coming data from haptic controllers, and to use this data to control hardware and software synthesizers and processors. More ambitious projects will address the issues of distributed synthesis and control routing in multi-server systems. Students are expected to know the basics of digital audio signal representation and processing, and to be proficient in C, C++, or Java (Smalltalk, SuperCollider or LISP are a plus). Grading will be on the basis of in-class participation and programming projects.



Course Outline

MIDI

- The MIDI HW and SW spec.
- MIDI drivers and APIs
- Scheduling and MIDI I/O

The Open Sound Control (OSC) protocol

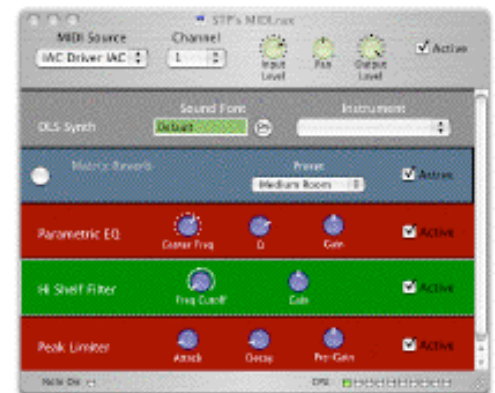
- OSC address space design
- OSC implementation, network programming
- OSC synthesis servers in CSL and Supercollider

The USB human interface device (HID) API

- Gesture sensor interfaces & USB

Audio streaming in distributed applications

- The RFS, RTP and RTCP protocols
- Distributed Processing Environments



Instructor

- Stephen T. Pope (stephen@mat.ucsb.edu)

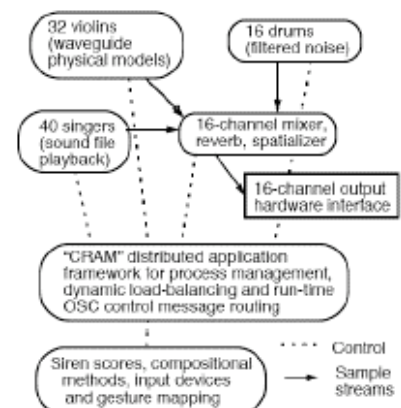
Meeting time and place

- Mon/Wed 1:00 - 2:50 PM, South Hall 4430

Electronic Resources

- Course Web Site: <http://www.create.ucsb.edu/240>
- Email Mailing List

See <http://www.mat.ucsb.edu/mailman/listinfo/240> to join



MAT 240E Reader Contents

MIDI

Musicians Make a Standard: The MIDI Phenomenon: D. G. Loy, CMJ 9(4), 1985

MIDI and Musical Instrument Control: JAES 51.4, 2003

Mac OS 9 OMS MIDI Header file for Squeak MIDI Plug-in

PortMIDI and OCCAM Header Files

Peavey PC-1600X Fader Box MIDI implementation

Code: MIDI Examples for Linux, MacOS & MS-Windows.

PortMITI, RtMIDI, QT_MIDI, JUCE

MIDI Monitor source, OCCAM, MACCO

Steinberg VST(i) SDK

Score Formats

Little Languages for Notelists: STP 2006

Code: EvetLists.h Header File

OSC

OSC: State of the Art 2003: Wright, Freed and Momeni, NIME 2003

OSC Spec, Applications, Examples

Code: LibLo, OSCPack, OSC++, WOscLib

CSL, SC, Max/Pd OSC APIs

Braun OSC Visualizer

USB HID, P5

Programming USB HID: Microsoft Learning

Examining the P5 Glove

P5Glove header file

Arduino Examples, Wii Remote Doc

Code: LibUSB, USBUtils, USB HID Examples

P5 glove examples, JUCE WiiRemote Code

Networking

Streaming Media Wikipedia entry

Audio over CAT5: EDN blog

Code: RTP/RTCP Examples, jRTPLib, LibRTP, RTPTools

CSL RFS Code

Distributed Processing, CRAM

All About CRAM

Code: CRAM Source

Appendix

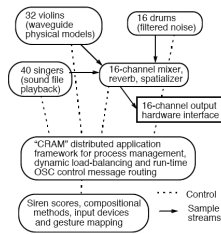
Greatest hits of NIME Proceedings

See Also

CUI, EcoMote documentation

La Kitchen Sensors

MAT 240E Digital Audio Programming: Synthesis Control and Streaming



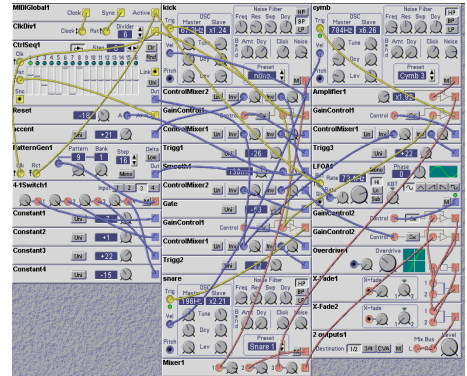
Stephen T. Pope
stp@mat.ucsb.edu
Winter, 2008

MAT 240E

stp@mat.ucsb.edu

1

MAT 240E Overview



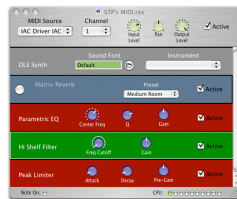
MAT 240E

stp@mat.ucsb.edu

2

Lecture I Outline

- MAT 240 course series: digital audio programming (in C/C++/Java)
- MAT 240E Goals
- Course logistics
- Written materials
- Course overview
- Topic I
 - Review: digital audio programming: signals and control



MAT 240E

stp@mat.ucsb.edu

3

MAT 240 Series

- Hands-on programming courses using (primarily) C, C++, and Java for digital audio application development
- Six-quarter (two-year) course series
- Students use a variety of software tools (IDEs) on MS-Windows, Linux/UNIX, and MacOS

MAT 240E

stp@mat.ucsb.edu

4

MAT 240 Topics (6 quarters)

- A: File I/O and Streaming Media
- B: Spectral Transformations
- C: Spatial Processing of Sound
- D: Sound Synthesis Techniques
- E: Control and Distributed Processing
- F: Analysis and Feature Extraction

MAT 240E

stp@mat.ucsb.edu

5

MAT 240E

- Sensor/control protocols and APIs
 - Musical Instrument Digital Interface (MIDI)
 - Open Sound Control (OSC) protocol
 - USB human interface device (HID) API
- Network streaming of real-time audio
 - UDP and RTP protocols
- Application integration
 - Controllers and sensors
 - Results from MAT 240A-D
 - In MAT 4th-floor lab



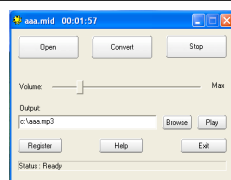
MAT 240E

stp@mat.ucsb.edu

6

Course Logistics

- Lectures
 - Mon/Wed 1:00 - 3:00 PM
 - MAT South Hall Lab (SH 4430)
- Lab, work groups TBD
- Grading
 - Class participation, discussions
 - 2-3 in-class presentations
 - Final presentation and write-up



MAT 240E

stp@mat.ucsb.edu

7

Course Materials

- Presentation slides (handed out)
- Readings: available in CREATE class room, C. Roads *Computer Music Tutorial*, articles from *Computer Music Journal* and *Proc. ICMC*
- MAT 240E web site and links
- 240@mat.ucsb.edu mailing list
- Example code archive
- Available software development tools

MAT 240E

stp@mat.ucsb.edu

8

Course Topics

- Synthesis, control, and performance
- MIDI
 - The MIDI HW and SW spec.
 - MIDI drivers and APIs
 - Scheduling and MIDI I/O
- The Open Sound Control (OSC) protocol
 - OSC address space design
 - OSC implementation, network programming
 - OSC synthesis servers in CSL and Supercollider
- The USB human interface device (HID) API
 - HID programming and gesture sensors
- Audio streaming between applications
 - The RFS, RTP and RTCP protocols

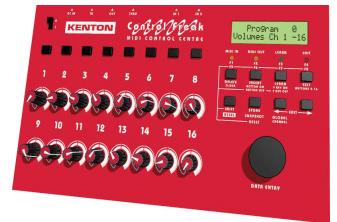
MAT 240E

stp@mat.ucsb.edu

9

Course Organization

- Theory, math
- Interfaces, hardware
- Protocol APIs
- Streaming APIs
- Role of development projects
- Tool-building



MAT 240E

stp@mat.ucsb.edu

10

MAT 240E Applications

- Required flexibility/configurability
- Required multi-threading
- Integration of control API and interface configuration
- Scheduling and sequencers
- Data massaging, smoothing, etc.

MAT 240E

stp@mat.ucsb.edu

11

Project Examples

- “Score” players
- Real-time generative “composition”
- Control recorders, editors, sequencers
- Input mappers
- Interactive PA/CSL/VST soft-synths (240D)
- HW interfaces (P5, WiiMote)
- RTP streamer
- CRAM for AlloSphere

MAT 240E

stp@mat.ucsb.edu

12

Basic Tools



- MIDI capture, dump utility, sequencer, editor/librarian
- Platform MIDI SDK, PortMIDI SDK
- OSC SDK
- CREATE CUI card or peer

MAT 240E

stp@mat.ucsb.edu

13

Review: Digital Audio Programming

- Digital audio data
 - Sampling and quantization
 - Data types for audio data
- Audio I/O APIs
 - See MAT 240A slides, web site, code archive
 - Call-back functions and buffer handling
 - Platform-specific: DirectX, CoreAudio, ALSA
 - Cross-platform: PortAudio, SDL, LibSndFile
 - Language-specific: JavaMF, Siren, CLM
- Synthesis techniques (see MAT 240D)
- Programming techniques (see MAT 201B)

MAT 240E

stp@mat.ucsb.edu

14

MAT 240E Part I



MAT 240E

stp@mat.ucsb.edu

15

Part I: Control Protocols and Sensor I/O

- Control data vs. content data
 - Control data manipulation
 - Interactive control APIs
 - Multi-threaded programming
- Control APIs
 - The MIDI standard
 - Open Sound Control
 - USB HID drivers



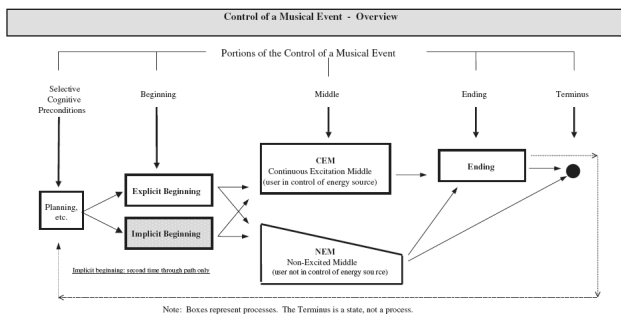
FUTUREMUSIC.COM

MAT 240E

stp@mat.ucsb.edu

16

Musical Events and Triggers

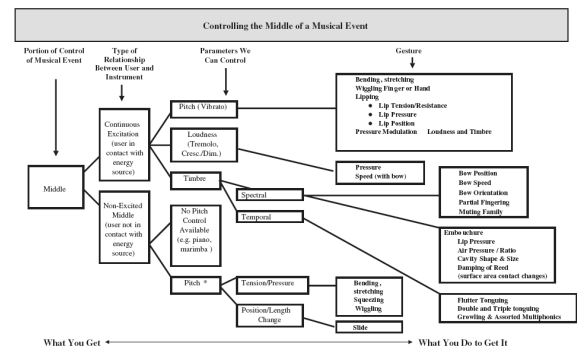


MAT 240E

stp@mat.ucsb.edu

17

Continuous Control



* Only if the player can contact some part of the tone generator during the Non-Excited Middle.

MAT 240E

stp@mat.ucsb.edu

18

Control vs. Content

- Media content
 - Constant-rate streaming
 - Large-ish fixed-size buffers
 - LAN-based apps generally have small number of sources/sinks and point-to-point connections
 - Connection-, QoS-oriented protocols (TCP, RTP)
- Control
 - Bursty and unpredictable
 - Packets are small and variable size
 - We often want multiple sources and sinks streaming (multicasting) on a LAN
 - Connection-less protocols (UDP)

MAT 240E

stp@mat.ucsb.edu

19

Semantics of Control

- How is control used in interactive applications?
- Control from simulations
 - Events and low-rate control
 - Timing alternatives
- Control from algorithmic drivers
 - Input to the algorithms
 - Flexible scheduling
- Control from sensors
 - Cache/map/filter layers
 - Applying to a model



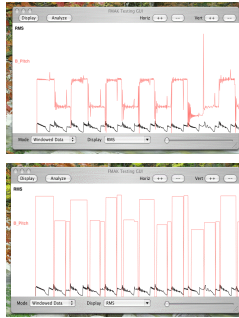
MAT 240E

stp@mat.ucsb.edu

20

Control Data Processing

- Input processing
 - Data caching, buffering
 - Ranges: offsets and scaling
 - Smoothing: averages, lags, and edge-sharpening
 - Mapping: functions and tables
- Applying to the model
 - Update timing and caching
- Basic code examples



MAT 240E

stp@mat.ucsb.edu

21

Massaging Input Data

```
float x = getValue(); // get a value
if ((x - cachedX) < xThreshold) // check delta
    continue;
scaledX = (x * xScale) + xOffset; // scale & offset
mappedX = xMap[(unsigned)floor(scaledX)]; // look-up

// apply lo-pass filter
// apply exp. lag
// apply smoothing, moving average

cachedX = mappedX;
globalX = cachedX;
this.changed("x");
```

MAT 240E

stp@mat.ucsb.edu

22

Synchronizing Control and Content

- Drivers/OS-level call-backs determine synchronous DSP processing or rendering (probably in a kernel thread)
- Sensors and composition routines generate bursty parameter updates (may be polled/buffered/cached in user threads)
- Latency and latency jitter control are issues
- Use smart buffering with accurate time-stamps

MAT 240E

stp@mat.ucsb.edu

23

Multi-threaded Interactive Programming

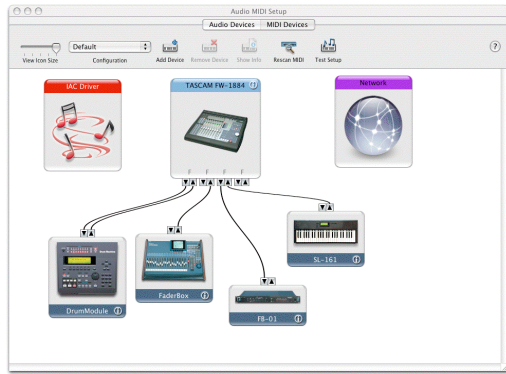
- Scenario: software synth using PortAudio thread with a separate callback thread for sensor data changing global parameters
- Issues:
 - Using shared data between threads
 - Signaling and triggering (parameter updates during DSP callback)
 - Streaming control inputs
- Main code examples
- Thread loop examples

MAT 240E

stp@mat.ucsb.edu

24

MIDI



MAT 240E

stp@mat.ucsb.edu

25

MIDI



- Musical Instruments Digital Interface (Smith 1983)
- Simple low-level serial protocol implemented over low-cost (current loop) P2P network
- Isochronous communication, some notions of clock and tempo, (mostly) stateless receiver
- Most commands are 2 or 3 bytes
- Also used for lighting, motor control, etc.
- Implemented over USB, etc.
- Terrible design, ubiquitous in music apps.

MAT 240E

stp@mat.ucsb.edu

26

- 1980-82
 - Seq. Circuits Universal Synth Interface
 - Roland Digital Control Bus
- Pre-PC days
 - Hardware controllers, sound modules, sequencers
- Post-1983
 - MIDI on Atari, Amiga
 - Mac MIDI + SCSI
 - Roland MPU for PCs
- 1990-92
 - General MIDI
 - File formats
- Post-1992
 - 4X MIDI, ZIPI, OSC, MIDI-over-USB

MIDI History

MAT 240E

stp@mat.ucsb.edu

27

Whose Fault is it Anyway?



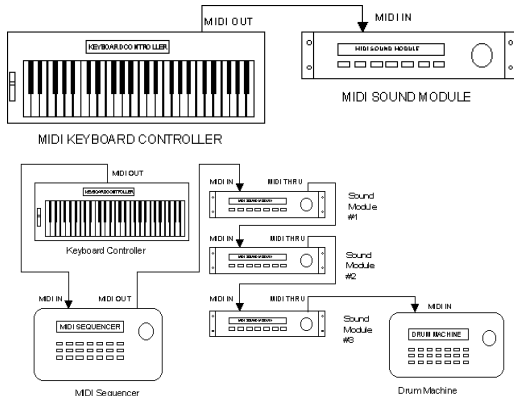
(S) Pope), R Linn, B Linn, D Rossum,
D Massie, J Snell, D Smith, G Loy

MAT 240E

stp@mat.ucsb.edu

28

MIDI Networks



MAT 240E

stp@mat.ucsb.edu

29

MIDI Details

- Keyboard-centric paradigm: note/key/velocity
- Weak parameter models, no mapping standards
 - Pitch as key number, amplitude as key velocity, timbre as channel
- Extensible via system-exclusive (SYSEX) messages
- Continuous control as controller updates (easy to soak up all bandwidth)
- Limited to 16 channels/voices, short cable runs, 31 kHz bandwidth
- Requires voice-stealing policies for polyphony
- Design pre-dated computer interfaces

MAT 240E

stp@mat.ucsb.edu

30

MIDI Command Examples

- Note-on, channel 0, middle-C, mezzo-forte
 - 90, 60, 70
- Note-off, channel 0, middle-C
 - 80, 60, 00 (or cmd 90 with velocity 0)
- Send channel 2, controller 14 value 127
 - B2, 14, 7F
- Patch-change, channel 9 to pgm/inst/bank 1
 - C9, 01

MAT 240E

stp@mat.ucsb.edu

37

Extended MIDI Messages

- Song select and index
- Timing, tempo
- Tuning
- Active sense
- System reset
- (see refs)



MAT 240E

stp@mat.ucsb.edu

38

MIDI Sysex Messages

- Extended standard commands
 - Note-on with duration (e.g., FB-01 table)
 - Note-on with fractional key # (microtonal)
- Extended controllers (also hi-res)
- Voice editing (synth model)
- Sample dumps (sound files)
- Sequencing (event lists)
- Format: header, manufacturer ID, command, data chunks, (checksum,) footer

MAT 240E

stp@mat.ucsb.edu

39

MIDI SysEx Format

- Example (Roland Juno 106)

```
F0 [Exclusive]
41 [Roland ID#]
32 [Function Type]
0N [N+1=MIDI channel]
XX [Parameter number]
YY [Variable value (0-127)]
F7 [End of Exclusive]
```

- Manufacturer IDs

	not used	American	European	Japanese	Other	Special
1 byte ID:	00	01 -- 1F	20 -- 3F	40 -- 5F	60 -- 7C	7D -- 7F
3 byte ID:	00 00 00	00 00 01	00 20 00	00 40 00	00 60 00	
		00 1F 7F	00 3F 7F	00 5F 7F	00 7F 7F	

7D: non-commercial use (e.g. schools, research, etc.)
 7E: Non-Real Time Universal System Exclusive ID
 7F: Real Time Universal System Exclusive ID (all call device ID)

Parameter Numbers (XX) For The Juno-106:

CODE (XX)	EDIT PARAMETER	COMPLETE SYSEX CODE
00	LFO rate	F0 41 32 00 00 00 F7
01	LFO delay	F0 41 32 00 01 00 F7
02	DCO LFO modulation level	F0 41 32 00 02 00 F7
03	DCO PWM modulation level	F0 41 32 00 03 00 F7
04	Noise level	F0 41 32 00 04 00 F7
05	VCF cutoff level	F0 41 32 00 05 00 F7
06	VCF resonance level	F0 41 32 00 06 00 F7
07	VCF ENV level	F0 41 32 00 07 00 F7
08	VCF LFO modulation level	F0 41 32 00 08 00 F7
09	VCF KBD modulation level	F0 41 32 00 09 00 F7
0A	VCA level	F0 41 32 00 0A 00 F7
0B	ENV attack rate	F0 41 32 00 0B 00 F7
0C	ENV decay rate	F0 41 32 00 0C 00 F7
0D	ENV sustain level	F0 41 32 00 0D 00 F7
0E	ENV release rate	F0 41 32 00 0E 00 F7
0F	Sub level	F0 41 32 00 0F 00 F7

MAT 240E

stp@mat.ucsb.edu

40

MIDI SysEx Utilities

- Utilities and APIs
- Record/playback of large SysEx messages
- Handling of sample dumps and sample bank management
- Advanced sequencing apps
- Really interesting possibilities
- Nightmare because of lack of standardization (and thousands of synth models)

MAT 240E

stp@mat.ucsb.edu

41

MIDI File Format

- Header chunk


```
struct MTHD_CHUNK {
    char ID[4]; // This will be 'M','T','h','d'
    unsigned long Length; // This will be 6
    unsigned short Format; // 0, 1, or 2
    unsigned short NumTracks; // # of tracks
    unsigned short Division; // PPQN
};
```
- Track chunk


```
struct MTRK_CHUNK {
    char ID[4]; // This will be 'M','T','r','k'
    unsigned long Length; // This will be the size of Data[]
    unsigned char Data[]; // Its actual size is Data[Length]
};
```
- Variable-length quantities like time-stamps (7 bits)

MAT 240E

stp@mat.ucsb.edu

42

MIDI File Format 0 Example

4D 54 68 64	MThd
00 00 00 06	chunk length
00 00	format
0 00 01	one track
00 60 96	per quarter-note
4D 54 72 6B	MTrk
00 00 00 3B	chunk length (59)
00FF 58 04 04 02 18 08	time signature
00 FF 51 03 07 A1 20	tempo
00 C0 05	Some program changes
00 C1 2E	
00 92 30 60	Play some notes
00 3C 60	running status
6091 43 40	
81 40 82 30 40	note-off with two-byte delta-time
003C 40	running status
0081 43 40	
00 FF 2F 00	end of track

MAT 240E

stp@mat.ucsb.edu

43

MIDI File Details



Meta-events

- Tempo, time-sig, key-sig
- Markers, instrument names, lyrics, copyright notices, etc.

4D 54 72 6B	MTrk ID
00 00 00 0C	Length of the chunk so far
00	Our Track Name Meta-Event
FF	Delta-time is 0.
03	A Meta-Event.
08	Track Name type.
4D	Length of "My track"
79	ASCII 'M'
20	ASCII 'y'
74	ASCII ' ' (space)
72	ASCII 't'
61	ASCII 'r'
63	ASCII 'a'
68	ASCII 'c'
	ASCII 'k'

MAT 240E

stp@mat.ucsb.edu

44

MIDI Input



- MIDI event sources
 - Select, trigger note-on/off
- MIDI continuous control sources
 - Range/value mapping
- Instrument models
 - Keyboard, drum (obvious trigger mappings)
 - Wind, string (less so, more continuous control)
- Tools
 - Mixers, transport/graphics controls, weapons
- Other models

MAT 240E

stp@mat.ucsb.edu

45

MIDI Sources

Sensor/Controllers

- Keyboard, drum, guitar, wind
- Breath controller, gun
- Mixer, fader, joy-stick, pedal
- Pitch-to-MIDI: VoiceRider, fiddle~, bonk~
- Programmable sensor interfaces (SensorLab, iCube, etc.)

Programs

- Sequencer, score follower
- Algorithmic composers
- Control mappers

Yamaha WX5
MIDI Controller



MAT 240E

stp@mat.ucsb.edu

46

History: EMS Pitch-to-CV

PITCH-TO-VOLTAGE CONVERTER

(WITH BUILT-IN VOLTAGE CONTROLLED SAWTOOTH OSCILLATOR AND ENVELOPE FOLLOWER)



MAT 240E

stp@mat.ucsb.edu

47

Keyboard & Drum Controllers



MAT 240E

stp@mat.ucsb.edu

48

Faders and Control Surfaces



49

Alternative Sources

- Buchla Thunder & Lightning
- JazzMutant Lemur
- VideoHarp
- AirDrums
- See *Proc NIME*, *Proc ICMC*
- <http://createdigitalmusic.com/tag/alternative-controllers/>

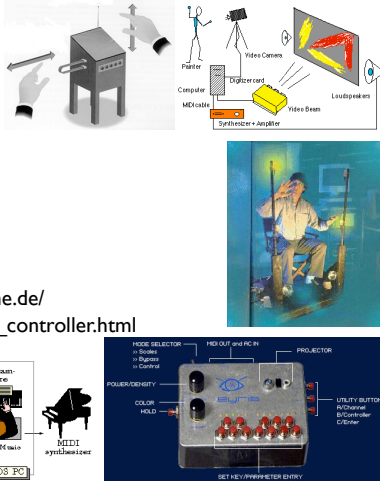


MAT 240E

50

Non-contact Controllers

- Theremin
- Video
- Infrared
- Magnetic
- http://www.stoffelshome.de/alt_controller/alt_midi_controller.html



51

Hybrids

- Hybrid vs. multi-way interface
- Drum/keyboard
- Guitar/keyboard
- Mixer/drum/keyboard
- Computer/instrument
- Game controllers



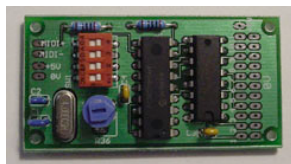
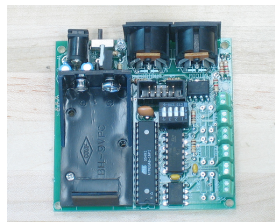
MAT 240E

stp@mat.ucsb.edu

52

MIDI Capture

- Gesture-to-MIDI
 - Steim SensorLab (RIP)
 - iCube systems
 - MIDI-Sense
 - MAnMIDI
- Pitch-to-MIDI
 - HW
 - IVL VoiceRider
 - Roland guitars
 - SW
 - MAX fiddle~ and bonk~

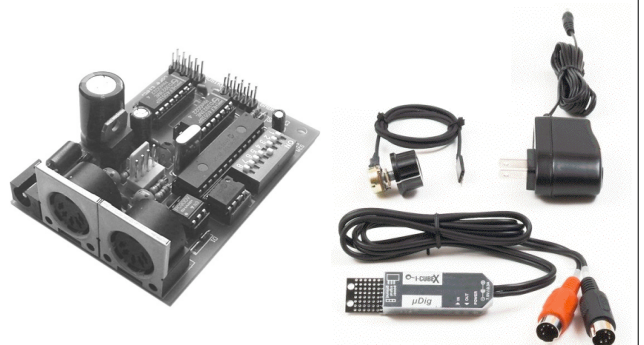


MAT 240E

stp@mat.ucsb.edu

53

Doepfer & iCube MIDI HW



MAT 240E

stp@mat.ucsb.edu

54

MIDI Sinks

- MIDI-controlled sound modules
 - General MIDI
 - Drums
 - Phys. Models
 - Samplers
- MIDI motors
- MIDI dimmers
- MIDI-sensing programs
 - Mapper/controllers
 - Sequencers



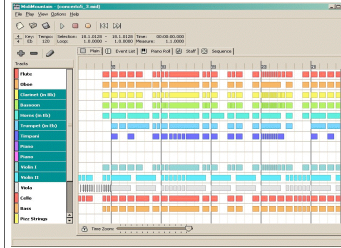
MAT 240E

stp@mat.ucsb.edu

55

MIDI Sequencers

- Capture (step input, record), edit, playback
- Transport controls, track view



MAT 240E

stp@mat.ucsb.edu

56

MIDI Sound Modules

- FM/Wavetable
- Sampler
- Drum synth



MAT 240E

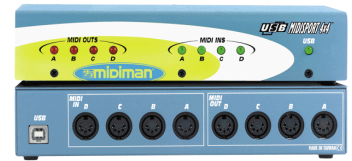
stp@mat.ucsb.edu

57

MIDI Interfaces



- Stand-alone
 - (Serial,) USB, PCI
- Built into audio interface
 - Ubiquitous
 - Good FireWire drivers



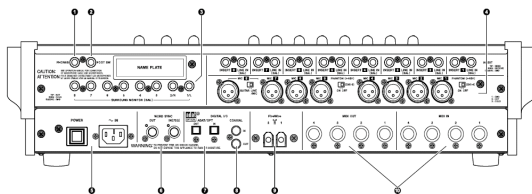
MAT 240E

stp@mat.ucsb.edu

58

Current Interfaces

- Built into most USB or FireWire AD/DA boxes



MAT 240E

stp@mat.ucsb.edu

59

Using MIDI

- Soft-synths: QTMusic, RAX, DirectX, DLS
- VSTi and other plug-in instrument hosts
- MIDI in web browsers
- MIDI in Flash
- SW-only solutions
- Managing MIDI HW: interface mappers, voice editors, bank managers, other tools
- Code examples
 - Playing over built-in soft-synth
 - Recording from the keyboard and knobs

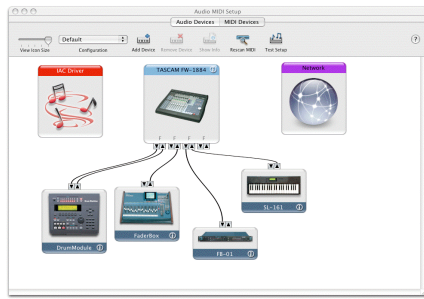
MAT 240E

stp@mat.ucsb.edu

60

MIDI System Configuration

- Sources
- Filters
- Sinks
- Computer interfaces

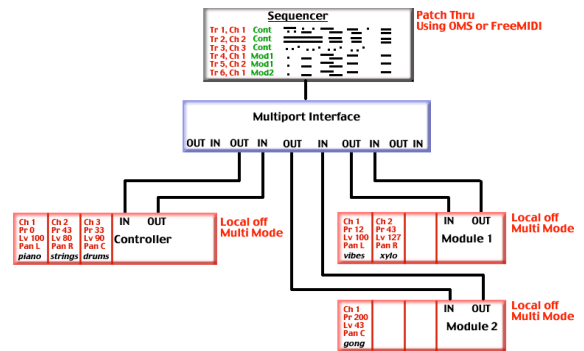


MAT 240E

stp@mat.ucsb.edu

61

Example

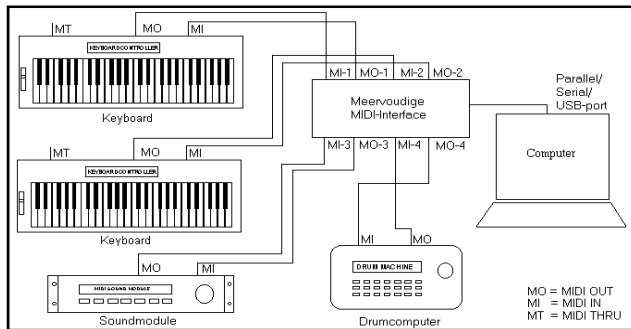


MAT 240E

stp@mat.ucsb.edu

62

Example



MAT 240E

stp@mat.ucsb.edu

63

MIDI Patch Bays



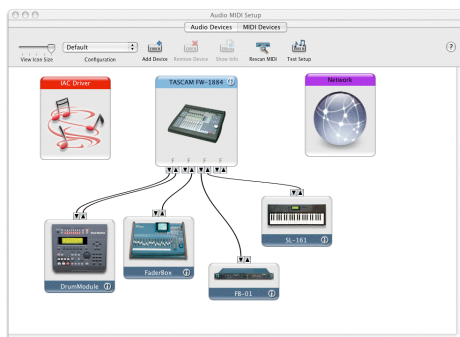
- Support driver-level hardware and inter-application MIDI routing
- Use a model of virtual/real devices with MIDI I/O ports
- Route channels, manage interfaces
- Provide an API for new devices
- Provide an API for new applications

MAT 240E

stp@mat.ucsb.edu

64

MIDI Patch Bay Example



MAT 240E

stp@mat.ucsb.edu

65

Patch Editors and MIDI HW

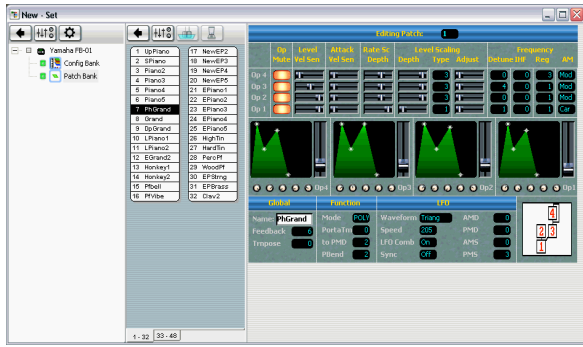
- Configure a MIDI sound module
 - Synth. technique (algorithm) selection
 - Constant parameters
 - Break-point functions (envelopes)
 - LFOs and dynamics
- Tuning, MIDI input maps, kbd splits
 - Controller maps
- Bank save/restore (patch librarian)
- Very system specific (& complex & messy)

MAT 240E

stp@mat.ucsb.edu

66

Editing & Librarian Functions



Example: Yamaha FB-01 Editor

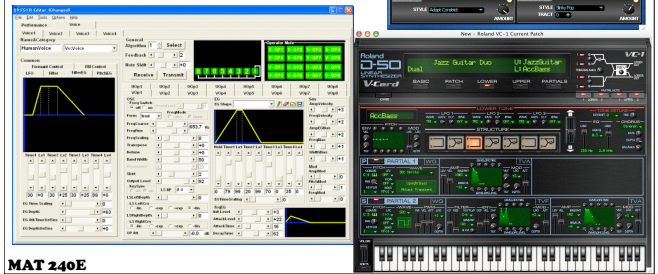
MAT 240E

stp@mat.ucsb.edu

67

Patch Editors

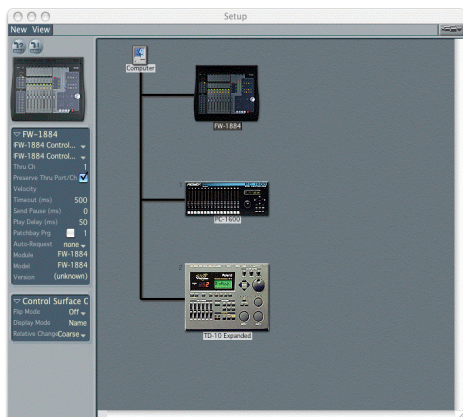
- Hardware-specific
- “Universal”



MAT 240E

68

SoundDiver Main Set-up

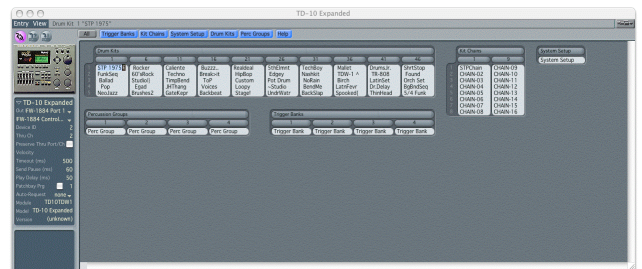


MAT 240E

mat.ucsb.edu

69

SoundDiver TD-10 Drum Synth Editor

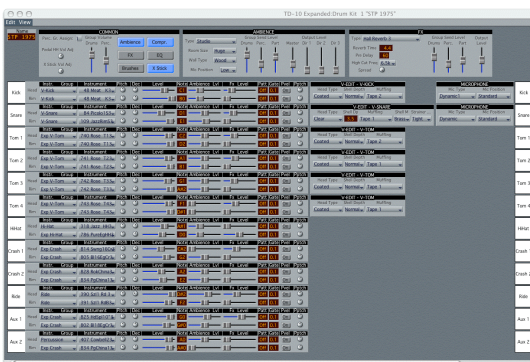


MAT 240E

stp@mat.ucsb.edu

70

SoundDiver TD-10 Editor

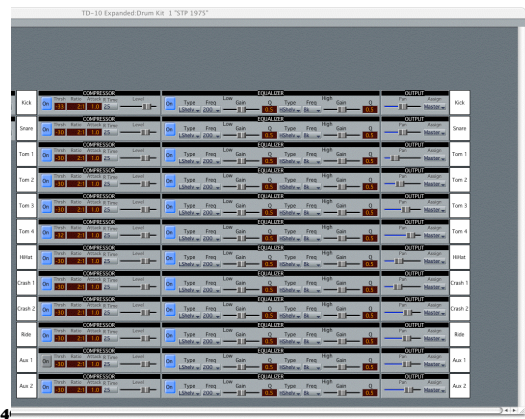


MAT 240E

stp@mat.ucsb.edu

71

SoundDiver TD-10 Editor

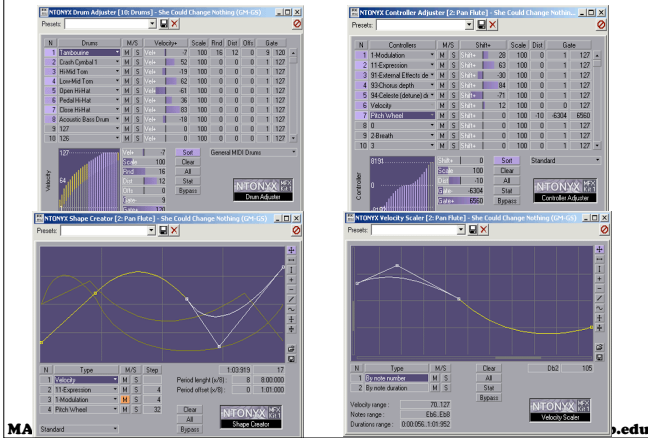


MAT 240E

ucsb.edu

72

NTONYX MFX Kit



73



74

GeneralMIDI

- Problem: no standard mapping between MIDI channel # and any given synth's "orchestra"
- Solution: default assignments for 128 sounds in banks by instrument family
- GM can be implemented by sampling, FM, wavetables, etc.
- Still no standard for timbre, velocity map...

MAT 240E

stp@mat.ucsb.edu

75

General MIDI Standards

TABLE 1 - General MIDI Instrument Patch Map			
(groups sounds into sixteen families, with 8 instruments in each family)			
Program	Instrument	Program	Instrument
1-8 PIANO	9-16 CHORDAL	17-24 ORGAN	25-32 GUITAR
1 Acoustic Grand	9 Celesta	17 Drawbar Organ	25 Acoustic Guitar(nylon)
2 Bright Acoustic	10 Glockenspiel	18 Percussive Organ	26 Acoustic Guitar(steel)
3 Electric Grand	11 Music Box	19 Rock Organ	27 Electric Guitar(jazz)
4 Honky-Tonk	12 Vibraphone	20 Church Organ	28 Electric Guitar(clean)
5 Electric Piano 1	13 Marimba	21 Reed Organ	29 Electric Guitar(muted)
6 Electric Piano 2	14 Xylophone	22 Accordion	30 Overdriven Guitar
7 Harpsichord	15 Tubular Bells	23 Harmonica	31 Distortion Guitar
8 Clav	16 Dulcimer	24 Tango Accordion	32 Guitar Harmonics
33-40 BASS	41-48 STRINGS	49-56 ENSEMBLE	57-64 BRASS
33 Acoustic Bass	41 Violin	49 String Ensemble 1	57 Trumpet
34 Electric Bass(finger)	42 Viola	50 String Ensemble 2	58 Trombone
35 Electric Bass(pick)	43 Cello	51 SynthStrings 1	59 Tuba
36 Fretless Bass	44 Contrabass	52 SynthStrings 2	60 Muted Trumpet
37 Slap Bass 1	45 Tremolo Strings	53 Choir Aahs	61 French Horn
38 Slap Bass 2	46 Piccolato Strings	54 Voice Oohs	62 Brass Section 1
39 Synth Bass 1	47 Orchestral Strings	55 Synth Voice	63 Synthbrass 1
40 Synth Bass 2	48 Timpani	56 Orchestra Hit	64 Synthbrass 2
65-72 REED	73-80 PIPE	81-88 SYNTH LEAD	89-96 SYNTH PAD
65 Soprano Sax	73 Piccolo	81 Lead 1 (square)	89 Pad 1 (new age)
66 Alto Sax	74 Flute	82 Lead 2 (sawtooth)	90 Pad 2 (warm)
67 Tenor Sax	75 Recorder	83 Lead 3 (challenger)	91 Pad 3 (polysynth)
68 Baritone Sax	76 Pan Flute	84 Lead 4 (chiff)	92 Pad 4 (choir)
69 Oboe	77 Blown Bottle	85 Lead 5 (charang)	93 Pad 5 (bowed)
70 English Horn	78 Shakuhachi	86 Lead 6 (voice)	94 Pad 6 (metallic)
71 Bassoon	79 Whistle	87 Lead 7 (fifths)	95 Pad 7 (halo)
72 Clarinet	80 Ocarina	88 Lead 8 (bass-lead)	96 Pad 8 (sweep)
97-104 SYNTH EFFECTS	105-112 ETHNIC	113-120 PERCUSSIVE	121-128 SOUND EFFECTS
97 FX 1 (rain)	105 Sitar	113 Tinkle Bell	121 Guitar Fret Noise
98 FX 2 (soundtrack)	106 Banjo	114 Agogo	122 Breath Noise
99 FX 3 (crystal)	107 Shamisen	115 Steel Drums	123 Seashore
100 FX 4 (atmosphere)	108 Koto	116 Woodblock	124 Bird Tweet
101 FX 5 (brightness)	109 Kalimba	117 Talko Drum	125 Telephone Ring
102 FX 6 (goblins)	110 Maracas	118 Melodic Tom	126 Helicopter
103 FX 7 (echoes)	111 Fiddle	119 Synth Drum	127 Applause
104 FX 8 (sci-fi)	112 Shanai	120 Reverse Cymbal	128 Gunshot

MAT 240E

stp@mat.ucsb.edu

76

MIDI++

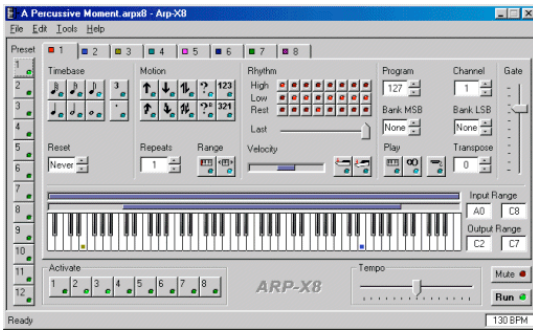
- Rich literature on "The dysfunctions of MIDI"
- Limits on # channels, continuous control
- Low-level models of pitch, amplitude, etc.
- Timing inaccuracies, low throughput
- Note on/off event model
- Many attempts to make it better
- 4xMIDI, ZIP, mLAN, OSC
- Separate control from event triggering
- Flexible models of pitch, amplitude, timbre
- Scalable continuous controls
- Run on top of a faster, wider-area, routed network

MAT 240E

stp@mat.ucsb.edu

78

MIDI Software



MAT 240E

stp@mat.ucsb.edu

79

Programming: MIDI APIs

- APIs for low-level MIDI I/O in most modern OSs (some still based on serial driver APIs)
- Driver-level mapping to multiple interfaces and HW/SW devices
- Manage SYSEX up/down-loads
- Store configurations (virtual studio)
- MIDI file I/O (opt.)
- API structs: ports, devices, messages, controller maps, timers, etc.

MAT 240E

stp@mat.ucsb.edu

80

MIDI API Outline



- Query attached devices and virtual device mapping (port/device enumeration, patch bays)
- Open/close stream/callback/filter set-up
- Poll driver for pending input (since it's bursty), or register a event handler call-back (with time-out)
- Schedule (small or large) messages for output
- Handle SysEx load/store transactions (voice editor/librarians, sample dump I/O)
- Timing and scheduling primitives

MAT 240E

stp@mat.ucsb.edu

81

MIDI Output on MS-Windows

```
HMIDIOUT device;           // device for sending MIDI output
union { unsigned long word; unsigned char data[4]; } message;

message.data[0] = 0x90;      // MIDI note-on message
message.data[1] = 60;        // Key number (60 = middle C)
message.data[2] = 100;       // Key velocity (100 = loud)

midiOutOpen(&device, 0, 0, 0, CALLBACK_NULL);

midiOutShortMsg(device, message.word);
```

MAT 240E

stp@mat.ucsb.edu

82

COREMIDI API Example

```
static void MyReadProc(MIDIPacketList *pktlist, void *data,
    void *connData) {
    if (gOutPort != NULL && gDest != NULL) {
        MIDIPacket *packet = (MIDIPacket *)pktlist->packet;
        for (unsigned int j = 0; j < pktlist->numPackets; ++j) {
            for (int i = 0; i < packet->length; ++i)
                printf("%02X ", packet->data[i]);
            printf("\n");
            packet = MIDIPacketNext(packet);
        }
        MIDISend(gOutPort, gDest, pktlist);    // echo input
    }
}
```

MAT 240E

stp@mat.ucsb.edu

83

Setup/Registration Code

```
// Include files
#include <CoreMIDI/MIDIServices.h>
#include <CoreFoundation/CFRunLoop.h>

// Global obj-refs
MIDIPortRef gOutPort = NULL;
MIDIEndpointRef gDest = NULL;

// Create client object for IAC API
MIDIClientRef client = NULL;
MIDIClientCreate(CFSTR("MIDI Echo"), NULL, NULL, &client);

// Create IO ports and assign call-back functions
MIDIPortRef inPort = NULL;
MIDIInputPortCreate(client, CFSTR("Input port"), MyReadProc, NULL, &inPort);
MIDIOutputPortCreate(client, CFSTR("Output port"), &gOutPort);
```

MAT 240E

stp@mat.ucsb.edu

84

Device Enumeration, Connection

```
n = MIDIGetNumberOfDevices();
for (i = 0; i < n; ++i) {
    MIDIDeviceRef dev = MIDIGetDevice(i);
    MIDIObjectGetStringProperty(dev, kMIDIPropertyName, &name);
    MIDIObjectGetStringProperty(dev, kMIDIPropertyManufacturer, &manuf);
    MIDIObjectGetStringProperty(dev, kMIDIPropertyModel, &model);
    printf("name=%s, manuf=%s, model=%s\n", name, manuf, model);
}

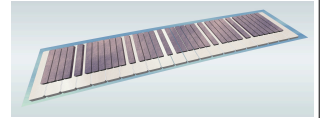
// open connections from all sources
n = MIDIGetNumberOfSources();
printf("%d sources\n", n);
for (i = 0; i < n; ++i) {
    MIDIEndpointRef src = MIDIGetSource(i);
    MIDIPortConnectSource(inPort, src, NULL);
}
```

MAT 240E

stp@mat.ucsb.edu

85

PortMIDI



- Related to PortAudio
- No call-backs
 - Blocking read(), waits for input
 - Non-blocking poll() checks for available input
- Open/close functions for in/out
- See portmidi.h and CSL examples
- See also other MIDI APIs:
 - RtMIDI (Scavone@McGill)
 - Linux /dev/midi and /dev/sequencer)

MAT 240E

stp@mat.ucsb.edu

86

PortMIDI Example

```
Pm_OpenOutput(&midi, out,          // open stream
              DRIVER_INFO, OUTPUT_BUFFER_SIZE,
              TIME_PROC, TIME_INFO, latency);
Pm_SetFilter(midi, PM_FILT_ACTIVE | PM_FILT_CLOCK);
status = Pm_Poll(midi);           // poll, check status
length = Pm_Read(midi, buffer, 1); // read an event
if (length > 0) {                 // dump it
    printf("Got message @ time %d, %2x %2x %2x\n",
           buffer[0].timestamp,
           Pm_MessageStatus(buffer[0].message),
           Pm_MessageData1(buffer[0].message),
           Pm_MessageData2(buffer[0].message));
}
```

MAT 240E

stp@mat.ucsb.edu

87

RtMIDI

- Simple, 2 C++ classes (RtMidiIn and RtMidiOut)
- Cross-platform implementation
- Supports device enumeration and queries
- Polling via


```
time_stamp = midi_in->getMessage( &message );
```

 or


```
midi_in->setCallback( &mycallback );
```
- Virtual port/device APIs supported

MAT 240E

stp@mat.ucsb.edu

88

RtMIDI Examples



```
RtMidiOut *midiout = 0;
std::vector<unsigned char> message;

// RtMidiOut constructor
midiout = new RtMidiOut();
midiout->openPort(1);

// Send out a MIDI message
// Program change: 192, 5
message.push_back( 192 );
message.push_back( 5 );
midiout->sendMessage( &message );
```

MAT 240E

stp@mat.ucsb.edu

89

Coding Examples

- EventList player
 - Simple MIDI-like commands
 - Score I I
 - “Little languages for music”
- Event & control recorder
- Score/control display (in JUCE, in OpenGL)
- CSL instruments
- VSTi clients
- VSTi host

MAT 240E

stp@mat.ucsb.edu

90

Score I I-lite Example

- Instrument blocks
- Parameter commands
- Value lists
- Motives
- Random masks, ranges

```
INSTRUMENT 10 0 10
  DURATION Rhythm 4//8//4//8//2/
  PITCH Notes c/cs//d//df//c//ds////////
  AMPLITUDE Move 5 0.01 0.7 / 5 0.7 0.01
  POSITION Random -1 1
CHANNEL 1
end
```

MAT 240E

stp@mat.ucsb.edu

91

MIDI Instrument APIs

- Plug-in host program
 - Dynamic loader to load plug-ins
 - Call their call-backs and manage buffers
- Instrument host
 - Route MIDI to plug-ins
 - They're soft-synths (MIDI-in, Audio-out)
- VST and VSTi
 - Audio call-back -- *process(in**, out**, #frames)*
 - Event control call-back -- *processEvents(events)*
- See MAT 240D

MAT 240E

stp@mat.ucsb.edu

92

VST Instrument Class

```
class VstXSynth : public AudioEffectX {           // Derive AE_X
void processReplacing (float **inputs,           // do-DSP
                      float **outputs, long numFrames); // (see MAT 240D)
long processEvents (VstEvents *events);          // do-MIDI
void setParameter (long index, float value);     // get/set params
float getParameter (long index);
long getMidiProgramName (long channel,           // MIDI features
                        MidiProgramName* midiProgramName);
long getCurrentMidiProgram (long channel,
                           MidiProgramName* currentProgram);
long getMidiProgramCategory (long channel,
                             MidiProgramCategory* category);
bool hasMidiProgramsChanged (long channel);
virtual VstInt32 getNumMidiInputChannels ();
}
```

MAT 240E

stp@mat.ucsb.edu

93

VSTi *processEvents()* Method

```
VstInt32 VstXSynth::processEvents (VstEvents* ev) {
for (VstInt32 i = 0; i < ev->numEvents; i++) {
if ((ev->events[i])>-type != kVstMidiType) continue;
VstMidiEvent* event = (VstMidiEvent*)ev->events[i];
char* midiData = event->midiData;
VstInt32 status = midiData[0] & 0xf0;           // ignoring channel
if (status == 0x90 || status == 0x80) {         // we only look at notes
VstInt32 note = midiData[1] & 0x7f;
VstInt32 velocity = midiData[2] & 0x7f;
if (status == 0x80) velocity = 0;               // note off by velocity 0
if (!velocity && (note == currentNote))
noteOff ();
else
noteOn (note, velocity, event->deltaFrames);
} else if (status == 0xb0) {
if (midiData[1] == 0x7e || midiData[1] == 0x7b) // all notes off
noteOff ();
}
event++;
}
return 1;
}
```

MAT 240E

stp@mat.ucsb.edu

94

VST Instruments



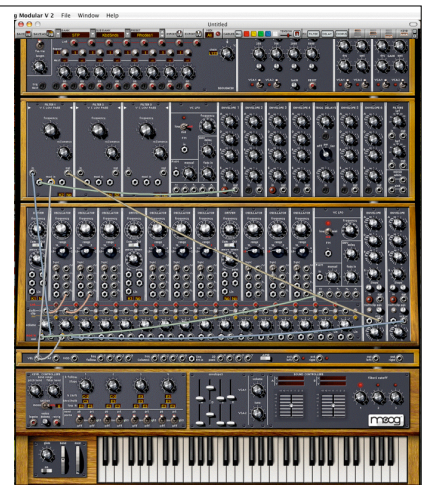
MAT 240E

stp@mat.ucsb.edu

95

Moog Modular VST Instrument

- Runs as VST, AU, RTAS, MAS, DXi, or HTDM plug-in or as a stand-alone app.



MAT 240E

96

So, let's build an API!

- Create MIDI_240.h
 - Basic data structures
 - Error/exception flags, sys. constants
 - Ports, devices
 - Events, messages, controllers, timing
 - Tracks, songs
 - Voices, samples, and synths
 - MIDI device management
 - Enumeration/query, name list
 - Init/terminate
 - Open/close

MAT 240E

stp@mat.ucsb.edu

97

240_MIDI.h cont'd

- Basic MIDI I/O
 - Core messages, active note mgmnt
 - Named commands and controllers
 - Reading: polling vs call-back
- SysEx
 - Extended command formats
 - Voice data up-load
- MIDI file IO
- Scheduler?
- Implement on top of other APIs
 - How to make multi-platform (1 vs. many files)

MAT 240E

stp@mat.ucsb.edu

98

What apps to start with

- Real-time output
 - Score player or algorithmic composer
- Real-time input
 - Event recorder/editor
 - Add good smoothing, massaging, configuration
- Both = sequencer
- Voice editors (e.g., for the FB-01)
- Input mapper to an app (control via faders)
- GUI apps with MIDI in and sound out
 - JUCE faders, button keyboard
 - DSP patcher

MAT 240E

stp@mat.ucsb.edu

99

Squeak Smalltalk MIDI Example Apps

stp@mat.ucsb.edu

100

Review: MIDI

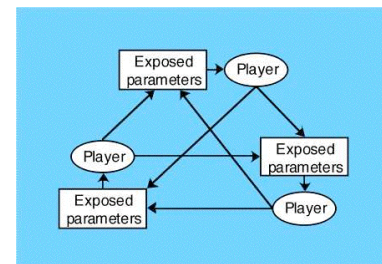
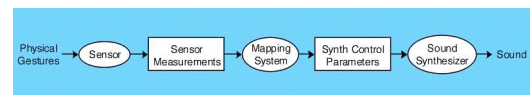
- MIDI
 - Hardware spec, serial protocol
 - Drivers and APIs
- Using MIDI
 - Controllers
 - HW/SW synths
- Writing MIDI soft-synths (VSTi, AU, etc.)
- Writing a new MIDI API
- Writing MIDI Utilities

MAT 240E

stp@mat.ucsb.edu

101

Topic: OSC



MAT 240E

stp@mat.ucsb.edu

102

Open Sound Control (OSC)

- UCB/CNMAT, derived from *ZIPI* (1990-94) and *SynthControl* (1994-96)
- Network protocol based on UDP/IP (or any other fast transport)
- Every parameter of every instrument has a unique address within a hierarchical namespace
- Commands also fit into this hierarchy
- Servers are assumed to have schedulers
- Soft-synth servers receive OSC
- Controller hardware sends OSC

MAT 240E

stp@mat.ucsb.edu

103

OSC Messages

- OSC addresses: UNIX hierarchical file-name style with optional wild cards
 /guitar/2/string/4/frequency -- parameter address
 /oscillator/*/phase -- multiple params
- Command: may be address + data (to set a param), or just the address (command name)
 /guitar/2/string/4/freq 440.0 -- set a variable
 /guitar/2/stop -- send a command
- Data types: int, float, string, blob

MAT 240E

stp@mat.ucsb.edu

104

OSC Command Bundles

- Bundles: several commands to take place at the same time can be bundled together (scheduled by receiving server)
- Annotated as,
 #bundle time-stamp
 list-of: size, bundle-item
- These may be nested with separate time-stamps (i.e., a hierarchical event list)

MAT 240E

stp@mat.ucsb.edu

105

OSC Features

- Open-ended, dynamic, URL-style symbolic naming scheme (needs good implementation)
- Numeric and symbolic arguments to messages
- Pattern-matching to specify multiple targets of a single message ("set all strings to quiet")
- High resolution time tags
- "Bundles" of messages whose effects must occur simultaneously (schedule-ahead)
- Query system to find out the capabilities of an OSC server and get documentation (discovery)

MAT 240E

stp@mat.ucsb.edu

106

Protocol Details

- All fields padded to multiples of 4 bytes
- Type strings used in commands
 - ,i ,f ,s ,ifffs etc.
- Bundles and containers
- Time-stamps and timing
- See examples in the spec at CNMAT
 - <http://www.cnmat.berkeley.edu/OpenSoundControl/OSC-spec.html>

MAT 240E

stp@mat.ucsb.edu

107

OSC Implementation

- CNMAT reference code (RIP)
 - Requires low-level UNIX networking
 - Needs better search/parsing for good performance with a large address space
 - Needs a router for large distributed systems
 - Includes *sendOSC* utility for testing
 - `sendOSC [-h host] [-r] port_number [message...]`
- Implemented by many interesting packages (SuperCollider, Max/MSP, Reaktor, Siren, CSL, Bidule, Traktor, OM, etc.)

MAT 240E

stp@mat.ucsb.edu

108

OSC-Native Systems

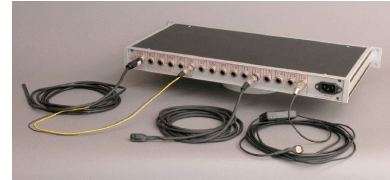
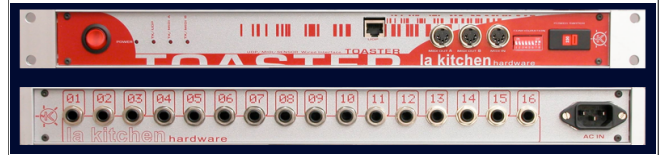
- Controllers
 - Kroonde, Lemur
- Interfaces
 - CUI, etc.
- Software
 - SC, CSL, Max/Mr, etc.



MAT 240E

109

Kitchen Toaster (RIP?)



MAT 240E

stp@mat.ucsb.edu

110

OSC Programming in C/C++

- LibLo low-level C API
 - Lightweight, can use IPv6 sockets
- OSCPack
 - RossBencina AudioMulch classes
- LibOSC++
- WOsclib (Weiss)
 - C++ OO wrappers for ports and messages

MAT 240E

stp@mat.ucsb.edu

111

OSC in Other Languages

- Other OSC APIs & Wrappers
 - NetUtil Java API
 - flosc in Flash
 - SirenOSC in Smalltalk
 - JavaOSC

MAT 240E

stp@mat.ucsb.edu

112

Using the OSC APIs

- Writing OSC
 - Open a socket on a port
 - Format a message to write
 - See CSL gesture sensor code
- Reading OSC
 - Straightforward call-back API
 - Listen for messages on a port
 - Register call-back functions at OSC addresses (variables or commands)
 - Main *select()* loop
 - See CSL instrument driver

MAT 240E

stp@mat.ucsb.edu

113

Simple GestureSensor API (EBeam example)

```
// Construct an OSC message
// The args are all floats
construct_OSC_Packet_For_Float_Array
    ("/EB_Start", 0, 0, buffer, &bufferLength);

// Use Sekhar's UDP port class
output = Create_UDP_Output (hostname, port);

// This calls sendto()
output->sendPacket(buffer, bufferLength);
```

MAT 240E

stp@mat.ucsb.edu

114

Writing OSC (P5Glove, CNMAT API)

```
int sendP5OSC( struct p5glove_data *info, void *htmsocket) {
    OSCTimeTag tag;
    OSCbuf buf;
    OSC_initBuffer(&buf, 1024, bufferForOSCbuf);
    OSC_openBundle(&buf, tag);
    OSC_writeIntArg(&buf, (info->buttons & P5GLOVE_BUTTON_B));
    // ...
    OSC_writeIntArg(&buf, z);
    OSC_closeBundle(&buf);
    SendHTMSocket(htmsocket, OSC_packetSize(&buf),
                  OSC_getPacket(&buf));
    return 0;
}
```

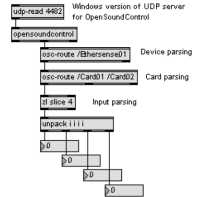
MAT 240E

stp@mat.ucsb.edu

115

Reading OSC (CSL, CNMAT)

```
while (morePackets) {
    buf = OSCPacketBufferGetBuffer(pb);
    n = recvfrom(sockfd, buf, capacity, 0,
                 &(ra->cl_addr), &(ra->clilen));
    if (n > 0) {
        OSCAcceptPacket(pb);
    } else {
        OSCFreePacket(pb);
        morePackets = 0;
    }
}
```



MAT 240E

stp@mat.ucsb.edu

116

Using LibLo to Write OSC

```
#include "lo/lo.h"
lo_address t = lo_address_new(NULL, "7770");

lo_send(t, "/a/b/c/d", "sfsff",
        "one",
        0.12345678f,
        "three",
        -0.00000023001f,
        1.0);

lo_send(t, "/jamin/scene", "i", 2);
```

MAT 240E

stp@mat.ucsb.edu

117

Reading OSC with LibLo

```
lo_server_thread sSrvThrd; // the liblo thread
                        // the callback
int msg_handler(const char *path, const char *types,
                lo_arg **argv, int argc, void *data,
                void *user_data)
{ .. }

                        // create thread
sSrvThrd = lo_server_thread_new(thePort, osc_error);
                        // register CB fcn
lo_server_thread_add_method(sSrvThrd, "/msg", "",
                             msg_handler, NULL);

                        // start srvr
lo_server_thread_start(sSrvThrd);
```

MAT 240E

stp@mat.ucsb.edu

118

Using libOSC++ on UDP

```
OSCPacker packer;
InetUDPMaster udpMaster;

packer.reset();
packer.packString("/osc/test"); // address
packer.packString(",s");        // typetag
packer.packString("hello");     // argument

udpMaster.setDestination("localhost", 10001);
udpMaster.transmit(packer.getData());
```

MAT 240E

stp@mat.ucsb.edu

119

Using NetUtil in Java

```
buf = ByteBuffer.allocateDirect(1024);
addr = new InetSocketAddress("127.0.0.1", 57110);
dch = DatagramChannel.open();
dch.configureBlocking(true);
new OSCMessage("/s_new", new Object[]
{ "default", new Integer(1001),
  new Integer(1), new Integer(0),
  "out", new Integer(0),
  "freq", new Float(0),
  "amp", new Float(0.1f) }).encode(buf);
buf.flip();
dch.send(buf, addr);
```

MAT 240E

stp@mat.ucsb.edu

120

Using OSCPack: Output

```
UdpTransmitSocket transmitSocket(IpEndpointName
                                (ADDRESS, PORT));
char buffer[OUTPUT_BUFFER_SIZE];
osc::OutboundPacketStream pstr(buffer, OUTPUT_BUFFER_SIZE);

pstr << osc::BeginBundleImmediate
  << osc::BeginMessage("/test1")
  << true << 23 << (float)3.1415 << "hello"
  << osc::EndMessage
  << osc::BeginMessage("/test2")
  << true << 24 << (float) 10.8 << "world"
  << osc::EndMessage
  << osc::EndBundle;
transmitSocket.Send(p.Data(), p.Size());
```

MAT 240E

stp@mat.ucsb.edu

121

OSC Input with OSCPack

```
class ExamplePacketListener : public osc::OscPacketListener {
void ProcessMessage( const osc::ReceivedMessage& m,
                    const IpEndpointName& remoteEndpoint ) {
    if( strcmp( m.AddressPattern(), "/test2" ) == 0 ) {
        osc::ReceivedMessage::iterator arg = m.ArgumentsBegin();
        bool a1 = (arg++)->AsBool();
        int a2 = (arg++)->AsInt32();
        float a3 = (arg++)->AsFloat();
        const char *a4 = (arg++)->AsString();
    }
}

in main()
    ExamplePacketListener listener;
    UdpListeningReceiveSocket s (IpEndpointName(ADDRESS, PORT),
                                &listener );

    s.RunUntilSigInt();
```

MAT 240E

stp@mat.ucsb.edu

122

How to Choose?

- Let's write some OSC tests & benchmarks!
 - Basic functionality testing
 - Formats, versions, types, discovery, ...
 - Reading/writing large messages
 - Many args, mixed types, composed types, BLOBs
 - Reading/writing large bundles
 - Scheduler queue size, timing accuracy
 - Serving large address spaces
 - Namespace parsing optimizations
- Compare "standard" C/C++ packages
 - LibLo, OSCPack, LibOSC++, WOsclib (at least)

MAT 240E

stp@mat.ucsb.edu

123

Programming for OSC

- Simple generator/transmitter
 - Open known port to server in SC, CSL, or Max
 - Send messages based on GUI interaction or sensor input (GestureSensors, CUI)
 - OSC for IAC
- Writing servers
 - Design address space
 - Command parser
 - Calling synth functions
 - Handling control updates

MAT 240E

stp@mat.ucsb.edu

124

Advanced OSC

- OSC queries and discovery
 - Dump address space to client or router
- OSC routing and server migration
 - Central node routes OSC commands to servers
- Advanced OSC server (see CORBA ORBs)
 - Faster name look-up (1-step hashing)
 - Faster argument parsing (marshalling)
- See Proc. 2004 OSC workshop
- See OSC wiki <http://www.opensoundcontrol.org>

MAT 240E

stp@mat.ucsb.edu

125

Review: OSC

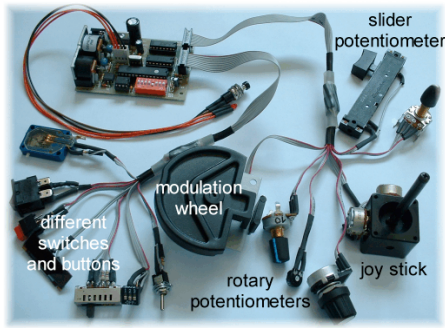
- OSC design, OSC protocol
- Address Spaces and Command Formats
- Writing OSC clients & servers
- OSC APIs in many flavors and languages
- OSC test/benchmark project
- Applying CORBA ORB technology to an OSC server API

MAT 240E

stp@mat.ucsb.edu

126

Topic: Gesture Sensors



MAT 240E

stp@mat.ucsb.edu

127

Sensor Input Drivers

- Other sensors
 - Serial port inputs: loop to parse messages send out some other protocol (see FlockOfBirds examples)
 - USB devices: same (see P5-to-OSC interface)
 - Camera input: frame-grabbers and video feature extraction



MAT 240E

stp@mat.ucsb.edu

128

CSL GestureSensors

```
class GestureSensor { // Abstract CSL/OSC with gesture sensor
public:
    void * mData;      // my data array (typically a float *)
    char * mCmd;       // my OSC command name (without the '/')
    char * mTypeString; // my OSC type string, e.g., "ffff"
    // OSC_CALL_BACK * mReaderFcn; // my OSC call-back function
    // (NB: not a class method)

    // Constructor
    GestureSensor(char * name, char * types, OSC_CALL_BACK * callback);
    // get my data

    // Subclasses add accessors like getX/Y/Z/Roll/Fingers...
    virtual void * get_data() { return mData; };
    // parse an OSC msg. into my array according to my type string
    virtual status parse_OSC_packet(char * typeString, void * args);
};
```

MAT 240E

stp@mat.ucsb.edu

129

Reading from a GestureSensor

```
// This is forked as a loop to update global data from the Ebeam input
void map_ebeam(EBeam & ebeam) {
    float x = ebeam.get_x();
    float y = ebeam.get_y();
    if ((x == 0.0) || (y == 0.0)) return;
    int xQuadrant = x > 1800 ? 0 : 1; // get the current quadrant
    int yQuadrant = y > 850 ? 0 : 1;
    float xVal = x / 1800 - xQuadrant; // and position in quadrant
    float yVal = y / 850 - yQuadrant; // now in the range 0-1
    int index = xQuadrant * 2 + yQuadrant;
    gBaseFreq[index] = (xVal + 1) * gSBaseFreq[index]; // store
    gFreqRange[index] = (yVal + 1) * gBaseFreq[index];
    // printf("EB: %5.1f @ %5.1f\n", x, y);
}
```

MAT 240E

stp@mat.ucsb.edu

130

Other Sensor Examples

- DanO's Matrix
 - Delivers vectors of 144 data values
- Flock of Birds
 - 6 DOF magnetic sensors
- Data Gloves (5DT, P5, etc.)
 - Position, orientation, finger flexion, switches
- CUI-based systems
- All would profit from a configurable cacheing, throttling, filtering interface

MAT 240E

stp@mat.ucsb.edu

131

Clients for Controllers

- Data mapping
 - Range scaling and offset
 - Smoothing, thresholding
 - Rate throttling, cacheing
 - Dynamic range adjustment
- Gesture recognition
- See CSL instrument classes
- Writing OSC servers
 - See GloveMapper

MAT 240E

stp@mat.ucsb.edu

132

Gesture Recognition

- A big issue in the literature
 - ICMC, NIME, CHI
- Low level
 - Limit detection and points of inflection
 - Timing information
 - Spatial feature extraction
- Matching to a gesture vocabulary
 - Reduced features
 - Use neural nets or expert rules



MAT 240E

stp@mat.ucsb.edu

133

Topic: USB

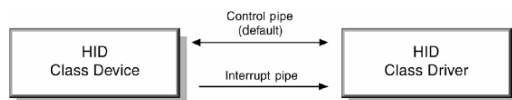


MAT 240E

stp@mat.ucsb.edu

134

USB Human Interface Device (HID)



- Packet format for HIDs
- Refined in the physical interface device (PID) spec. for interfaces with feedback (force joysticks, steering wheels, etc.)
- Mouse, keyboard, gaming controller, etc.
- See P5-to-OSC driver example

MAT 240E

stp@mat.ucsb.edu

135

USB Devices

- Mice, keyboards
- Joysticks, game controllers
- Non-HID
 - Audio IO
 - MIDI IO
 - Camera
 - Storage device
 - Special interfaces



MAT 240E

136

USB Protocol

- USB Report Descriptors
 - controls—information about the state of the device such as on/off or enable/disable
 - data—all other information that passes between the device and the host
 - collections—groups of related controls and data
- USB device hierarchy modeled as “pages” in a document with section numbers; e.g., a generic Desktop usage page (page number 0x01), usage number 0x05 is a game pad

MAT 240E

stp@mat.ucsb.edu

137

The USB Model

- Distinct components of an HID device such as an x axis, y axis, dial, or slider, are called elements
- Information about the elements of a HID class device are grouped into arrays of nested dictionaries
- Each element dictionary contains a 32-bit element cookie, the usage page and usage number, the collection type, and other information (such as the element’s minimum and maximum), and whether or not the element has a preferred state

MAT 240E

stp@mat.ucsb.edu

138

P5 USB Data Structures

- Raw

```
struct p5glove {
    unsigned char data[24], later[24]; // 24 bytes raw data
    char name[128];
    struct usb_dev_handle *usb;
    long long nextsamp;
};
```

- Parsed

```
struct p5glove_data {
    int buttons; // Button bitmask
    int finger[5]; // Finger flex values (0-63)
    struct p5glove_ir {
        int visible; // Was the sensor visible?
        int x,y,z; // position (-511 - 511)
    } ir[8]; // 8 IR Sensor values
};
```

MAT 240E

stp@mat.ucsb.edu

145

Reading & Parsing HID Data

```
// This is called in an endless loop from main()
int p5glove_sample(p5glove p5, struct p5glove_data *info) {
    int err;
    long long now;
    struct timeval tv;
    gettimeofday(&tv, NULL); // see if it's time yet
    now = tv.tv_sec*1000000 + tv.tv_usec;
    if (p5->nextsamp != 0 && now < p5->nextsamp) {
        return -1;
    }
    p5->nextsamp = now + (1000000/30); // set 30Hz refresh rate
    err = usb_bulk_read(p5->usb, 0x81, p5->data, 24, 2000); // READ
    if (err == 24) {
        process_sample(p5, info); // PARSE
        return 0;
    }
}
```

MAT 240E

stp@mat.ucsb.edu

146

USB HID Code Examples

- Windows

- <http://www.lvr.com/hidpage.htm>
- <http://www.usb.org/developers/hidpage/microhid>

- Mac OS X

- <http://developer.apple.com/documentation/DeviceDrivers/Conceptual/HID/index.html>

- Linux

- <http://www.frogmouth.net/hid-doco/linux-hid.html>

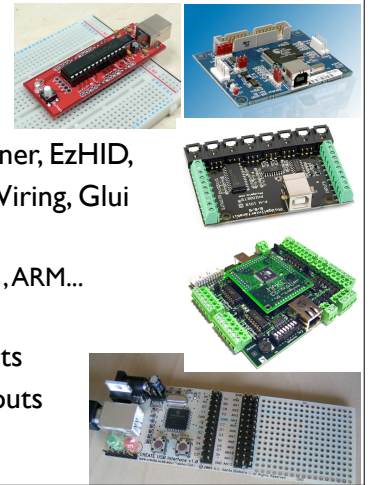
MAT 240E

stp@mat.ucsb.edu

147

USB Interfaces

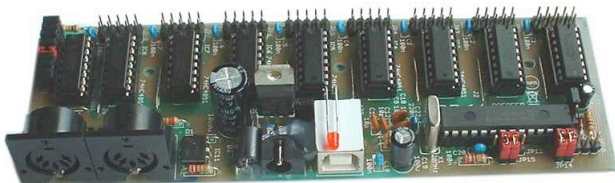
- CUI, Arduino, Gainer, EzHID, Makelt, Phidgets, Wiring, Glui
- Microcontroller
 - PIC, ATmega, 8051, ARM...
- USB interface
- Analog/digital inputs
- Analog/digital outputs
- Programmability



MAT 240E

148

All-in-One: Doepfer USB64



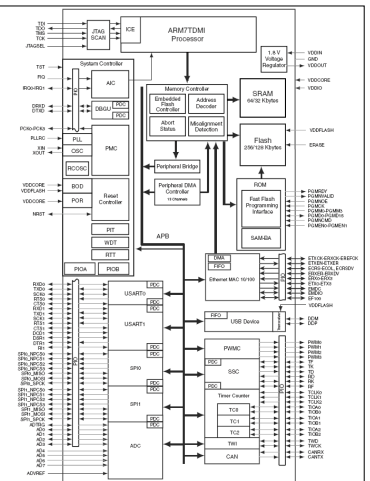
MAT 240E

stp@mat.ucsb.edu

149

SAM7X

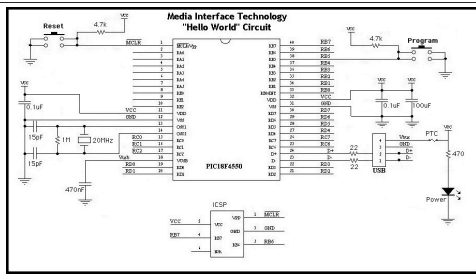
- Processor
- RAM/ROM
- Controller
- UARTs
- Timer(s)
- USB
- Ethernet



MAT 240E

150

CUI



- 13 analog inputs
- 18 digital inputs
- USB HID
- Easy configurability

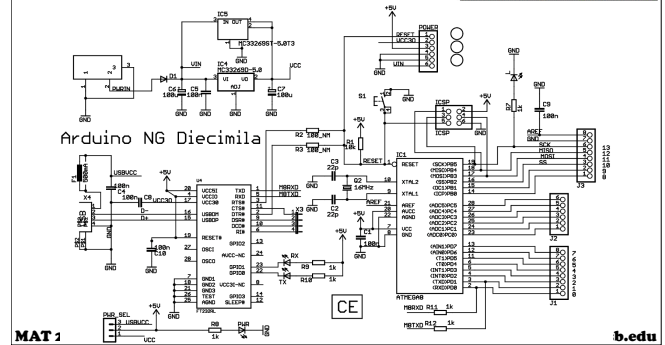


MAT 240E

stp@mat.ucsb.edu

151

Arduino & its Applications



MAT 1

b.edu

152

Arduino Programming in Wiring

```

/* AnalogInput:
 * Turns on and off a light emitting diode(LED) connected to digital
 * pin 13. The amount of time the LED will be on and off depends on
 * the value obtained by analogRead(). In the easiest case we connect
 * a potentiometer to analog pin 2.
 */
int potPin = 2;    // select the input pin for the potentiometer
int ledPin = 13;   // select the pin for the LED
int val = 0;       // variable to store the value coming from the sensor

void setup() {
  pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT
}

void loop() {
  val = analogRead(potPin); // read the value from the sensor
  digitalWrite(ledPin, HIGH); // turn the ledPin on
  delay(val);                // stop the program for some time
  digitalWrite(ledPin, LOW); // turn the ledPin off
  delay(val);                // stop the program for some time
}

```

MAT 240E

stp@mat.ucsb.edu

153

Comparison (by Wiring)

	Wiring board	BasicX	BasicStamp	PIC 16F876	atmega8 / atmega168 (Arduino)
I/O Pins	53	16	15	22	11
Memory	128K	32K	2K	14K	8K / 16K
Analog Inputs	8	8	No	No	6
External Interrupts	8	0	0	1	2
Hardware Serial Ports	2 hardware serial ports, One (Serial1) available on Wiring pins 2(Rx) and 3 (Tx) the other (Serial) is available through the USB connector on the board	1	1	1	1 Available through the USB port on the board
USB	Yes, Included on the Wiring board	No, it requires an extra adapter	No, it requires an extra adapter	No, it requires an extra adapter	Yes, Included on the board
Power	External 7-12V generic adapter or through the USB when connected to a computer	Requires power regulator circuit + adapter	Requires power regulator circuit + adapter	Requires power regulator circuit + adapter	External 7-12V generic adapter or through the USB when connected to a computer
PWM (Analog Outputs)	6 PWM Channels (analog outputs)	No	No	2	3
Programming language	Wiring	Basic	Basic	Basic	Wiring
OS Supported	Windows, MacOSX, Linux	Windows, MacOSX	Windows	Windows	Windows, MacOSX, Linux

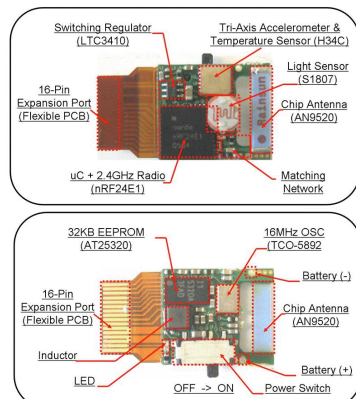
MAT 240E

stp@mat.ucsb.edu

154

UCI Eco Mote

- Tiny
- Wireless
- A/D IOs
- Built-in sensors
 - Accelerometer
 - Light
 - Temperature
- Python API



MAT 240E

155

Tiny Wireless Interfaces

	uPart	TinyNode	ZN1	Spec	Eco
Developed by	U. of Karlsruhe	EPFL & Shockfish	Hitachi	UCB	UCI
Size (mm)	<10x10x10	30x40x??	15x15x4	2x2.5x??	13x10x8
Battery	CR1620	Li-Poly 30mAh	Li-Poly 30mAh	Li-Poly 30mAh	Li-Poly 30mAh
uC	PIC12F657 4MHz	MSP430F1611 8MHz	H8S2218 4-24MHz	AVR-Like RISC, 4MHz	8051 16MHz
RF Transceiver	TX Only, 310 ~ 930MHz Max. 40Kbps -12 ~ 10dBm	XE1205 433, 868, 915MHz 2.4GHz 250Kbps -25 ~ 0dBm	CC2520 2.4GHz 250Kbps -25 ~ 0dBm	CC1000 315/433/868/915 1Mbps -20 ~ 0dBm	nRF2401 2.4G 1Mbps -20 ~ 0dBm
Sensing Devices	Light, Temp, 2D Acc	Separate sensor board	Separate sensor board	X	Light, Temp, 3D Acc.
Memory	14K F, 128B E 64B SRAM	48KB Flash 10KB RAM	128KB Rom 12KB RAM	X	512B R, 32K E 256B + 4KB RAM
Expandability	No	YES, Stackable	YES, Stackable	NO	YES, Flex PCB
TinyOS	No	YES	NO	YES	No
Power	Tx: 17mA @ 9dBm Rx: 15.5mA Standby: 0.6uA	Tx: 25mA Rx: 25.5mA Standby: 5.1uA	Tx: 24.9mA Rx: 25.5mA Standby: 1uA	Tx: 24.9mA Rx: 25.5mA Standby: 1uA	Tx: <10mA Rx: 21mA Standby: 6uA

MAT 240E

stp@mat.ucsb.edu

156

Other Sensors

- Wii Remote (WiiMote)
- Custom gloves
- Custom joysticks
- Graphics “knob boxes”
- Extended gaming controllers
- Other media (video, sensing)



MAT 240E

157

Review

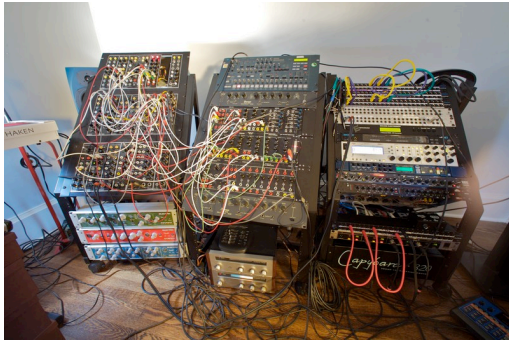
- Custom Gesture Controllers
- USB HID Spec
- HID Programming
- Integration with OSC/MIDI
- USB Interfaces

MAT 240E

stp@mat.ucsb.edu

158

Topic: Audio Streaming



MAT 240E

stp@mat.ucsb.edu

159

Audio Streaming

- Audio streaming protocols and APIs
- Networking Background
- Simple Sample Streaming: RFS
- SDIF as a Streaming Protocol
- RTP and RTCP

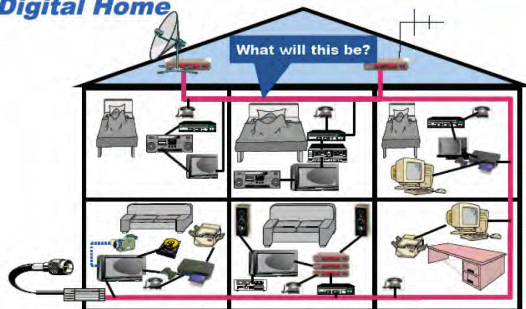
MAT 240E

stp@mat.ucsb.edu

160

State-of-the-art 2008 = ?

Interactive Multimedia Network for the Digital Home



MAT 240E

stp@mat.ucsb.edu

161

Media Streaming Standards



- Yamaha mLAN (FireWire)
 - >150 channels at 24/48 over FW400
- MADI (AES10-2003)
 - 64 channels of 24/96 over coax or fiber
- CobraNet
 - 64 * 2 channels of 20/48 over 100BaseT
- Aviom A-Net
- MediaNet (1000BaseT)

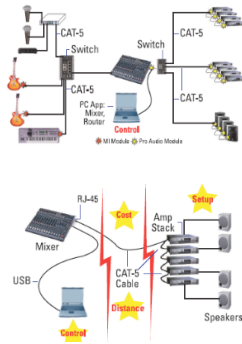
MAT 240E

stp@mat.ucsb.edu

162

E.g., MediaNet (1000BaseT)

- 512 X 512 audio channels of 24/48
- 256 X 256 audio channels of 24/96
- Uncompressed video or compressed HD
- Control data channel up to 100 Mbps
- Latency <200 usec over a 140 meter cable
- Power over Ethernet (PoE)



MAT 240E

stp@mat.ucsb.edu

163

Audio Streaming

- Requirement: send data packets between a source and a sink (at a fixed rate & latency)
- Optional: I-many links, multicasting
- Design: probably use UDP rather than TCP
- Add some checking
 - Error detection and error recovery
 - Packet sequencing
 - Packet checksums

MAT 240E

stp@mat.ucsb.edu

164

Network Programming

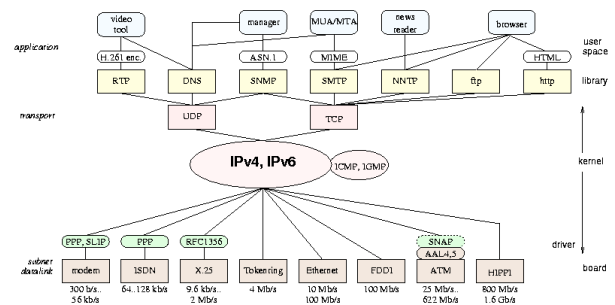
- Hosts/machines: 4-byte IP addresses
- Sockets/ports: 2-byte socket numbers
- Socket I/O
 - Bind(), listen() -- telephone analogy
 - Akin to File I/O
 - send()/recv()
 - Use select() for multi-socket input
- Packet Content Interpretation (how a protocol is implemented)

MAT 240E

stp@mat.ucsb.edu

165

IP Protocols



MAT 240E

stp@mat.ucsb.edu

166

Media Streaming (from Wikipedia)

Streaming media technologies

- Adobe Flash
- Accordent Technologies
- Ampache
- Clipstream
- FORScene
- FreeCast
- Icecast
- Matroska
- Microsoft Windows Media
- Ogg/Vorbis
- Orb
- Philips Media Manager
- QuickTime
- ReelTime.com
- RealNetworks
- RealPlayer
- SHOUTcast
- Slingbox
- SlimServer
- Winamp
- Unreal Media Server

Stream and transport protocols

- HTTP
- MMS
- RTMP
- RTP
- RTCP
- RTSP
- RealNetworks RD1
- UDP

MAT 240E

stp@mat.ucsb.edu

167

CSL RemoteFrameStream Protocol

- CSL RFS: Example of a simple I-to-I audio streaming protocol
- Audio packets have headers like sound files
- Use sequence numbers for detecting dropped packets
- Sender looks like a CSL IO
- Receiver looks like a CSL UnitGenerator

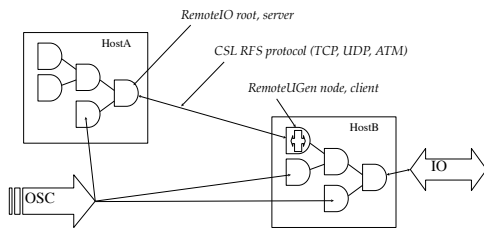
MAT 240E

stp@mat.ucsb.edu

168

Distributed CSL

RemoteUGen and RemoteIO



MAT 240E

stp@mat.ucsb.edu

169

CSL RFS Packet Header

```
// CSL RS protocol message header structure (8 bytes)
typedef struct {
    unsigned magic;           // Magic number (=0x004200XX,
                              // whereby XX is the command)
    unsigned short frames;    // Number of frames requested
    unsigned short channels;  // Number of channels requested
} CSL_RS_MSG;

// Remote commands
// Set up the server's response socket
#define CSL_CMD_SET_CLIENT 42
// Request a buffer of samples from the server
#define CSL_CMD_NEXT_BUFFER 43
// Stop the server
#define CSL_CMD_STOP 47
```

MAT 240E

stp@mat.ucsb.edu

170

RFS Classes in CSL 4.1

RemoteStream - is-a UnitGenerator

```
// Protected low-level setup methods
int initSockets(char * name, unsigned short port);
void initPacket();
virtual int connectToServer();
// Work method, uses ring-buffer and reader thread
void nextBuffer(Buffer &outputBuffer)
```

RemoteIO - is-a IO

```
// This is forked to read & process output requests
extern "C" void * RemoteIO_read_loop(void * inst);
void process_request_packet();
// Work methods use timing and socket threads
open(), close(), start(), stop()
```

MAT 240E

stp@mat.ucsb.edu

171

Updating CSL RFS

- Test it
- Scale it
- Write a simple benchmark
- Port to RTP or other standard protocol
- What's the overhead?

MAT 240E

stp@mat.ucsb.edu

172

RTP & RTCP

IP Header	UDP Header	RTP Header	RTP Payload
-----------	------------	------------	-------------

Real-time Transfer Protocol (RTP)

- RFC 1889
- Support transfer of "data that has real-time properties," i.e., multi-party media streaming
- Semi-reliable

RTP Control Protocol (RTCP)

- RTP control (changes of participants in an on-going session)
- QoS monitoring
- QoS negotiation

MAT 240E

stp@mat.ucsb.edu

173

RTP/RTCP Concepts

- Session state
- Synchronized/contributing sources
- End system (data sink)
- Member, profile
- Payload, packet (RTP & RTCP)
- Mixer, translator
- RTP packet header
- RTCP sender reports

MAT 240E

stp@mat.ucsb.edu

174

RTP APIs

- jRTPLib
 - C++ Objects for Session, Packet, Source, Time, Report, Params, TransmissionInfo, etc.
 - Complex
 - Used in CSL by Ryan Avery
- LibRTP
 - Minimal C API
 - Dormant?

MAT 240E

stp@mat.ucsb.edu

175

A Taste of jRTPLib

```

RTPSession sess;
RTPUDPV4TransmissionParams transparams;
RTPSessionParams sessparams;
sessparams.SetOwnTimestampUnit(1.0/10.0);
sessparams.SetAcceptOwnPackets(true);
transparams.SetPortbase(portbase);
RTPIPv4Address addr(destip, destport);
status = sess.Create(sessparams, &transparams);
checkerror(status);
sess.AddDestination(addr);
// send a packet
sess.SendPacket((void *)"1234567890", 10, 0, false, 10);
// reader loop
RTPPacket *pack;
while ((pack = sess.GetNextPacket()) != NULL) { ... }

```

MAT 240E

stp@mat.ucsb.edu

176

RTCP

- Gather sender/receiver reports
- Get source description items (like unique name)
- Set-up/terminate connections
- Application-specific extended packet format

MAT 240E

stp@mat.ucsb.edu

177

Other Streaming Protocols

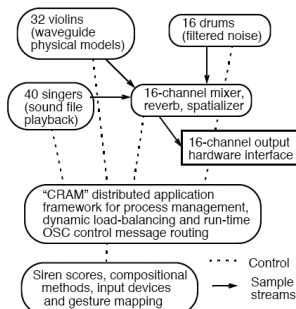
- See list above
 - MAGIC, etc. on cat5
 - MADI on coax/optical
 - NAS Audio/Ethernet
 - FireWire, mLAN
- Streaming SDIF?
- Integrating with common services?

MAT 240E

stp@mat.ucsb.edu

178

Topic: Distributed Processing



MAT 240E

stp@mat.ucsb.edu

179

Distributed Processing Environments (DPEs)

- The goal (previous slide)
- Background: CORBA and TINA/C
- DPE Requirements
- CRAM as a lightweight DPE

MAT 240E

stp@mat.ucsb.edu

180

DPE Background

- Cluster Mgmt. in TelCos
- Requirements
 - Frameworks to deploy, start/stop, and monitor multi-host distributed real-time OO applications
 - Both fault-tolerance and load-balancing
 - Robustness, simplicity, and low overhead
 - CORBA-like services and scalability/replication

MAT 240E

stp@mat.ucsb.edu

181

CRAM

- CRAM: Yet another Distributed Processing Environment
- CRAM is 3rd-gen. DPE implementation at CREATE (1996-2004) (HPDM/TAO, Yellow/CORBA_AV)
- Support for mixed C++, SC, Smalltalk components in a network

MAT 240E

stp@mat.ucsb.edu

182

CRAM DPE Operation Model

- Applications are composed of services
 - which probably talk among themselves apart from the management system
- Any program can be a service (lightweight wrapper)
- Nodes (HW) can run services
 - and monitor their hardware and services
- The system manager talks to a node and its services
 - Start/stop, load-balance, monitoring, fault-detection/restart, etc.

MAT 240E

stp@mat.ucsb.edu

183

Components of CRAM

- Node manager program (Node)
 - Daemon that provides "remote exec" functionality and status/performance monitoring; runs on all machines
- Service interface (Service)
 - Wrapper code that is added to all application server programs managed by the DPE
 - Adds socket listener threads for mgr messages
 - May add init/restart functions between server obj. & app.
 - May integrate logging, status monitoring into app. code
- System manager (Manager)
 - Talks to nodes to administer services for distr. apps
 - Uses DBs for network, services, and app. configurations
 - May offer an expert/constrained configuration planner

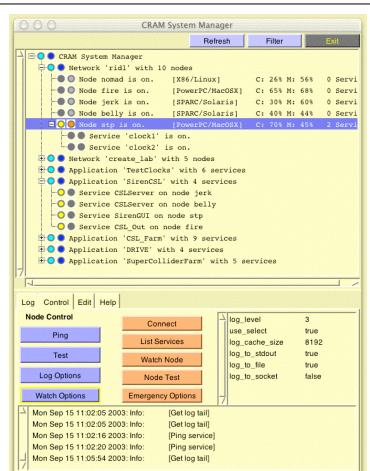
MAT 240E

stp@mat.ucsb.edu

184

CRAM Manager

- Network/Node
- Node/Service
- Application/Service
- Log/Control pane
 - Run-time monitor
 - Planning
 - DB play-back



MAT 240E

stp@mat.ucsb.edu

185

Using CRAM

- Write a Service class for the app
- It handles main() (creates service object) and start/stop messages (start/stop the base app)
- (Optional) add run-time option-setting, logging, app. status monitoring (handle CRAM msgs)
- Create a service type record in the DB
- Add applications to the DB
- Go!

MAT 240E

stp@mat.ucsb.edu

186

CRAM Databases (Simple RDBMS)

- Networks
 - Nodes and their properties
 - HW, OS, MIPS/MFLOPS, LAN, special I/O
- ServiceTypes
 - Name, arguments, options
 - HW/OS/IO requirements (for configuration)
- Services
 - Type and actual run-time arguments (net IO)
- Applications & History
 - Lists of services (& their options) on nodes

MAT 240E

stp@mat.ucsb.edu

187

CRAM Implementation Details

- Node/Service source = 4kLOC C++, 2.5 kLOC Java, 0.8 kLOC ST80 (w. DB-IO)
- Manager & DB source = 4kLOC Smalltalk [~50% auto-gen]
- Uses low-level UDP/TCP protocols
- Several levels of failure recovery, node-service-discovery, heart-beat monitors
- Logging and monitoring to DB, replay

MAT 240E

stp@mat.ucsb.edu

188

CRAM vs Cluster Managers

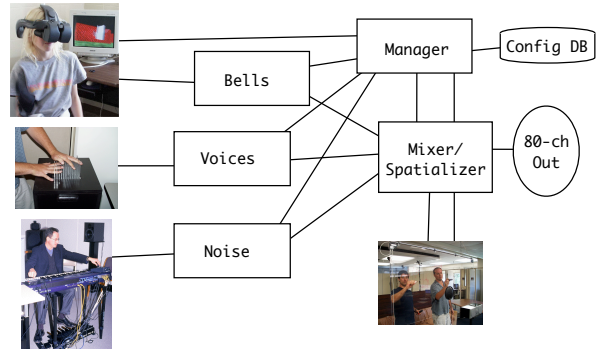
- Assume dynamic application
- Assume heterogeneous application
- Offer real-time fault recovery (wrt page pick-up time)
- Application configuration and planning (to come)
- Lightweight infrastructure (wrt MPI, TINA/C DPE, TAO, etc.)

MAT 240E

stp@mat.ucsb.edu

189

CRAM Configuration for CNSI



MAT 240E

stp@mat.ucsb.edu

190

CRAM Coding

- CSL + RFS/RTP + CRAM
- OSC with SC servers
- GestureSensor servers
- CRAM tools and DBs
- CRAM for the AlloSphere

MAT 240E

stp@mat.ucsb.edu

191

MAT 240E Review

- Control vs. content
- MIDI protocol
- Open Sound Control
- Other control interfaces
- Networking and streaming
- Distributed processing

MAT 240E

stp@mat.ucsb.edu

192

Where do we go from here?

- Applications!
- Code from MAT 240D: soft synths
- Sensors and control: CUI projects
- Performances, installations
- AlloSphere applications

MAT 240E

stp@mat.ucsb.edu

193

Projects

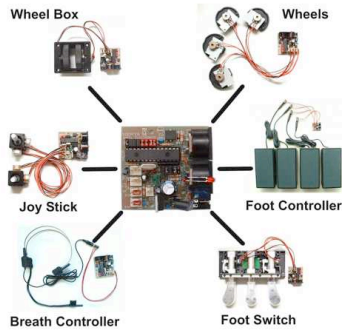
- General Score II player
- Control recorder editor
- OSC benchmarks
- RTP (or SDIF) streaming
- CRAM for CNSI

MAT 240E

stp@mat.ucsb.edu

194

MAT 240E



MAT 240E

stp@mat.ucsb.edu

195

MAT 240F - Digital Audio Programming: Audio Analysis and Music Information Retrieval (Spring, 2008)

The MAT 240 course sequence is a six-part (two-year) practical programming course; it consists of hands-on software development devoted to digital audio and multimedia applications. Students read a selection of papers from the literature, with the emphasis on learning to use and extend the current state-of-the-art programming methods, tools, and programming interfaces. Class assignments involve C/C++/Java programming on Linux, Macintosh, MS-Windows, various plug-in APIs, and other platforms.

The focus of the MAT 240F course is on audio analysis and signal processing techniques applied to sound/music databases and music information retrieval systems. We will work with libraries for signal analysis and feature extraction to develop skills in time-domain processes such as beat following, tempo analysis, and song segmentation, and in spectral-domain analysis techniques such as pitch estimation, spectral peak analysis and tracking, and instrument signature identification. The topic of feature vector design will play an important role in the development tasks. Applications will include music segmentation, finger-printing, thumb-nailing, clustering and genre classification, and user preference matching.

Students are expected to know the basics of digital audio signal representation and processing, and to be proficient in C, C++, or Java (Smalltalk, SuperCollider or LISP are a plus). Grading will be on the basis of in-class participation and programming projects.

Course Outline

- Multimedia Database Applications
- Signal Processing for Feature Extraction
- Time-domain, Frequency-domain Analysis
- Other Kinds, Cross-domain analysis
- Audio Segmentation and Musical Form
- Clustering and Classification
- Handling of Large or Dynamic Feature Vectors
- Application Requirements and Design

Instructor

- Stephen T. Pope (stephen@mat.ucsb.edu)

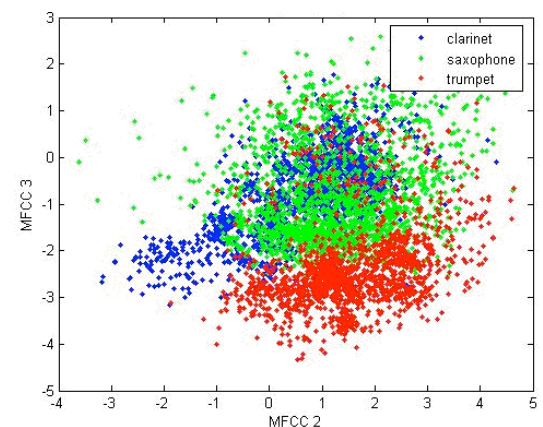
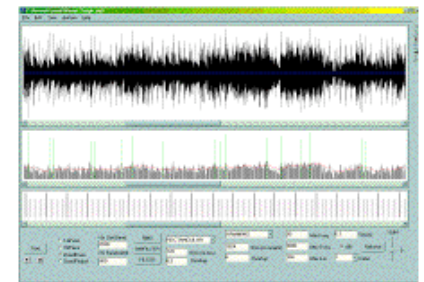
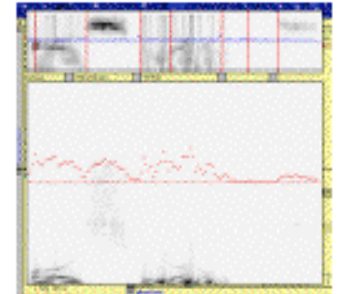
Meeting time and place

- Tues/Thurs 2:00 - 3:50 PM, Music 2215

Electronic Resources

- Course Web Site: <http://www.create.ucsb.edu/240>
- Email Mailing List

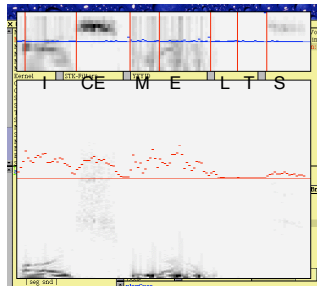
See <http://www.mat.ucsb.edu/mailman/listinfo/240> to join



MAT 240F: Digital Audio Programming: Audio Analysis and Music Information Retrieval

Stephen T. Pope
stephen@mat.ucsb.edu
Spring, 2008

with materials from
G. Tzanetakis, J. O.
Smith, X. Serra, R.
Dannenberg, et al.

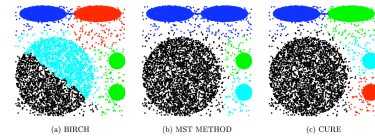


Steven Smallick, Speech Segments, S. T. Pope, 2001

1

240F: Analysis & MIR Topics

- 1: Music Metadata
- 2: Signal Analysis
- 3: Numerical Processing
- 4: Databases & Applications



2

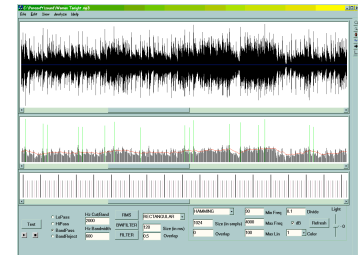
MAT 240F Overview

- Multimedia Database Theory, Design and Applications
- Signal Processing for Feature Extraction
 - Time-, Frequency-domain Analysis
 - Other domains, Cross-domain Analysis
- Numerical & Higher-level techniques
 - Derived Features and Segmentation
- Application Requirements and Feature Vector Design

3

Lecture 1 Outline

- MAT 240 course series: digital audio programming
- MAT 240F Goals
- Course logistics
- Written materials
- Course overview
- Topic 1
 - Media metadata and databases
 - Application tour



FASTLab Rhythm Browser, Frodo Horn, 2000

4

MAT 240 Series

- Hands-on programming courses using (primarily) C, C++, and Java for digital audio application development
- Six-quarter (two-year) course series
- Students use a variety of software development tools on MS-Windows, Linux/UNIX, and the Macintosh

5

MAT 240 Topics (6 quarters)

- A: File I/O and Streaming Media
- B: Spectral Transformations
- C: Spatial Processing of Sound
- D: Sound Synthesis Techniques
- E: Control and Distributed Processing
- F: Analysis and Feature Extraction**



6

Course Materials

- Presentation slides (handed out)
- Reader at the BookStore: papers from *ISMIR* and *ICMC Proceedings*, *CMJ*, etc.
- MAT 240F web site and links
- 240@mat.ucsb.edu mailing list
- Example code archive
- Available software development tools

Course Logistics

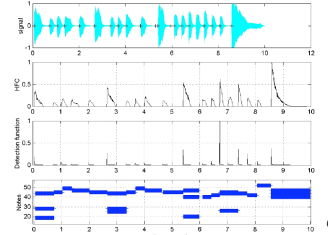
- Lectures
 - Tues/Thurs 2:00 - 4:00 PM
 - CREATE class room (Music 2215)
- Lab, work groups TBD
- Grading
 - Class participation, discussions
 - 2-3 group projects
 - Final group presentation and write-up

MAT 240F Topics

- Multimedia Database theory & Design
- Signal Analysis for Feature Extraction
 - Time-domain, Frequency-domain Analysis
 - Other Kinds of Analysis: Wavelets
 - Cross-domain Analysis and Heuristics
- Numerical methods: smoothing and grouping
 - Segmentation and Musical Form
 - Data Clustering and Classification
- Handling of Large or Dynamic Feature Vectors
- MIR/MDB Application Domains

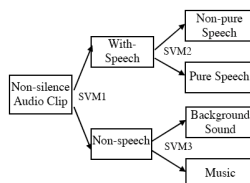
Problem Statement: Applications

- Examples
 - Automatic playlist generation
 - Audio transcription



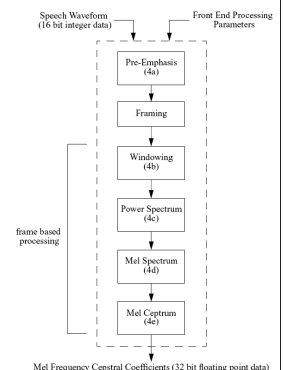
Topic 1: Music Metadata

- Introduction
 - * Kinds of Audio Data and Metadata
 - * Dimensions of Music Info. Retrieval
 - * APIs for MIR Tools
- Multimedia Databases
 - * Feature Vectors and Indexing
 - * Feature Extraction and Signal Analysis
 - * Numerical Processing: Clustering, Classification
- Audio Signal Processing for Feature Extraction
 - * Time Sequences, Windowing
 - * Analysis Domains, Transformations



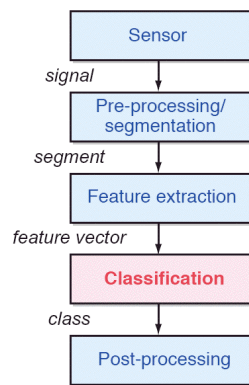
Topic 2: Signal Analysis

- Time-domain Audio Analysis and Applications
 - * Windowed RMS Envelope Extraction
 - * Beat Detection and Tempo Analysis
 - * Time-based signal segmentation
- Frequency-domain Analysis
 - * Pitch Detection Techniques
 - * Spectral Analysis and Interpretation
 - * Spectral Peaks and Tracking
 - * Other Spectral Measures
- Other Kinds of Analysis: Wavelets
- Cross-domain analysis



Topic 3: Numerical Processing

- Data Reduction, Smoothing
- Correlation, Grouping
- Princ./Indep. Component Analysis
- Audio Segmentation and Musical Form
- Clustering and Classification

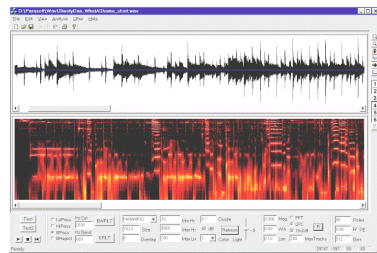


Topic 4: Databases & Applications

- Database Issues
- Handling of Large or Dynamic Feature Vectors
- Application Requirements and Design
 - * Searching, Indexing, and Players
 - * Audio Summarization and Thumb-nailing
 - * Content Matching and Finger-printing
 - * Data Clustering and Genre Classification
 - * Other Applications: Mapping Systems

Course Organization

- MIR/MMDB Intro
- DASP Techniques
- Feature Vector Design
- Application Requirements
- System Organization
- Development Projects
- Tool- and API-building

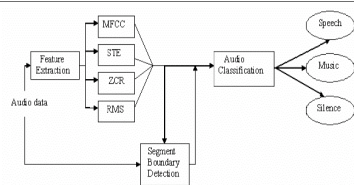


MAT 240F Reader

- Overviews, tutorials
 - Foote, Tzanataakis, Pope et al., Wright
- DASP Techniques
 - Pope et al., Serra, Smith, de la Cuadra, Meng
- Higher-level Techniques
 - Segmentation, clustering, classification
- Applications
 - Tracking/mapping, ID, summarization
- Tools and APIs

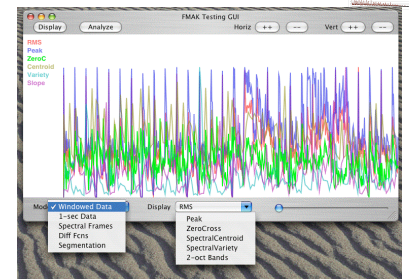
MAT 240F Applications

- One-step Tools
 - Tracker, segmenter, single feature extraction
 - Interactive programs
- Multi-feature tools
 - Finger-print, thumb-nail, etc.
- Heuristic techniques
 - Blackboard, neural nets, SOMaps
- Real-world MIR applications



Basic APIs, Techniques

- Audio I/O
- SoundFile I/O
- Data formatting, windowing
- Scale, limit, offset
- Peak detection, weighting, tracking
- FFT + statistics
- Auto-correlation



Code Organization

- Application Models
 - PortAudio streamers
 - Using libSndFile and binary IO (SDIF)
 - Feature extraction plug-ins
 - GUIs to MIR data
 - Query, Browsing, Editors
 - JUCE, GLV, Cocoa

19

Digital Audio Programming

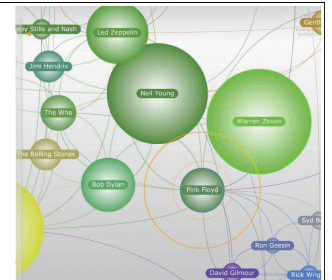
- Digital audio data
 - Sampling and quantization
 - Data types for audio data
- Audio I/O APIs
 - See MAT 240A-E slides, web site, code archive
 - Call-back functions and buffer handling
 - Platform-specific, cross-platform, language-specific
- Synthesis techniques (see MAT 240D)
- Advanced programming techniques (see MAT 201B)
 - Multi-threaded, exception-handling, GUIs,
 - Design patterns, APIs, I/O

20

MIR/MDB Applications

Part 1: Intro to MIR and MMDBs

- Overview of MMDB
- Audio Data and Metadata
- MDB Applications
- Models and Frameworks in C++
- DASP for MDB Feature Extraction



22

In the Reader

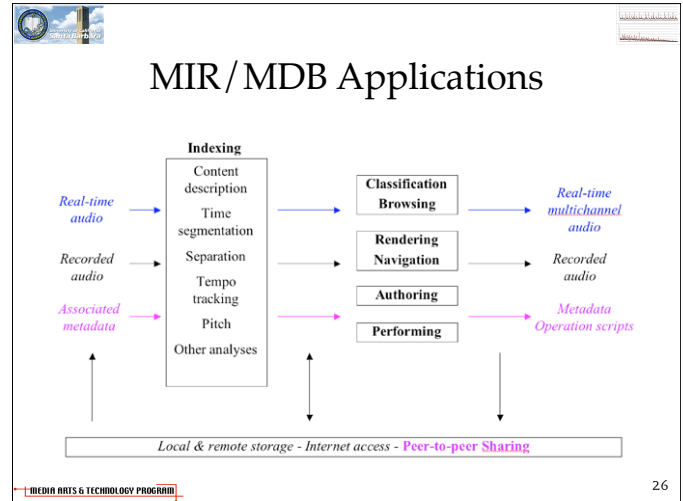
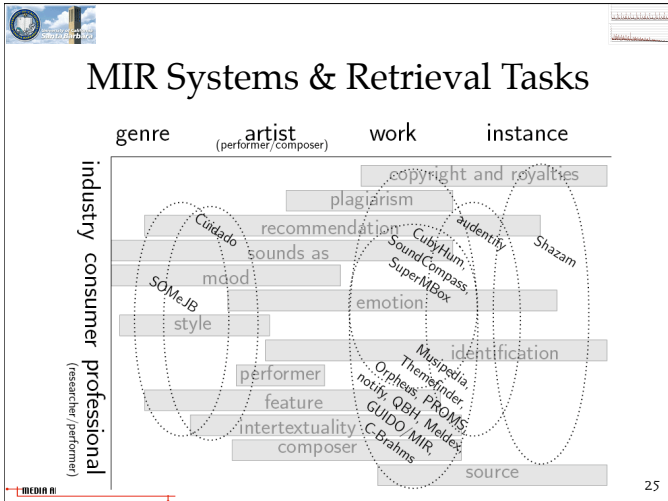
- Foote, Tzanetakis, Pope, Orio, Roy overview/tutorial articles
- Typeke MIR application survey
- Wright analysis-resynthesis comparison
- Tool-API notes
- DBMS background

23

Typeke Appl. Table

Name	Input		Matching					Features					Indexing	Collection Size (Records)
	Audio	Symbolic	Symbolic	Exact	Approximate	Polyphonic	Audio Fingerprints	Pitch	Note Duration	Timbre	Rhythm	Contour	Intervals	Other
audentify!	*	*	*	*	*	*	*	*	*	*	*	*	Inverted files	15,000
C-Brahms	*	*	*	*	*	*	*	*	*	*	*	*	none	278
CubyHum	*	*	*	*	*	*	*	*	*	*	*	*	LET	510
Cuidado	*	*	*	*	*	*	*	*	*	*	*	*	not described	words for 100,000
GUIDO/MIR	*	*	*	*	*	*	*	*	*	*	*	*	Tree of transition matrices	150
Meldex/Greenstone	*	*	*	*	*	*	*	*	*	*	*	*	none	9,354
Musipedia	*	*	*	*	*	*	*	*	*	*	*	*	Vantage objects	> 30,000
notify! Whistle	*	*	*	*	*	*	*	*	*	*	*	*	Inverted files	2,000
Orpheus	*	*	*	*	*	*	*	*	*	*	*	*	Vantage objects	476,000
Probabilistic "Name That Song"	*	*	*	*	*	*	*	*	*	*	*	*	Clustering	100
PROMS	*	*	*	*	*	*	*	*	*	*	*	*	Inverted files	12,000
Cornell's "QBH"	*	*	*	*	*	*	*	*	*	*	*	*	none	183
Shazam	*	*	*	*	*	*	*	*	*	*	*	*	Fingerprints are indexed	> 2.5 million
SOMeJB	*	*	*	*	*	*	*	*	*	*	*	*	Yes	11,132
SoundCompass	*	*	*	*	*	*	*	*	*	*	*	*	Hierarchical Filtering	12,000
Super MBox	*	*	*	*	*	*	*	*	*	*	*	*	none	35,000
Themefinder	*	*	*	*	*	*	*	*	*	*	*	*	none	35,000

24



Code Starters

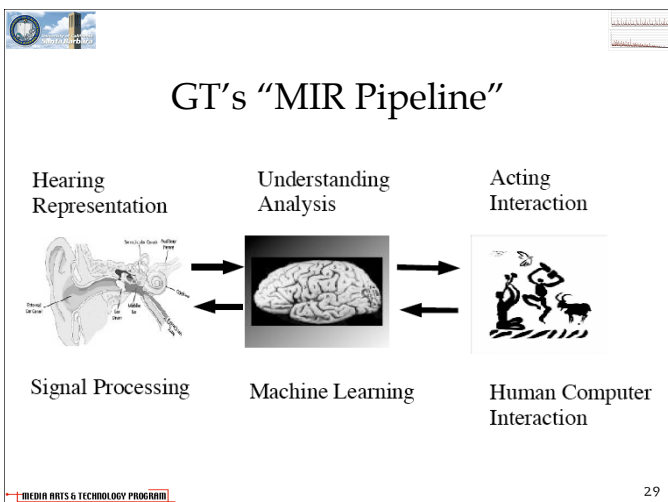
- 💡 PortAudio + windowed processing
 - 💡 RMS env. extraction
 - 💡 Tempo estimation
 - 💡 Beat induction
- 💡 With FFTW
 - 💡 Spectra, peaks, and tracks
- 💡 Pitch trackers
 - 💡 F0 envelope
 - 💡 LPC

27

Audio Data and Metadata

- 💡 Sound / music data
 - 💡 Mixed, sampled, quantized audio
 - 💡 "Carefully recorded" monophonic audio
 - 💡 Speech (clean or noisy)
 - 💡 MIDI tracks (raw or performed)
 - 💡 Common-practice Western notation
 - 💡 MusicXML
- 💡 Related Metadata
 - 💡 ID: title, artist, medium, genre, encoding
 - 💡 Segmentation, summary, derived features

28

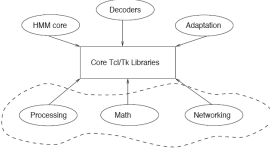


Dimensions of Music Information Retrieval Applications

- 💡 Indexing, query, access
 - 💡 Use content or metadata for query
- 💡 Understanding, transcription
 - 💡 Derive (music/speech) model
- 💡 Clustering, classification
 - 💡 Feature vector for discrimination
- 💡 Content identification, finger-printing
- 💡 Preference-matching, recommendation

30

APIs for MIR Tools

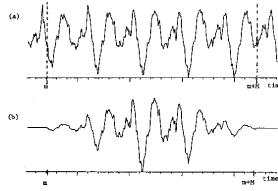


- **Marsyas**: G. Tzanetakis (06), flexible tool set, scripting language, segmentation and classification
- **LibOFA**: Holm/Pope (00), simple FV for unique ID comparing to a large pre-analyzed database
- **FMAK**: FASTLab (04), complex/dynamic FV, segmentation, CURE clustering, CSL-like framework
- **8S**: Pope (94), simple Smalltalk-based speech segmenter and database
- **D2K/M2K**: West/MIREX (06), Java-based GUI related to D2K, many apps.
- **LipTSP**: P. Kabal (00), C routines for DASP & IO

31

APIs - 2

- Libxtract
- Aubio
- SonicVisualizer plug-ins
- Loris
- Marsyas
- SPEAR
- CSL/FMAK
- LibTSP



32

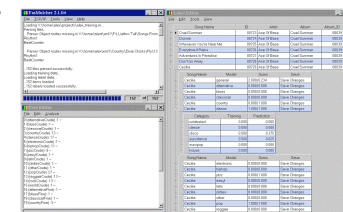
Databases for Music (MDB)

- Uses of databases in music software
 - DB access as application (query/search)
 - Music SW that uses music/audio DB (increasingly common in the future)
- Databases offer
 - Persistency
 - Query capacity
 - Multiple access

33

Multimedia Databases

- Background: relational databases
- How is media data different?
 - Structure / Applications
 - Feature Vectors and Indexing
 - Feature Extraction and Signal Analysis
- Numerical Processing
 - Content segmentation
 - Clustering and Classification



34

Database Technology

- Relational DBMS
 - Fixed table-formatted data
 - Few data types (number, string, date, ...)
 - One or more indices/table (part of DB design, application-specific, impacts performance)
 - Cross-table indexing and joins
 - SQL examples (create, insert, update, select)
- Media data (historically images)
 - Volume (large single items)
 - Format (items no known structure)
 - Content and metadata (required for usage)

35

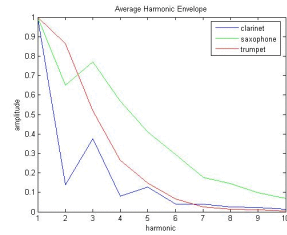
DB & System Design Issues

- Feature vector design
 - Simple/flat vs. complex/hierarchical
 - General purpose vs. application-specific
- Signal analysis (feature extraction) for a given feature vector
- DB storage of content and meta-data
 - Data volume, indexing, rights management, sharing/distribution

36

Feature Vectors and Indexing

- Feature = derived (numerical) parameter
- Feature vector = list of features for a single point in time, or average for an entire selection
- Feature table = list of feature vectors for several time slices (not always used)



— MEDIA ARTS & TECHNOLOGY PROGRAM —

37

Example Features

Features:

- Time-domain, low-level
 - Windowed RMS amplitude
- Time-domain, high-level
 - Tempo, beat structure, segmentation
- Frequency-domain, low-level
 - Pitch, spectrum, spectral peaks
- Frequency-domain, high-level
 - Peak track birth/death statistics, instrument ID
- Many other possibilities (see below)

— MEDIA ARTS & TECHNOLOGY PROGRAM —

38

Feature Vector Examples

Field	Bringin' Da Noise	(I) Be Your Everyth...	Weighted
Volume Width	48.126621	47.903584	0.182064596871...
LPC Avg-Track-Dur	260.071	291.854	0.246736659056...
Bass Loudness	-3.82097	-3.48169	1.151592910141...
Spectral Contrast	17.8124	27.7138	1.260984294687...
LPC Track-Harmo...	1.15606	1.10925	1.386355020613...
BusyMid	399.87873138	382.9394489400...	2.090529929650...
Freq Max	0.579932	0.629081	2.756166578401...
Average Volume	34.344021	37.742193	3.092778888824...
Freq Avg	0.004416	0.004209	3.273244781783...
Tempo	111.966	105.943	3.872166433080...
LPC Peaks-Per-S...	258.61	229.837	5.144795608229...
LPC Freq-Deviation	6257.06	5584.61	5.146495852036...
% Freq Over Avg	24.050509	21.890819	5.313072728419...
Spectral Variety	57.0208	97.2588	5.591531132924...
BusyLow	412.44579522	341.0040312499...	6.891936456624...
Spectral Saturation	0.712956	0.651703	7.476978442821...
LPC Tracks-Per-S...	56.5431	48.2628	7.601499754612...
Snare Strength	0.328855	0.235586	8.982285629537...
Overall Grunge	0.248330529671...	0.067614786427...	12.20650524954...
% Rhythm	99.48301435406...	97.82279545454...	N/A
BEAT: hiquot	5.2	5.8	N/A
BEAT: maxscore	1550.0	926.0	N/A
BEAT: spikewon	0.0	0.0	N/A
BEAT: window	20.0	20.0	N/A

— MEDIA ARTS & TECHNOLOGY PROGRAM —

39

Example: FMAK3 Feature Table

```

class FeatureTable {
    // FeatureTable is a root object (no parents)

public:
    // Data members (instance variables)
    float mTimeStamp; // When do I start?
    float mTimeDur; // How long a time-span do I represent?

    // Time-domain features
    unsigned int mRMSWindowSize; // Size of RMS window
    FeatureDatum mRMS; // Rectangular-windowed RMS amplitude
    FeatureDatum mPeak; // Max sample amplitude
    FeatureDatum mLP RMS; // RMS amplitude of LP-filtered signal
    FeatureDatum mHPRMS; // RMS amplitude of HP-filtered signal
    size_t mZeroCrossings; // Count of zero crossings
    FeatureDatum mDynamicRange; // RMS dynamic range of sub-windows
    FeatureDatum mPeakIndex; // RMS peak sub-window index
    FeatureDatum mTempo; // RMS/FWT instantaneous tempo estimate
    FeatureDatum mTimeSignature; // Time signature guess
    FeatureDatum mBassPitch; // Bass pitch guess in Hz
    unsigned int mBassNote; // Bass note (MIDI key number) guess
    FeatureDatum mBassDynamicity; // Bass note dynamicity (size of histogram)

    // Spatial features
    FeatureDatum mStereoWidth; // L/R difference
    FeatureDatum mSurroundDepth; // Front/Surround difference
    FeatureDatum mCenterDistinction; // Center vs. L/R sum difference
  
```

— MEDIA ARTS & TECHNOLOGY PROGRAM —

40

Example: FMAK3 Feature Table, cont'd

```

    // Frequency-domain features
    unsigned int mFFTWindowSize; // Size of FFT window
    FtVector mSpectrum; // Hanning windowed FFT data (1024 points, or NULL)
    FtVector mReducedSpectrum; // 1-octave FFT data (10-12 points)
    FtVector mBandSpectrum; // 2.5-octave FFT data (4 points - spectral bands)
    FPartialVector mSpectralPeaks; // List of major spectral peak indices
    FPartialVector mSpectralTracks; // List of tracked peak frequencies
    FeatureDatum mSpectralCentroid; // Spectral centroid measure
    FeatureDatum mSpectralSlope; // Spectral slope measure
    FeatureDatum mSpectralVariety; // Inter-frame spectral variety measure

    // Hi-frequency properties
    FeatureDatum mHFReqBalance; // Relative HF level
    FeatureDatum mHFReqVariety; // HF inter-frame spectral variety
    FeatureDatum mHFReqCorrelation; // Correlation between HF and audio-band tracks
    FeatureDatum mSTrackBirths; // Spectral peak track births and deaths

    // LPC features
    unsigned int mLPCWindowSize; // Size of LPC window
    FPartialVector mLPCFormants; // List of LPC formant peaks
    FPartialVector mLPCTracks; // List of tracked LPC formants
    FeatureDatum mLPCResidual; // LPC residual level (noisiness)
    FeatureDatum mLPCPitch; // Pitch estimate
    FeatureDatum mLTPTrackBirths; // LPC formant peak track births, deaths

    // Wavelet-domain (Fast Wavelet Transform = FWT) features
    FtVector mWaveletCoeff; // FWT coefficient or NULL
    FtVector mWTNSpectrum; // Reduced FWT HiFreq noise spectrum
    FtVector mWTTracks; // List of tracked FWT peaks
    FeatureDatum mWTNoise; // FWT noise estimate
  
```

— MEDIA ARTS & TECHNOLOGY PROGRAM —

41

Example: FMAK3 Feature Collection

```

// A FeatureCollection is a list of FeatureTable structures stored for various times in a song.
class FeatureCollection : public RefCounted {
public:
    // Per-selection features
    const char * mName; // mArtist, * mTitle, * mAlbum; // track metadata
    unsigned mVersion, mTrack, mYear, mSampleRate; // DB track data
    Genre * mGenre; // primary genre match
    FeatureDatum mDuration; // float dur
    unsigned mSizeInWindows; // # of rms windows

    // Main vectors of feature matrices - core data
    FeatureTableList mWindowData; // collection of feature tables for every window (10/sec)
    FeatureTableList mOneSecData; // collection of feature tables for the 1-sec average
    // Averaged or reduced data sets
    FeatureTable mAvgData; // Averaged FeatureTable data for the song
    FeatureTable mWeightedData; // Matrix of the "weighted" (peak/thresholded) data
    FeatureTable mTypicalData; // Matrix of the "typical verse" (selected by the segmenter)
    FeatureTable mSoloData; // Matrix of the "solo verse" (or solo instrument note)
    FeatureTable mNoteData; // Matrix of a single solo instrument note

    // Segmentation data
    FtVector mSegmentPoints; // Vector of segment cue points
    FeatureDatum mSegmentWeight; // Confidence weight of the segmentation
    FeatureDatum mVerseLen; // Most common segmentation interval
    FeatureDatum mFirstVerse; // Duration of initial segment

    // Segment/envelope statistics
    FeatureDatum mQuietSections; // % of long-ish quiet sections
    FeatureDatum mLoudSections; // % of long-ish tutti sections
    FeatureDatum mRepeatSections; // % of repeated sections
    FeatureDatum mFadeIn; // dur of initial fade-in
    FeatureDatum mFadeOut; // dur of final fade-out
  
```

— MEDIA ARTS & TECHNOLOGY PROGRAM —

42

Code Exercises

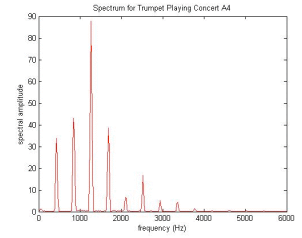
- Buffer, Window classes (see CSL)
- Analyzer class (see FMAK, Marsyas)
- Driver, main(), aubio, libxtract
- IO libraries (libSndFile, PortAudio)
- DASP libraries (libTSP, etc.)

MEDIA ARTS & TECHNOLOGY PROGRAM

43

Development Stages

- Dimensions
 - Content format
 - Low-level analysis procedures
 - High-level derived features
 - DB design
 - Application flow and integration
- Design Issues
 - System architecture and design impacted by each of the MDB dimensions



MEDIA ARTS & TECHNOLOGY PROGRAM

44

Content Format

- Impacts all levels of system
 - Data volume, storage options, analysis DSP, DB design, etc.
- Systems may or may not maintain original source content (vs. metadata)
- Systems may preserve several formats of source and metadata (n-tier)
- This is typically a given rather than a design option

MEDIA ARTS & TECHNOLOGY PROGRAM

45

Content Formats

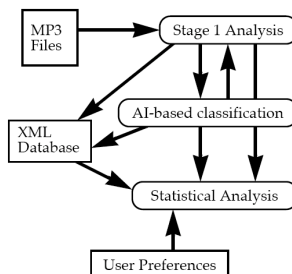
- Audio-based
 - Properties / volume of source recordings
 - MP3 / AAC / WMA decoders
- MIDI-based
 - Problems with MIDI, assumptions to make
 - Human-performed vs “dead pan” MIDI
- Score image based
 - Useful, but not treated here
- Formal language-based
 - SCORE, SMDL, Smoke, etc.
 - MusicXML

MEDIA ARTS & TECHNOLOGY PROGRAM

46

Real Applications

- DBMS issues
- Query systems, browsers, and MIR frameworks
- Informed tools
- Stand-alone delivery applications



MEDIA ARTS & TECHNOLOGY PROGRAM

47

Summary

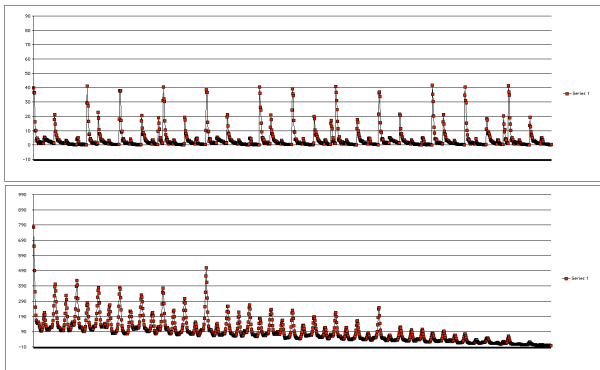
- Overview of MIR / MDB domain
 - Technology
 - Applications
- RDB and MMDB Design
 - Indexing and feature vectors
 - Core data structures for features, vectors, tables, collections
- MIR / MDB content formats

MEDIA ARTS & TECHNOLOGY PROGRAM

48

Topic 2

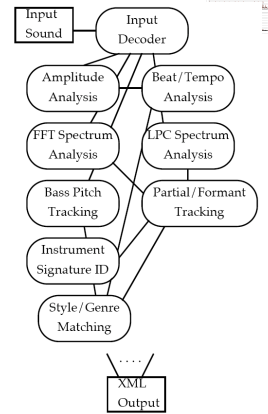
Signal Analysis for Feature Extraction



49

Topic 2: Signal Analysis

- Time-domain windowing
- Standard (domain-specific) analysis techniques
- Peak finding, peak continuation
- Feature statistics
- Perceptual mapping
- Deriving higher-level features



50

In the Reader

- Pope, Holm, Kouznetsov feature vector design overview
- Pitch detection papers
- Peak detection and continuation
- Spectra and spectral envelopes
- Multi-stage processing: NN, ICA, etc.

51

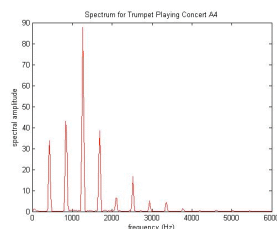
Code Exercises

- FFT-based processing
- Instrument signatures
- LPC and F0 tracking
- Speech detection
- Perceptual mapping
- Distance vectors and segmentation

52

Audio Signal Processing for Feature Extraction

- Time Sequences, Windowing
- Analysis Domains and Transformations
- Multi-level Analysis and High-level Features
- Data Smoothing and Reduction Techniques



53

Analysis Domains and Transformations

- Time-domain Audio Analysis and Applications
 - Windowed RMS Envelope Extraction
 - Beat Detection and Tempo Analysis
 - Time-based signal segmentation
- Frequency-domain Analysis
 - Pitch Detection Techniques
 - Spectral Analysis and Interpretation
 - Spectral Peaks and Tracking
 - Other Spectral Measures
- Other Kinds of Analysis: Wavelets
- Cross-domain analysis

54

Feature Extraction and Signal Analysis

- Multi-step process
 - Read input
 - Apply window
 - Derive several low-level features
 - Map, derive next-level features
 - Possible heuristics determine which next-level features are relevant
 - Prune data when appropriate

55

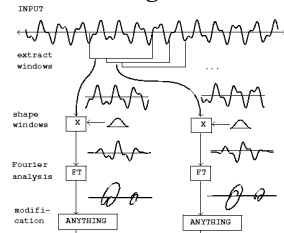
Windowed Amplitude Envelopes

- Choice of window size, hop size, window function shape
- May use several frequency bands (kick drum vs. hi-hat)
- Useful for silence detection, beat tracking, simple segmentation, summarization, etc.
- Simple, effective, well-understood

56

Time Sequences, Windowing

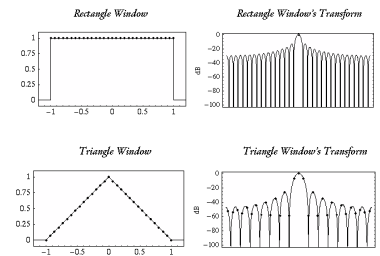
- Read audio input
- Vector multiply by window function
- Perform analysis
- Step to next window
- Hop size not normally = window size
- Window features
 - Main lobe width, side lobe level, side lobe slope



57

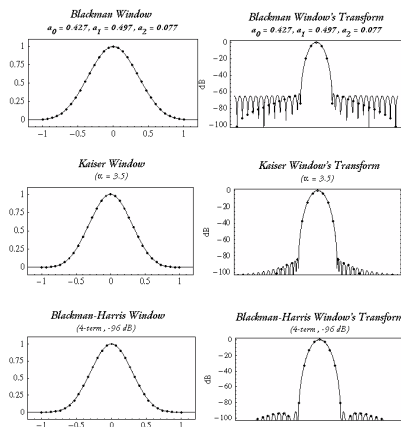
Windows and their Spectra (see MAT 240B)

- Trade-offs between window characteristics
- Different windows for different analysis domains



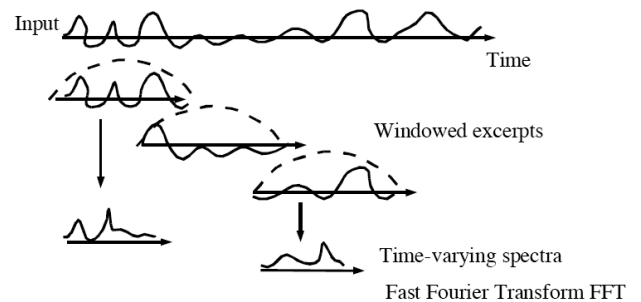
58

Advanced Windows for Spectral Analysis



59

Windowing and the STFT



60

Multi-window Multi-rate Analysis

Example: FMAK3 analysis driver

```
-r rmsWindow_size rmsHop_size
    # window size and hop size for the RMS time-domain analysis
-f fftWindow_size fftLen fftHop_size
    # for the FFT spectral analysis
-l lpcWindow_size lpcOrder lpcHop_size
    # for the LPC analysis
-w fwtWindow_size fwtLen fwtHop_size
    # for the wavelet analysis (not used in this version)
```

61

Time-domain Audio Analysis and Applications

- Use rectangular window if no overlap or triangular window if overlapping
- Medium-sized window (10 Hz or better resolution desired)
- Derived windowed RMS value
- Count zero crossings

62

Windowed RMS Envelope Extraction

C code for envelope extraction

- Outer loop for windows
- Inner loop to run window and compute RMS value
- Silence threshold (noise gate)
- Note-on trigger (peak detector)
- Example sound: piano sample, drum loop

63

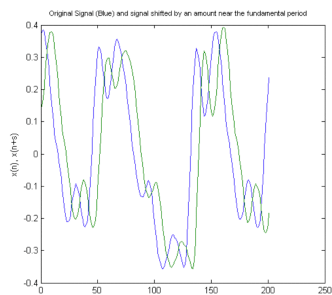
Optional Time-domain Steps

- Pre-filter to get low-freq and high-freq RMS values
- Process stereo channels to get M/S (sum/difference) signals
- Noise detection
- Silence detection
- Loop code examples and main(s)

64

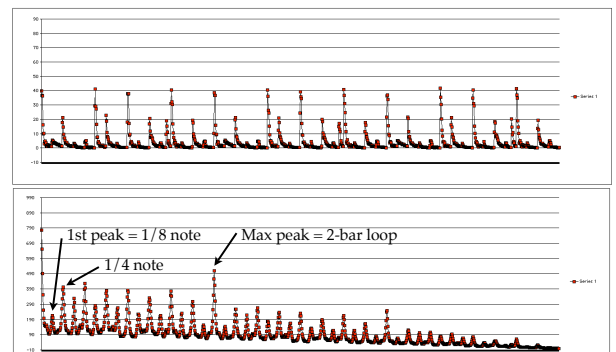
Auto-Correlation

- Slide a signal across itself, taking the vector product at each step
- This AC array has a peak at 0 and the period of the signal
- No peaks for noise

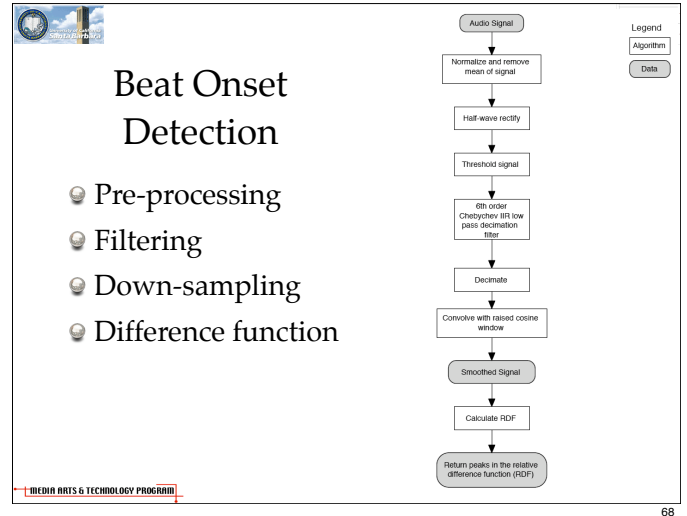
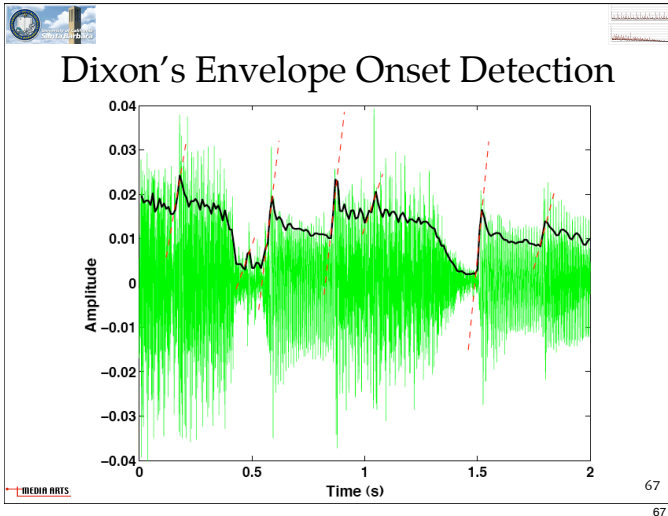


65

Windowed RMS and its Autocorrelation (for drum loop)



66



Code Exercises

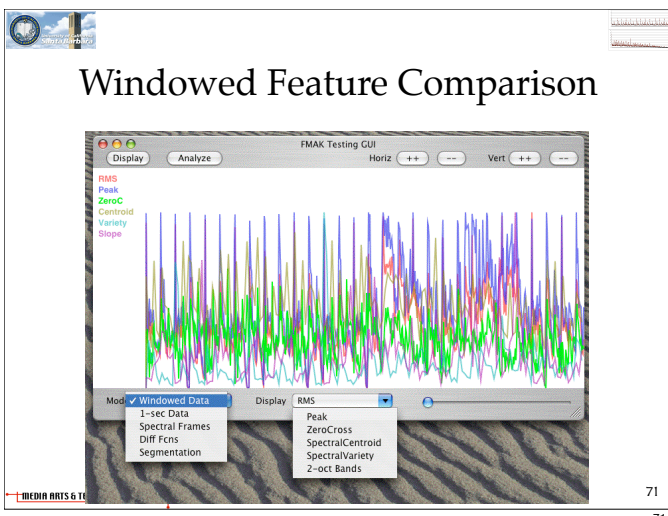
- Multi-band time-domain processing
- Advanced numerical processing
 - Autocorrelation
 - Adding constraints
 - Other techniques: neural nets, etc.
- Advanced applications
 - Segmentation
 - "Find the 1"

69

Applications

- Beat follower
- Tempo tracker
- Sing-along
- Song sync
- Accompaniment
- Score alignment

70



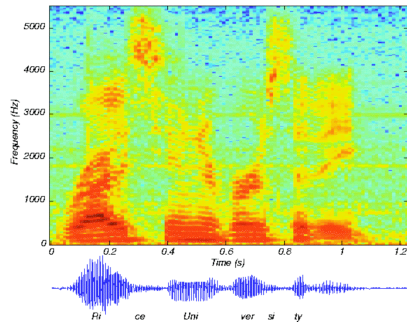
Frequency-domain Analysis

- Short-time Fourier transform
 - Configuration options and trade-offs
 - Interpretation/weighting of spectral bins
- Other frequency-domain techniques
 - Filter banks
 - Linear prediction
 - Filter matching

72

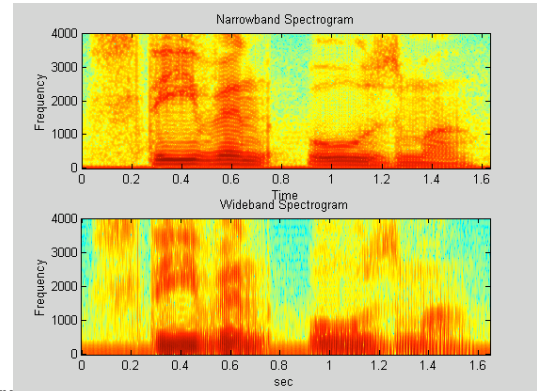
Speech Spectrogram

- Kinds of spectral plots
- Features



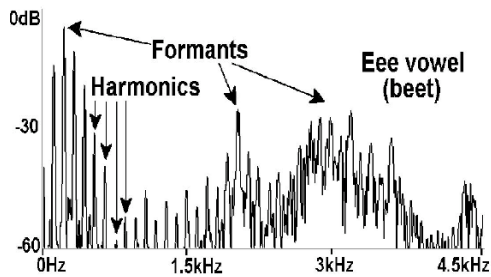
73

The Pitch/Time Trade-off



74

Harmonics and Formants



- Source/Filter - instr resonances

75

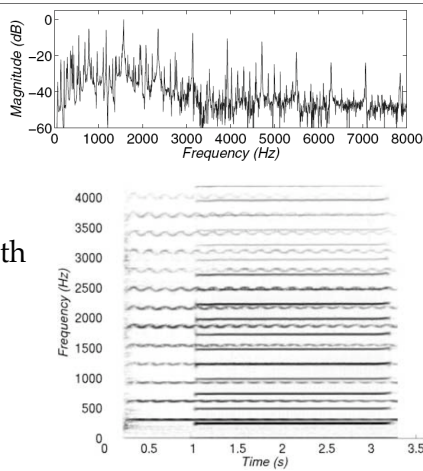
Using FFT APIs

- Simple FFT
 - See MAT240B
 - See F. R. Moore's *Elements of Computer Music*
- FFTW
 - FFTW data types
 - FFTW plans
 - See CSL Spectral class

76

Composite Spectra

- How to disambiguate?
- Track birth/death statistics
- Vibrato (see figure)
- Statistical techniques



77

Spectral Analysis and Interpretation

- Spectral data extraction
 - Base frequency
 - Overtone spectrum
 - Formants, resonances, regions
 - Instrument signatures
- Spectral statistics
 - Peak, mean, average, centroid, slope, etc.
 - Spectral variety, etc.

78

MEDIA ARTS & TECHNOLOGY PROGRAM

MEDIA ARTS & TECHNOLOGY PROGRAM

The figure consists of two parts. The top part is a graph with 'freq.' on the vertical axis and 'time' on the horizontal axis. It shows a dashed curve labeled 'true lobe' and a solid curve labeled 'parabola' that approximates it. Three points on the 'true lobe' are marked with circles and labeled 'lobe samples'. The bottom part is a grid of waveforms. The vertical axis is labeled 'freq.' and the horizontal axis is labeled 'time'. The grid shows several waveforms that change shape over time. Labels with arrows point to specific features: 'start up' points to the beginning of a waveform, 'peak' points to the maximum amplitude, 'turn off' points to the end of a waveform, and 'reverse' points to a waveform that is inverted.

MEDIA ARTS & TECHNOLOGY PROGRAM



```

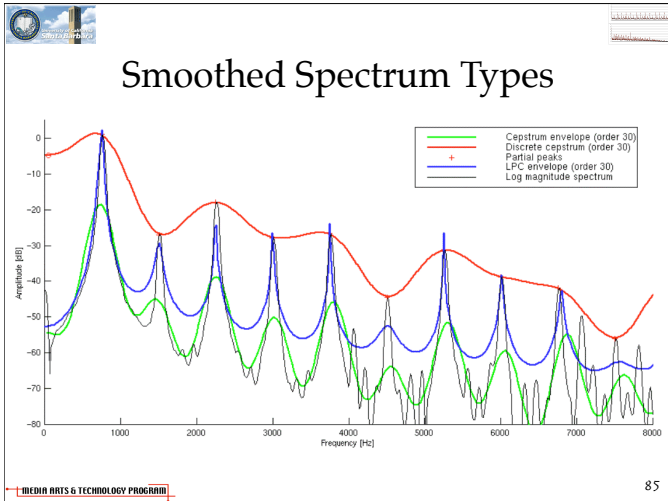
graph TD
    Start([Start]) --> 400
    400["maxAmp = MAX(max[1], frameSize/2)"] --> 402
    402["binIndex = 0  
peakCount = 0  
freqnc = samplingRate / frameSize"] --> 404
    404["binIndex = binIndex + 1"] --> 406
    406{"Is  
binIndex >  
(frameSize/2)"} -- Yes --> 408
    406 -- No --> 408
    408["left = max(binIndex-1)  
center = max(binIndex)  
right = max(binIndex+1)"] --> 410
    410{"Is  
center > left  
AND  
center > right"} -- No --> 412
    410 -- Yes --> 412
    412{"Is  
center > 0.05  
* maxAmp"} -- No --> 414
    412 -- Yes --> 414
    414["deviation = 0.5 * (left - right) / (left - 2 * center + right)  
peakLeft = peakCount + (binIndex - freqnc) * (deviation + freqnc)  
peakRight = peakCount + center - (0.25 * left - right) * deviation"] --> 416
    416["peakCount = peakCount + 1"] --> 404

```

MEDIA ARTS & TECHNOLOGY PROGRAM

Figure 10 consists of two subplots, (a) and (b), showing the magnitude of $Z(k)$ versus frequency (Hz). Both plots have a y-axis labeled "Magnitude of $Z(k)$ " ranging from 0 to 40 and an x-axis labeled "Frequency (Hz)" ranging from 0 to 6000. Subplot (a) shows the magnitude of $Z(k)$ for the original signal, with several peaks labeled with numbers: 3, 6, 9, 12, 15, 18, 21, 24, 27, and 30. Subplot (b) shows the magnitude of $Z(k)$ for the reconstructed signal, with a smooth curve overlaid on the peaks, indicating a reconstruction process.

MEDIA ARTS & TECH



Deriving Spectral Bands

- Octave-band loops
 - outer loop - step size doubles every octave
 - inner loop - sums bins in range
- Weighted, non-linear bands

86

Spectral Tools

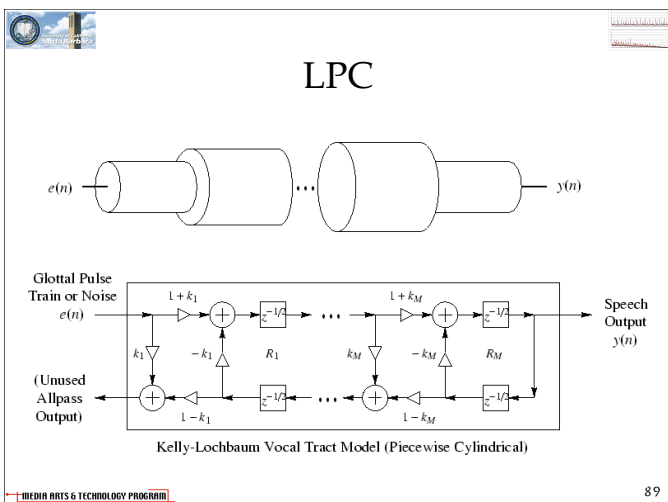
- SPEAR
- Loris
- Marsyas
- Sonic visualizer
- FMAK SysID

87

Code Exercises

- Time-D windowed analysis
 - Statistics
- FFT analysis
- Spectral statistics
- FFT- and LPC-based tracking
 - Single note discrimination
 - Chord detection

88



Linear Predictive Coding

- Assume a periodic (pitched) time-domain signal can be approximated by an n-th-order polynomial
- AOK, except that there will be periodic errors
- The polynomial factors (weights) can be interpreted in several ways
 - "Reflection coefficients"
 - Filter coefficients
- Relationship to source/ filter model

90

LPC: The Math

Polynomial $s(n) \approx a_1 s(n-1) + a_2 s(n-2) + \dots + a_p s(n-p),$

As summation $s(n) = \sum_{i=1}^p a_i s(n-i) + G u(n),$

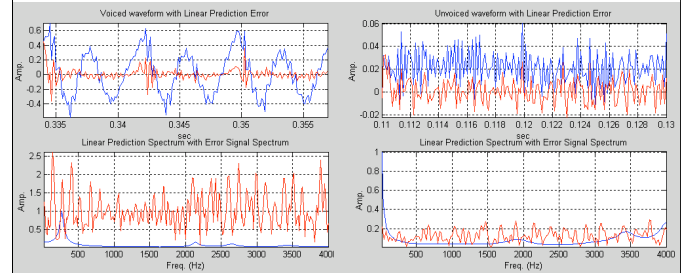
In Z-domain $S(z) = \sum_{i=1}^p a_i z^{-i} S(z) + G U(z)$

Error signal $e(n) = s(n) - \hat{s}(n) = s(n) - \sum_{k=1}^p a_k s(n-k)$

Transfer function $H(z) = \frac{S(z)}{G U(z)} = \frac{1}{1 - \sum_{i=1}^p a_i z^{-i}} = \frac{1}{A(z)}$

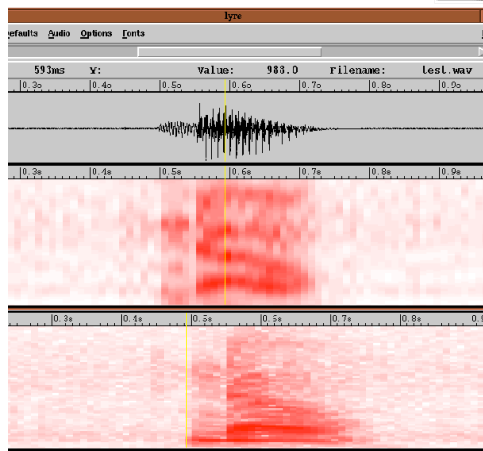
91

Examples for Voiced and Unvoiced Speech



92

LPC vs. Power Density Spectrum



93

LPC APIs: LibTSP in FMAK

```
// LPC analysis -- get autocorrelation
// void SPautoc (const float x[], int Nx, float cor[], int Nt)
SPautoc (mSampleBuffer, mWindowSize, mCorr.GetRawData(), mLPCOrder + 1);

// Find predictor coefficients from autocorrelation values
// double SPcorXpc (const float rxx[], float pc[], int Np)
featureTable.mLPCResidual = SPcorXpc (mCorr.GetRawData(), mLPCOrder);

// Convert predictor coefficients to filter coefficients
// void SPpcXec (const float pc[], float ec[], int Np)
SPpcXec (mPred.GetRawData(), mLPCCoeffs, mLPCOrder);

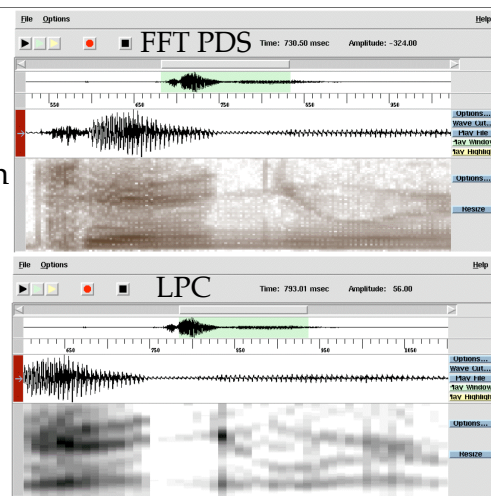
// Convert predictor coefficients to cepstral coefficients
// void SPpcXcep (const float pc[], int Np, float cep[], int Ncep)
SPpcXcep(mPred.GetRawData(), mLPCOrder, mLPCCoeffs, mCepst);

// Locate the spectral peaks
// locate_peaks(FtVector * fv, FtVector * res, epsilon, spacing);
mPeakExtractor.LocatePeaks( mPred, featureTable.mLPCFormants, 4 );
```

94

Spectral Analysis Comparison

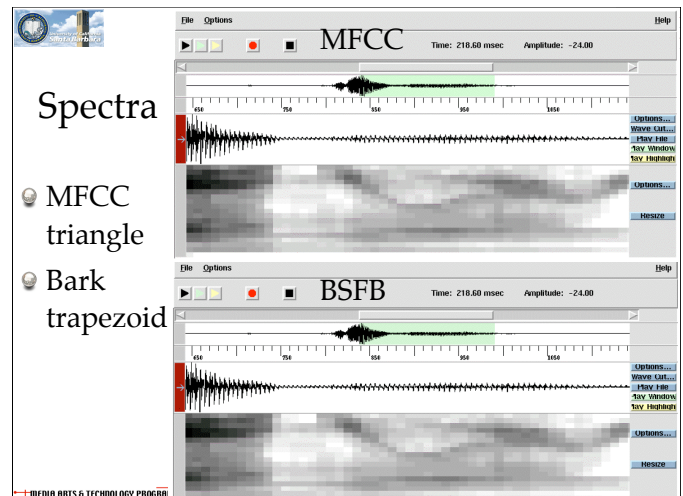
- Pitch/time trade-off
- Scalability of the model
- Feature extraction



95

Spectra

- MFCC triangle
- Bark trapezoid



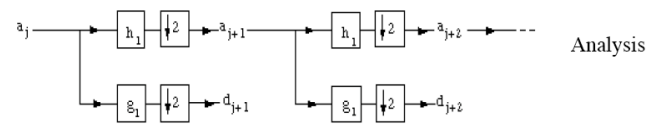
96

Wavelets and Grains

- Deterministic wavelet transform (iterative filtering and decimation)
- Interpretations of wavelet coefficients
- Time-domain decomposition and granular representations
 - “Matching pursuits” solutions
 - Decomposition dictionaries
 - Gaborlets vs. noise bursts
 - Current systems are unacceptably slow

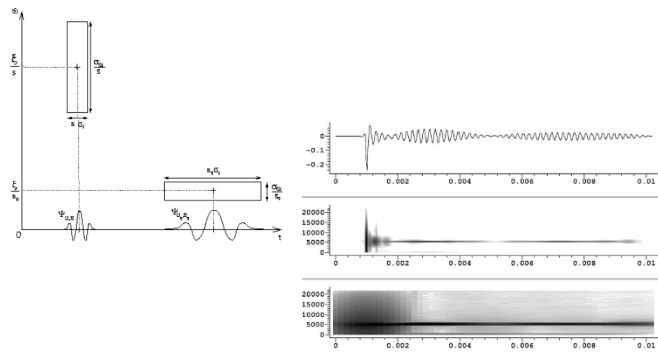
Wavelet Transform

Octave filterbank

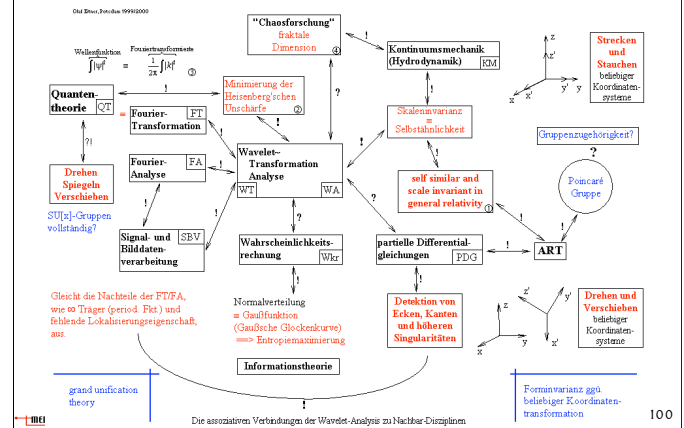


- Successive Decimation
- View as bursts of different length scales

FT vs WT

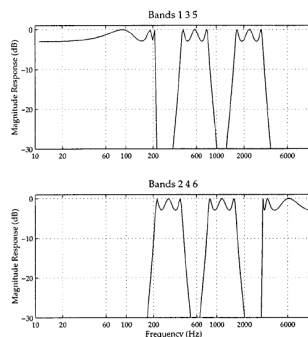


Wavelets and The New Math



Filter Banks

- Filter bank vs. spectrum
- When to use
- How to implement



Working with Spectral Data

- Spectral statistics
 - Slope, centroid, band weights, variety, etc.
- Instrument signatures
 - Solo instrument identification
 - Location of style-indicative instruments
 - Still a challenge in mixed/compressed music
- Pitch tracking
 - Melody instrument vs bass tracking
 - Generally requires segmentation and/or band-pass filtering

Code Exercises

- Pitch detection
 - Time-domain methods
 - Frequency-domain methods
 - Example sounds: single melodies
- Spectral statistics
 -

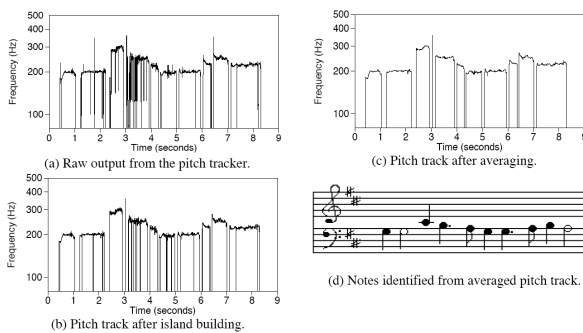
103

Code Exercises

- FFT spectral extraction
 - Real vs/ complex
 - Magnitude/phase spectra
- Weighting of spectral data
 - Numerical scaling
 - Perceptual scaling
- LPC spectral extraction
- Peak location, continuation

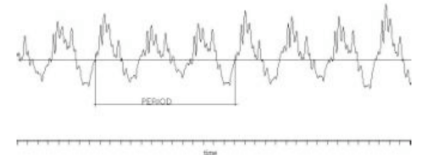
104

Pitch Detection



105

Pitch Detection Techniques



- Find the period of a “periodic” signal
 - First guess whether or not it’s periodic
- Simple techniques work for many signals
 - Zero-crossings (with direction, slope)
 - Autocorrelation (with range limitation)
- It’s hard to tell when they fail
 - Random data, silence
 - Octave over/under-tone errors

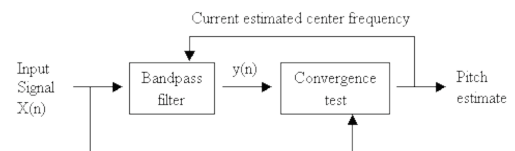
106

AC PDet

- Audio
- Libxtract
- cmix
- LipTSP
- CSL/FMAK

107

Filter-based Pitch Detection

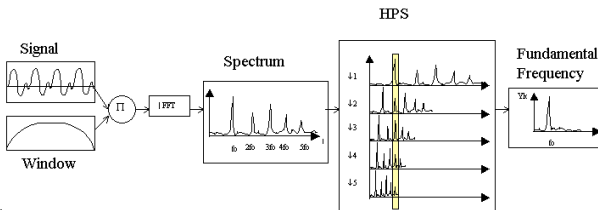


- Simple adaptive process for single-frequency source with strong fundamental (i.e., many, but not all, instruments and voices)
- Easily implemented in analog circuitry
- Many variations

108

Harmonic Product Spectra

- Decimation of FFT spectra, summation, and spectral peak location
- Assumes overtones are significant, not that fundamental is



09

109

Harmonic Product Spectra

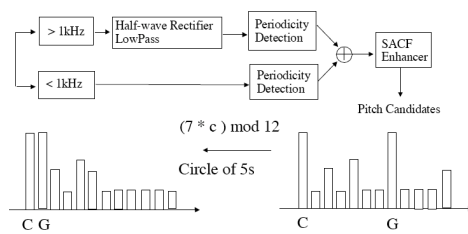
- Implementation
 - Outer loop (octaves)
 - Scales copy of spectrum into buffer
 - Inner loop
 - Take max or avg of sub-window?
 - Use interp. peak picker?
- Post
 - first max > min_val

110

110

Multi-Pitch Estimation

- Still rocket science in the general case



111

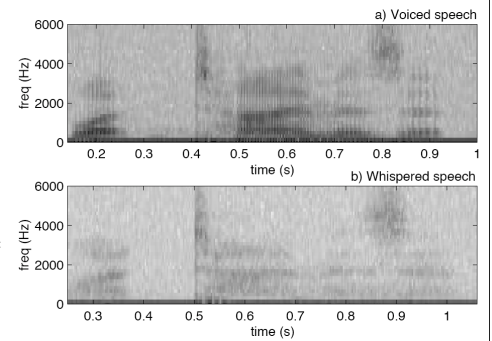
111

FFT vs LPC: Voiced / Unvoiced Speech

- Example: "What time is it?"

Voiced = normal

Unvoiced = whispered



112

112

Mel-Freq Cepstral Coefficients

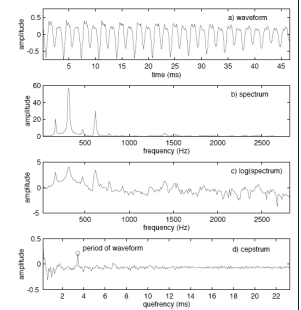
- Steps:
 - Signal
 - FT
 - Log magnitude
 - Phase unwrapping
 - FT (or DCT)
- Name reversal
- Interpretations
 - Quefrency
 - Mel-scale
 - Mel-scale filters
- Instead of AC, use FFT or DCT of PDS
- Leads to interesting statistics of higher-level spectral properties, see next section

113

113

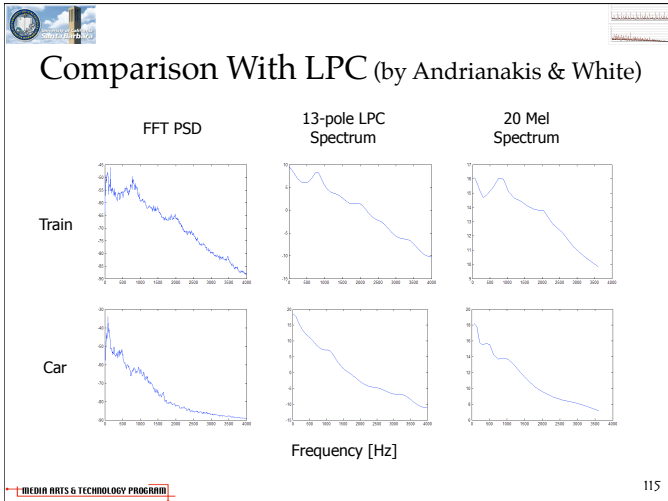
MFCC Analysis

- Analogy
 - Start with log spectrum of mixed complex tones: several sets of related partial peaks
 - Take, e.g., the autocorr. of the FFT PDS
 - Warped frequencies of peaks correspond to fundamental frequencies of overtone series



114

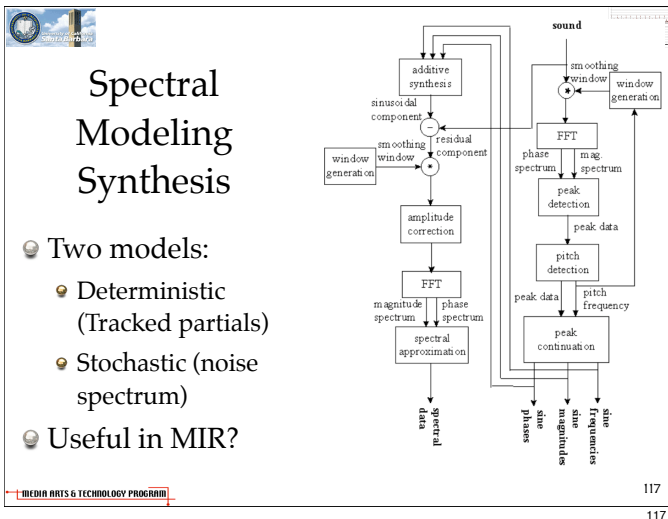
114



Other Spectral Measures

- MFCC and harmony tracking
- Resonances and modes (blind source separation)
- Physical models (source/filter, excitation separation)
- Reduced spectra and spectral bands
- Other derived statistics (weights)
- Other perceptual mappings

116



Cross-domain analysis

- SMS as cross-domain, mixed-representation, multi-stage process
- Mixing time-domain, freq-domain, MFCC, and wavelet data
- Rhythm, tempo, beats
- Segmentation and pruning
- Learning multi-dimensional weightings
- You guessed it: Feature Vector Design

118

Multi-stage Analysis (again)

- Use initial time-domain analysis to focus frequency-domain stages
- Derive smoothed spectra (LPC, MFCC) for all, detailed spectra for selected windows
- Segmentation, compaction
- Tempo and segment-length statistics
- Weighting, distance metrics, fingerprints, search keys

119

Coding

- Speech understanding
- Accompaniment
- DB population
- DJ-ing
- Transcription
- Control

120

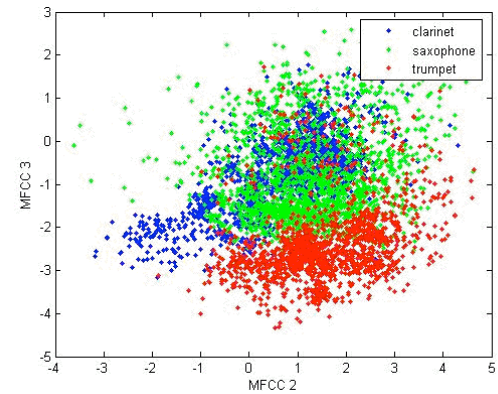
Summary

- Basic DASP techniques, windowing
- Domain-specific analysis techniques
- Peak detection and continuation
- Feature statistics
- Perceptual mapping
- Mixed-mode analysis

MEDIA ARTS & TECHNOLOGY PROGRAM

121

121



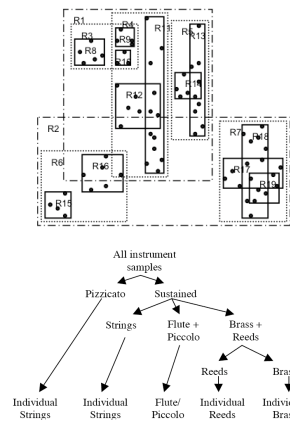
MEDIA ARTS & TECHNOLOGY PROGRAM

122

122

Topic 3: Numerical Processing

- Analysis Data Processing, Smoothing
- Dimensionality Reduction
- Clustering and Classification
- Segmentation and Form
- Audio Semantics and Scene Analysis



MEDIA ARTS & TECHNOLOGY PROGRAM

123

123

In the Reader

- Pattern recognition: Ellis, Dannenberg
- Segmentation
- Classification
- Techniques: SOM, SVM, string compression
- FMAK utilities

MEDIA ARTS & TECHNOLOGY PROGRAM

124

124

Code Exercises

- More input data smoothing, filters
- Scale/offset/threshold processing
- Curve-fitting and envelope extraction
- Handling groups of data: PCA, clustering
- Segmentation using a rich feature vector and weighted distance metric

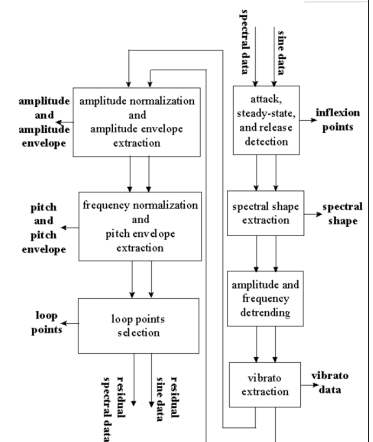
MEDIA ARTS & TECHNOLOGY PROGRAM

125

125

Multi-stage Processing

- Feature extraction
- Perceptual mapping
- Signal statistics
- Calculus
- Smoothing
- Segmentation
- Classification



MEDIA ARTS & TECHNOLOGY PROGRAM

126

126

Multi-pass Cross-domain

- Initial segmentation, ID of significant notes or sections based on time-domain or spectral track statistics
- Detailed analysis of short segments (see Fig 1 above to find first note of vocal)
- Suggests heuristic approach, agents, or blackboard architecture

127

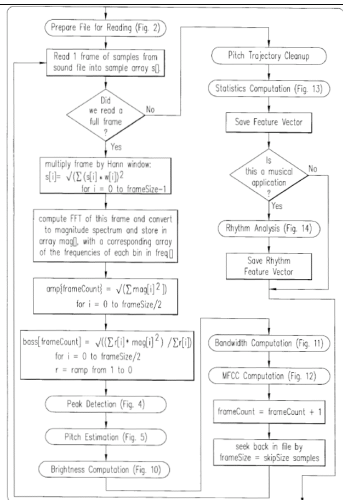
Derived Features

- Adding semantics to low-level features
- Perceptual mapping
- Tracking tempo, meter, harmonic center
- Segmentation: lengths, regularity, intro, coda, fade, per-segment averages
- Instrument identification (signatures)
- Location of important sections/ notes

128

Example

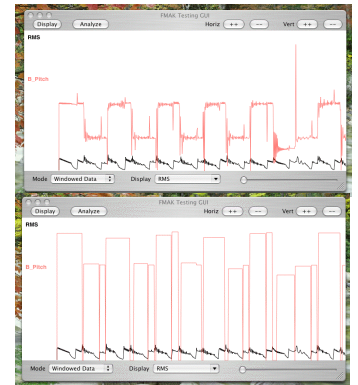
- Complex feature vector processing (from Blum et al. patent #5,918,223)



129

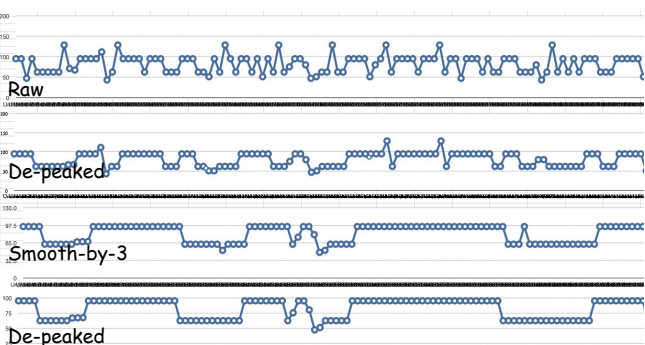
Data Smoothing and Reduction

- Windowing and processing artifacts
- Use statistical processing and perceptual limits
- Example: bass line for 12-bar blues, before and after smoothing
- Techniques
 - Leaky integrators
 - Sticky values and islands



130

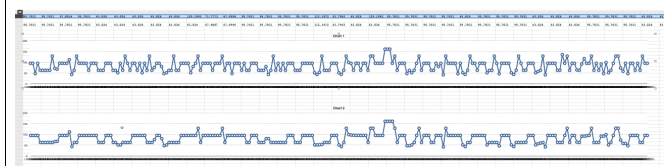
Smoothing (of tempo)



Now take histogram of smoothed value

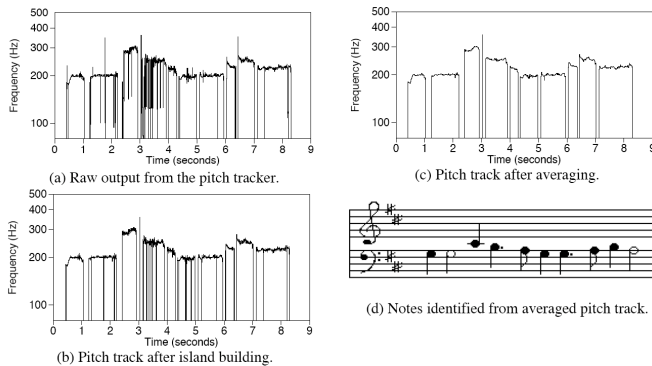
131

Simpler, faster Method



132

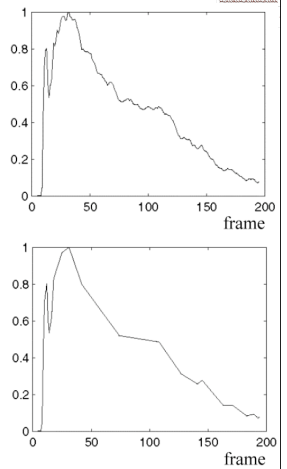
Typical Cleaning Products



133

Envelope Data Reduction

- Curve-fitting algorithms
- Least-squared error calculation
- Hierarchical techniques
- Geometrical techniques
- Catalog-based systems



134

Perceptual Mapping

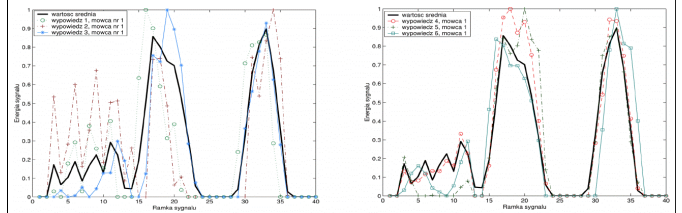
- Amplitude
 - dB scale, mapped by equal-loudness curves, grouped into freq. regions
- Frequency
 - Pitch-mapped, scale warped, spectral bands
- Envelopes
- Tempo
- See more examples

135

135

Noise and Variability

- Energy / time for word "pusc"
- Same speaker, multiple recordings

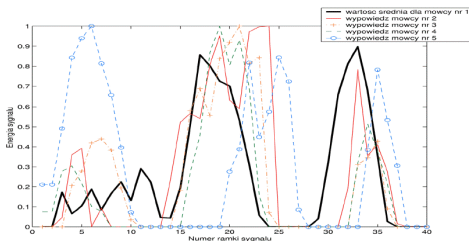


136

136

Variability between Speakers

- "Pusc"
- 5 speakers
- Avg. shown in black
- Common features?

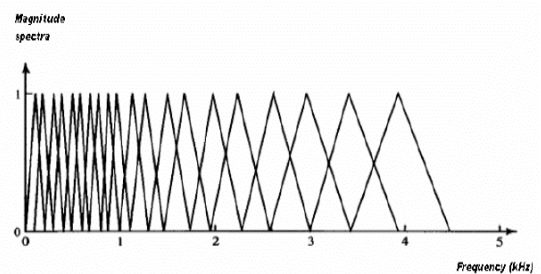


137

137

Frequency Regions and Scaling

- Mel-warped frequency bands



138

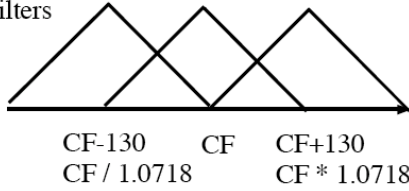
138

Mel Scale and Coefficients

Mel-scale

13 linearly-spaced filters

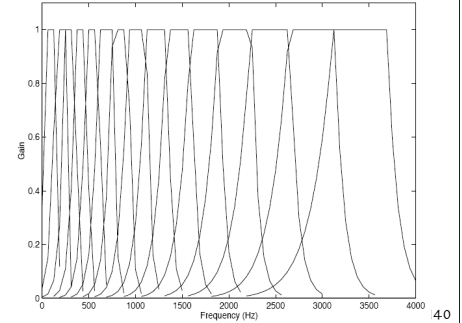
27 log-spaced filters



139

Bark-scale Trapezoidal Filters

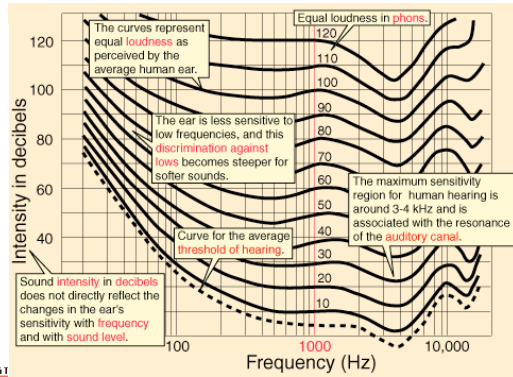
Similar spacing to Mel filters



140

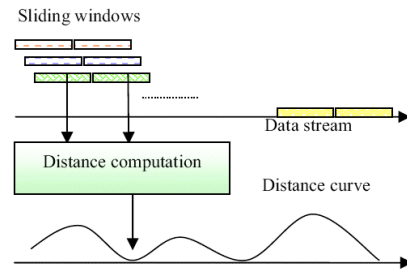
Equal-loudness Curves

Fletcher-Munson vs. Robinson-Dadson



141

Audio Segmentation



142

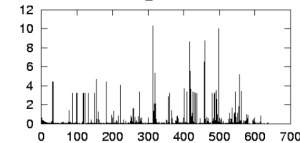
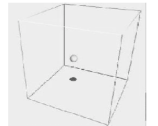
Audio Segmentation

- Typical audio segmentation requirements
 - Speech vs. music vs. other sound
 - Speaker change detection
 - Discover song verses/refrains
- Segmentation of Music
 - Assume somewhat regular segment length (verse)
 - Relationship to musical form guidelines
 - Assume good autocorrelation of segments
 - What feature weighting to use?

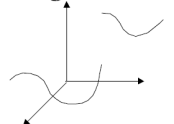
143

Time-based signal segmentation

- Time series of feature vectors $V(t)$
- $f(t) = d(V(t), V(t-1))$
 - $d(x,y) = (x-y)^T C^{-1} (x-y)$ (Mahalanobis)
- df/dt peaks correspond to texture changes



(GT)



144

Basic Segmentation Techniques

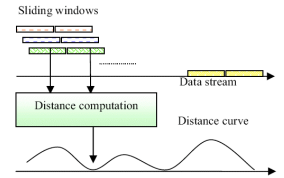
- Inter-window distance metrics
 - Choose a weighting of the feature vector that gives a good dynamic range (of distance)
 - Look for regular distance peaks (autocorrelation of distance metric)
 - Iterate as necessary
- Heuristic techniques
 - Neural nets, HMM
- Statistical techniques
 - Bayesian, Gaussian

145

145

Segmentation with a Distance Metric

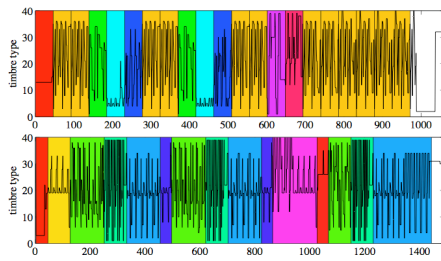
- Use overlapping windows
- Choose distance metric (feature weighting) carefully or try several of them
- Locate distance peaks, possibly assume they lie at regular intervals
- Examples: speech, single instrument, mixed songs



146

Segmentation Examples

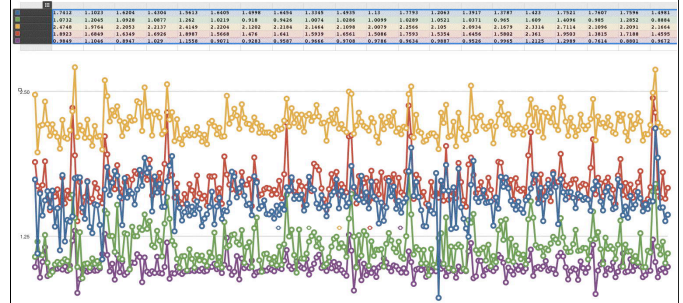
- SoundBite finds salient segments
 - 1/8 octave Constant Q transform, PCA, HMMs
- 8S (Siren) phoneme segmenter
 - 2-octave filter banks, RMS, LPC



147

147

Distance Metrics with 5 Weightings



148

148

FMAK Distance Metric Weighting

Spectral-/pitch-centric segmenter configuration

```
SegmenterConfiguration {
  kHPRMS 0.5
  kDynamicRange 0.5
  kZeroCrossings 0.5
  kBassPitch 0.5
  kSpectralSlope 1
  kSpectralCentroid 0.5
  kSpectralVariety 1
  kSpectralBandMax 0.5
}
```

MFCC- & tracking-centric configuration

```
SegmenterConfiguration {
  kHPRMS 0.2
  kSpectralVariety 1
  kBassDynamicity 1
  kZeroCrossings 0.2
  kBassPitch 0.5
  kTrackBirths 0.5
  kTrackDeaths 0.5
  kMFCCMax 1
  kMFCCFirst 1
  kMFCCAvg 0
}
```

RMS-centric configuration

```
SegmenterConfiguration {
  kRMS 1
  kLPRMS 1
  kHPRMS 1
  kSpectralVariety 0.5
  kSpectralBandMax 0.5
  kZeroCrossings 0.5
  kBassPitch 0.5
  kDynamicRange 1
  kTempo 0.5
}
```

Tracking-centric configuration

```
SegmenterConfiguration {
  kRMS 0.5
  kHPRMS 0.2
  kSpectralSlope 0.2
  kSpectralCentroid 0.2
  kSpectralVariety 0.5
  kSpectralBandMax 0.5
  kBassPitch 1
  kTrackBirths 1
  kTrackDeaths 1
  kMFCCFirst 1
  kTempo 0.5
}
```

149

149

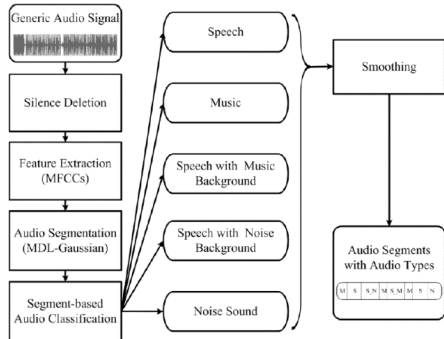
Rhythmic Segmentation

- Windowed amplitude envelope, extract note onsets if possible
- Look for changes in total energy (e.g., silence)
- Look for regularly-space events at several levels of hierarchy (assume ratios of 2 or 3 as basis)
- Works well for many genres

150

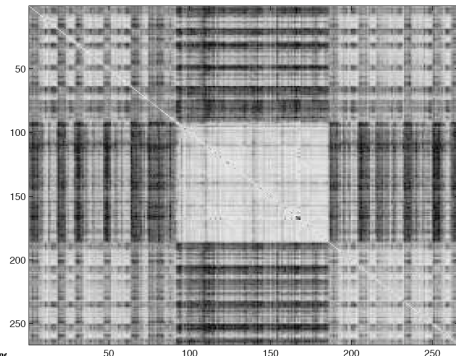
150

Segmentation/Classification Framework



151

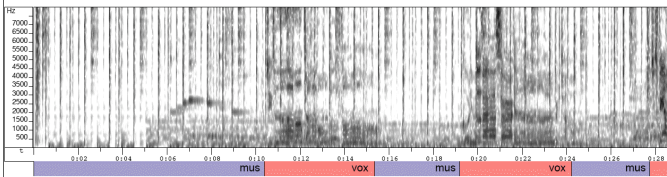
Self-Similarity and Recurrence Maps



152

Segmentation for Indexing

Berenzweig's singing detector



153

Segmented Feature Vectors

- Multi-stage feature extraction
 - 1: Data analysis and segmentation (using a complex set of distance metrics)
 - 2: Prune the data to maintain a small set of feature vectors (1-per-segment & song peak/average data).
- Add in segment-length statistics, confidence
- Delivers a complex feature matrix with segment-length data, segmentation confidence, and a collection of individual feature vectors

154

Using Segmentation

- Seg for summarization
- Seg for clustering
- Seg for finger-printing
- Seg statistics alone
 - # segs, confidence, fade I/O
- Song structure
 - typical/solo verses
- Solo instr, singer ID

155

Song Segmentation Statistics

```

32, FeatureDatum, TempoAvg, "TempoAvg", getFloat, true)           ///< Average tempo
33, FeatureDatum, TempoWeight, "TempoWeight", getFloat, true)    ///< Tempo tracker global confidence
34, FeatureDatum, TempoDistr, "TempoDistr", getFloat, false)     ///< Tempo tracker off-by-half confidence
35, unsigned, TrackerConf, "TrackerConf", getUInt, false)       ///< which peak tracker Config was used
36, FeatureDatum, SegmentWeight, "SegmentWeight", getFloat, true) ///< Confidence weight of the segmentation
37, unsigned, SegmentConf, "SegmentConf", getUInt, false)      ///< which segmenter Config was used
38, unsigned, NumSegments, "NumSegments", getUInt, true)        ///< # sections
39, FeatureDatum, VerseLength, "VerseLength", getFloat, false)   ///< Most common segmentation interval
40, FeatureDatum, FirstVerseStart, "FirstVerseStart", getFloat, false) ///< Duration of initial segment
41, unsigned, SoloIndex, "SoloIndex", getUInt, false)           ///< index of solo segment
42, unsigned, TypicalIndex, "TypicalIndex", getUInt, false)     ///< index of typical segment
43, FeatureDatum, SoloStart, "SoloStart", getFloat, false)       ///< start of solo segment
44, FeatureDatum, TypicalStart, "TypicalStart", getFloat, false) ///< start of typical segment
45, FeatureDatum, QuietSections, "QuietSections", getFloat, false) ///< % of long-ish quiet sections
46, FeatureDatum, LoudSections, "LoudSections", getFloat, false) ///< % of long-ish loud sections
47, FeatureDatum, FadeIn, "FadeIn", getFloat, false)            ///< dur of initial fade-in
48, FeatureDatum, FadeOut, "FadeOut", getFloat, false)          ///< dur of fade-out
49, FeatureDatum, SoloCentroid, "SoloCentroid", getFloat, true)  ///< Solo verse spectral centroid ratio
50, FeatureDatum, SoloVariety, "SoloVariety", getFloat, true)    ///< Solo verse spectral variety ratio
51, FeatureDatum, SoloDynRange, "SoloDynRange", getFloat, true)  ///< Solo verse dynamic range ratio
52, FeatureDatum, SoloRMS, "SoloRMS", getFloat, true)           ///< Solo verse RMS ratio
53, FeatureDatum, SoloTempo, "SoloTempo", getFloat, true)       ///< Solo verse tempo ratio

```

156

Multi-table Feature Sets

- 💡 What to store
 - 💡 Song avg data
 - 💡 Peak feature values
 - 💡 Feature variance
 - 💡 "Typical" verse
 - 💡 "Solo" verse
- 💡 Use Song table with 2-5 vectors per song

157

Code Exercises

- 💡 Envelope Data Reduction
 - 💡 ADSR from RMS
 - 💡 Clean pitch tracking
- 💡 Rhythmic segmentation
 - 💡 Autocorrelation of multi-band RMS data
- 💡 Distance metrics for segmentation
 - 💡 RMS, spectral bands, MFCC, formants, etc.
 - 💡 Spectral statistics
 - 💡 Segmentation

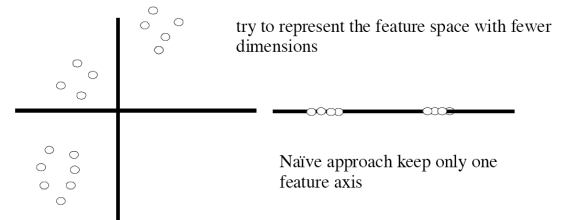
158

Processing Large Data Sets

- 💡 Given a rich feature vector for a large data set
 - 💡 Dimensionality reduction
 - Find a small set of discriminating variables or feature weightings
 - 💡 Clustering
 - Find groups in a higher-dimensional space
- 💡 (Related?) Classify a sound according to a set of criteria and/or a catalog

159

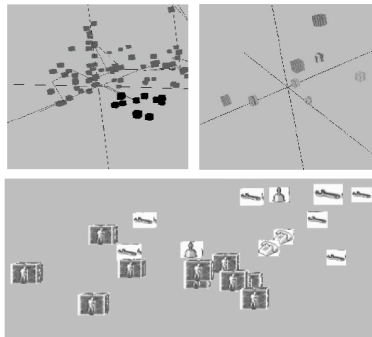
Dimensionality Reduction



160

Dimensionality Reduction

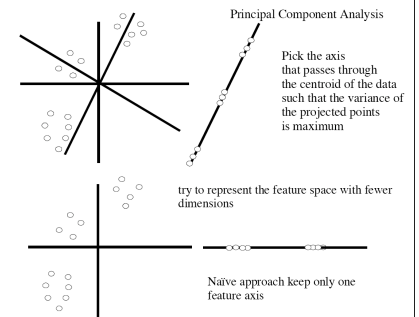
- 💡 Grouping of large data sets
- 💡 PCA & ICA
- 💡 KNN and clustering



161

PCA

- 💡 Find the *principal components* or *axes of discrimination* of a set of n-dimensional data points

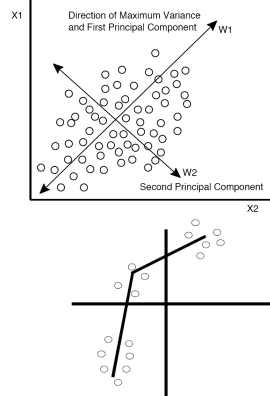


- 💡 AKA Karhunen-Loève transform (or KLT) or data whitening

162

PCA

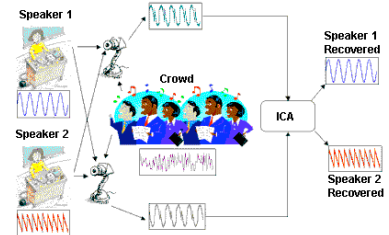
- Easy with clear clusters or equal density
- Often necessary to repeat with different weighting metrics to find good clustering criteria
- Statistics based on covariance matrices



163

Independent Component Analysis (ICA)

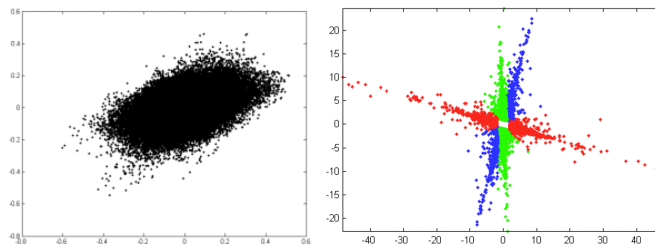
- Relative of non-linear PCA
- Discovers independent basis functions
- Effective for blind source separation



164

PCA Examples

- Same feature vectors, different weightings
- 90% of FV is "noise"



165

Problems Working in High-dimensional Spaces

- „Curse of dimensionality“
 - High overlap of MBRs in almost all directory nodes
 - Hence, almost all (data) pages have to be scanned
 - Query time approaches $O(n)$: sequential scan becomes competitive
- High-D and nearest neighbors
 - Small statistical variance for distances to other points in data base
 - Is „nearest neighbor“ meaningful in high-D spaces? [BGRS 99]
 - Reasonability of NN depends on data distribution [Keim et al.]
 - Real data are typically not distributed evenly but form clusters
- Techniques to cope with problems of high-D
 - Reduction of dimensionality
 - Accelerating the sequential scan (VA-File, IQ-tree, ...)

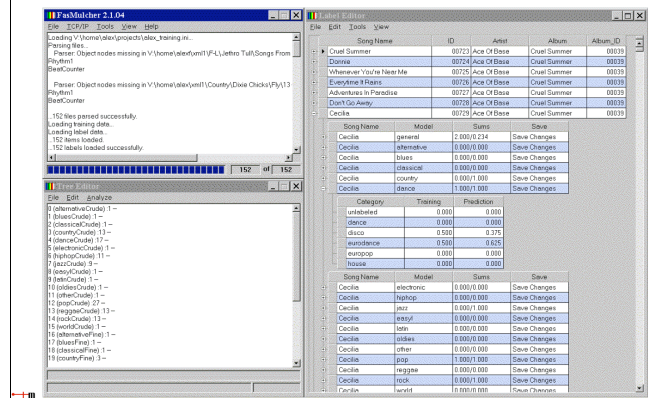
166

Dimensionality Reduction

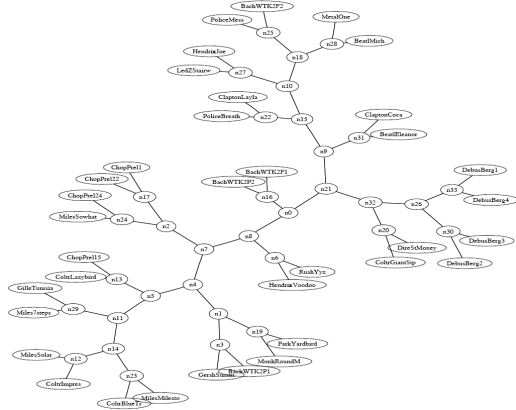
- Linear Reduction Techniques
 - First phase: Rotation of data space
 - Principal Components Analysis (PCA, KLT): $O(d^3) + O(d^2)$
 - Discrete Fourier/Cosine Transform: $O(d \log d)$
 - Wavelet Transform: $O(d)$
 - Second phase: Projection to lower dimensions ($d \rightarrow r$)
 - After rotation, just discard the last $(d - r)$ coordinates
 - L_p distance values for sure get smaller: $\left(\sum_{i=1}^r |p_i - q_i|^p \right)^{1/p} \leq \left(\sum_{i=1}^d |p_i - q_i|^p \right)^{1/p}$
- Non-linear techniques
 - Multi-Dimensional Scaling (MDS)
 - FastMap (extension of MDS) [Fal+]

167

Regression Trees & CART

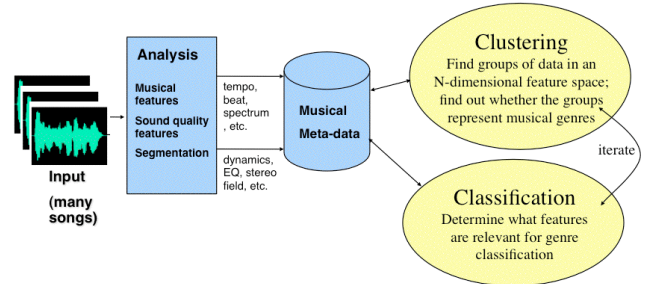


168

169

Gathering data

Deriving higher-level meta-data

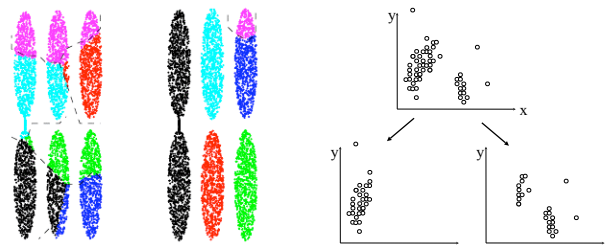


170
170

- 🧠 Clustering: Given a feature vector derived for a data set, find the feature weighting for which the data groups are best delineated
- 🧠 Classification: relate clusters to perceptual groups such as speech/music/snd-effect or musical genres

MEDIA ARTS & TECHNOLOGY PROGRAM

171



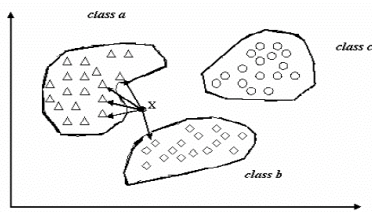
MEDIA ARTS & TECHNOLOGY PROGRAM

172

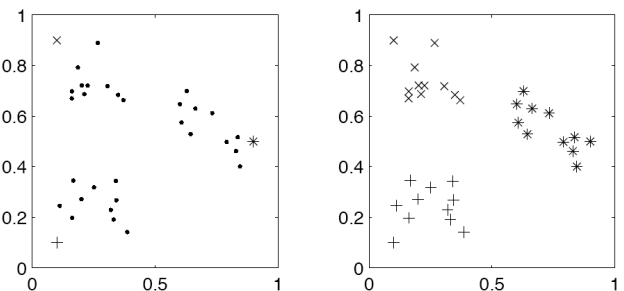
- ☞ For each item, locate its k nearest neighbors and collect the distances to them
- ☞ Apply a threshold to identify data clusters
- ☞ Change distance metric as necessary

$$d_q\text{-Ranking } r_q: \mathfrak{I}_{DB} \rightarrow DB$$

$$i_1 \leq i_2 \Rightarrow d(r_q(i_1), q) \leq d(r_q(i_2), q)$$

 k -nearest neighbors: $r_q(\mathfrak{I}_k)$ 

173

**MEDIA ARTS & TECHNOLOGY PROGRAM**

174

Advanced KNN/Region Systems

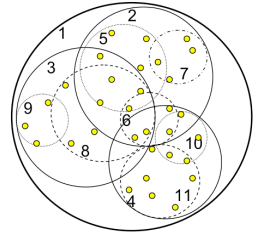
- Hierarchical indexes (trees) for high dimensions
 - Adopt the size of directory nodes in order to locally simulating sequential scans of the data
 - Examples: X-Tree [BKK 96], IQ-Tree [BBJK+ 00]
- Metric Indexes for spaces w/o dimensions
 - For arbitrary distance functions (edit dist, morphological dist)
 - Applicable even to high-dimensional spaces
 - Examples: M-Tree [CPZ 97], Slim tree [FSTT 00]
- Anchoring Methods (vantage points, etc.)
 - Select r landmarks among the data objects
 - Use distance values to r landmarks as r dimensions
 - Examples: vp-tree [FCCM 00], mvp-tree [TÖ 97], [VV 02]

175

Region and Sphere Trees

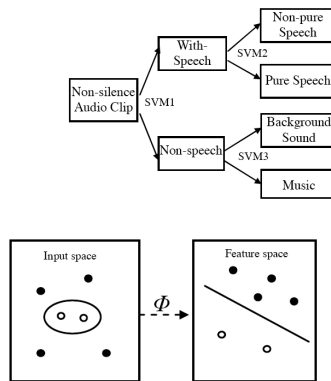
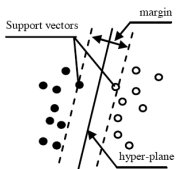
- Many variants of N-D KNN algorithms

- Sphere-organized Tree
 - R-tree like balanced directory with overlapping regions
 - Use minimum bounding spheres (MBS) instead of MBRs



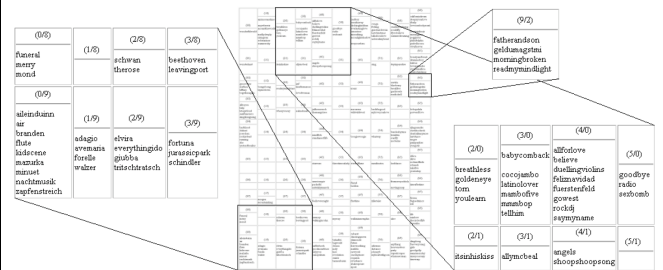
176

Support-Vector Machines



177

Self-Organizing Maps



178

Clustering Using Representatives (CURE)

- Middle path between single representative (centroid based) or full representative (complete linkage)
- Uses multiple representative points to evaluate the distance between clusters, adjusts well to arbitrary shaped clusters and avoids single-link effect
- Pick/label representatives carefully

179

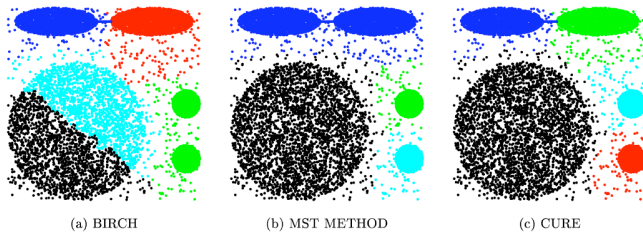
The CURE Method



- Draw random sample s .
- Partition sample to p partitions with size s/p
- Partially cluster partitions into s/pq clusters
- Eliminate outliers
 - By random sampling
 - If a cluster grows too slow, eliminate it.
- Cluster partial clusters.
- Label data in disk

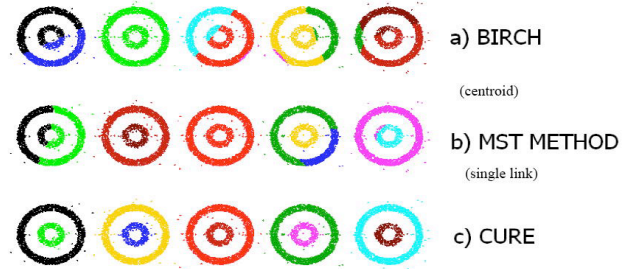
180

Clustering Results for Various Methods



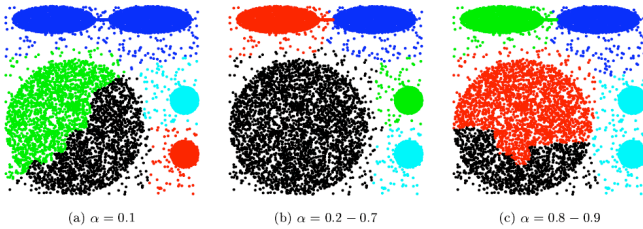
(a) BIRCH (b) MST METHOD (c) CURE

Experimental Results: CURE



Picture from *CURE*, Guha, Rastogi, Shim.

CURE's Alpha Factor



(a) $\alpha = 0.1$ (b) $\alpha = 0.2 - 0.7$ (c) $\alpha = 0.8 - 0.9$

Figure 9: Shrink factor toward centroid

Using CURE

The clusterer command is executed with command-line options to configure the clustering partition pools.

The options are:

- p [NUM] - Number of pre-clustering partitions (2 - 50)
- q [NUM] - Pre-clustering partition factor (3)
- k [NUM] - Desired number of clusters (4 - 50)
- c [NUM] - Representatives per cluster (2 - 15)
- a [NUM] - Representative scaling factor (0.2 - 0.7)

[OPTIONAL_FLAGS]:

- x [STR] - configuration file name [Configuration.txt]
- t - Displays the parsed command parameters and exits
- ? - Displays this message

Typical values (for a database of about 50000 entries):

```
fclusterer -p 20 -q 3 -k 20 -c 8 -a 0.4
```

The clusterer reads FMAK_Configuration.txt for DB table names and logs its progress verbosely, as in,

CURE Clusterer Execution

```
----- PRE CLUSTERING ROUND -----
Input Space Size : 5000
Partitions : 4
Partition Sizes : 1250
Clusters / Partition : 417

Cluster run #0 clustering 1250 data points...
Cluster run #0 found 541 representatives.
Updating representative data points to new status (541)
Updating non-representative data points status (709)
Cluster run #0 done.

Cluster run #1 clustering 1250 data points...
Cluster run #1 found 486 representatives.
Updating representative data points to new status (486)
Updating non-representative data points status (764)
Cluster run #1 done.

Cluster run #2 clustering 1250 data points...
Cluster run #2 found 480 representatives.
Updating representative data points to new status (480)
Updating non-representative data points status (770)
Cluster run #2 done.

Cluster run #3 clustering 1250 data points...
Cluster run #3 found 517 representatives.
Updating representative data points to new status (517)
Updating non-representative data points status (733)
Cluster run #3 done.

----- CLUSTERING FINAL REPRESENTATIVES -----
Final cluster run clustering 2024 data points...
Final cluster run found 74 representatives.
Number of Clusters constructed : 7024
Number of DataPoints constructed : 0
Number of DataPoints destroyed : 50927

Updating non-representative data points status(1950)
Marking Final cluster 1 with 1 reps
Marking Final cluster 2 with 10 reps
Marking Final cluster 3 with 10 reps
Marking Final cluster 4 with 1 reps
Marking Final cluster 5 with 1 reps
Marking Final cluster 6 with 3 reps
Marking Final cluster 7 with 4 reps
Marking Final cluster 8 with 1 reps
Marking Final cluster 9 with 4 reps
Marking Final cluster 10 with 10 reps
Marking Final cluster 11 with 10 reps
Marking Final cluster 12 with 1 reps
Marking Final cluster 13 with 1 reps
Marking Final cluster 14 with 1 reps
Marking Final cluster 15 with 1 reps
Marking Final cluster 16 with 1 reps
Marking Final cluster 17 with 1 reps
Marking Final cluster 18 with 1 reps
Marking Final cluster 19 with 10 reps
Marking Final cluster 20 with 2 reps
end! << Final cluster run done.

----- LABELING DB ITEMS -----
Reading cluster member table
Reading cluster representative table
RunLabelingRound on 4926 points with 74 cluster reps

----- POST-PROCESSING DB -----
Run post processing
----- CLUSTERING DONE -----
```

Post-Clustering Data Mining

When clustering is done, test with,

```
SELECT count(*) FROM FSongsR WHERE ClusteringStatus = 'F';
SELECT Artist,Album,Title FROM FSongsR WHERE ClusteringStatus = 'F';
SELECT genre,subgenre FROM FSongsR WHERE ClusteringStatus = 'F';
SELECT distinct clusterid FROM FSongsR WHERE ClusteringStatus = 'F';
```

Count clusters

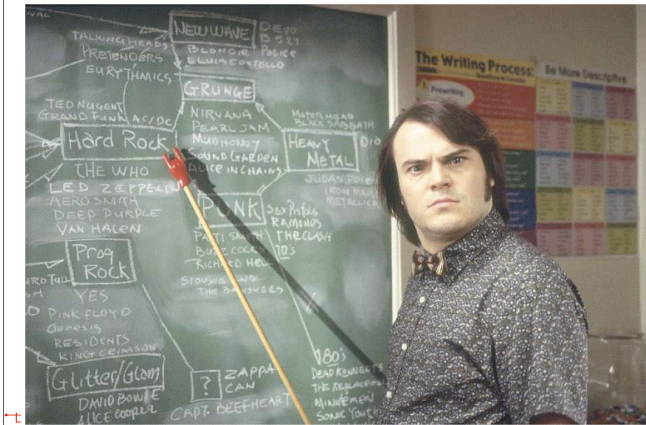
```
SELECT distinct ClusterID FROM FSongsR;
SELECT count(*) from FSongsR WHERE ClusterID =
SELECT Artist,Title FROM FSongsR WHERE ClusterID =
```

The following query will display the all the clusters and the size of each cluster

```
SELECT ClusterId, count(*) as ClusterSize
FROM FSongs
WHERE ClusterId is not null AND ClusteringStatus = 'F'
GROUP BY ClusterId;
```

Gather most common genre in cluster
Cluster genre histograms

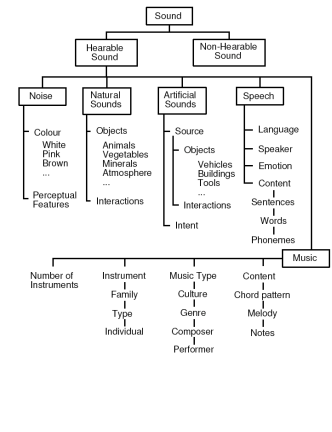
Music Classification



187

Classification

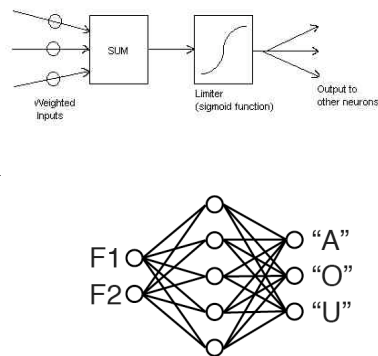
- Data clustering in high-dimensional non-linear spaces is complex
- Mapping discovered clusters to genres isn't obvious
- Rule-based techniques
- Easy for a small number of genres (< 10)
- Quite a challenge for many genres (> 50)
- Requires a large segmented feature vector



188

Neural Networks

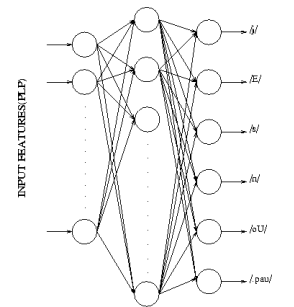
- Traditional 3-layer weighted NN model
 - Input
 - Hidden
 - Output
- Training by *back-propagation of error*
- Used for classification



189

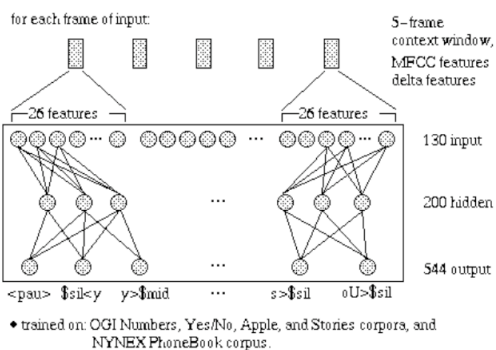
NN Training

- Given inputs and desired outputs,
- Iterate tuning neuron weights to distribute errors
- Relation to simulated annealing, weighted statistical networks, etc.



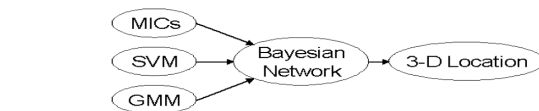
190

NN/NLP Example

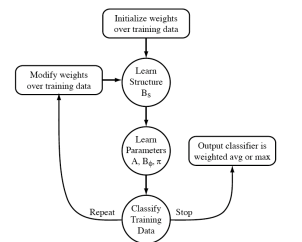


191

Bayesian (BIC) MAP Classifiers



- Trained statistical discriminators
- Statistics of covariance
- Bayesian information criterion (rule)



192

Code Exercises

- PCA for timbre spaces
- KNN or CURE clustering
 - MIDI Scores
 - Single sounds
 - Song data
- NNs for sound classification

MEDIA ARTS & TECHNOLOGY PROGRAM

199

199

Other Techniques

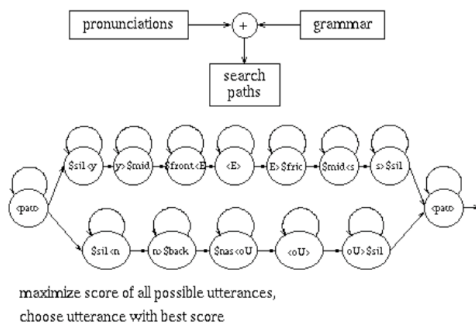
- For clustering
- For classification
- Search algorithms
- Noisy matching techniques
- Music/sound/speech grammars

MEDIA ARTS & TECHNOLOGY PROGRAM

200

200

Viterbi Search Algorithms

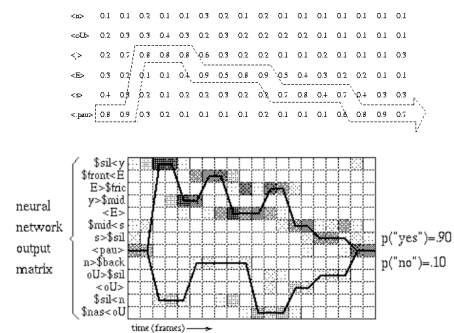


MEDIA ARTS & TECHNOLOGY PROGRAM

201

201

Viterbi Search Examples

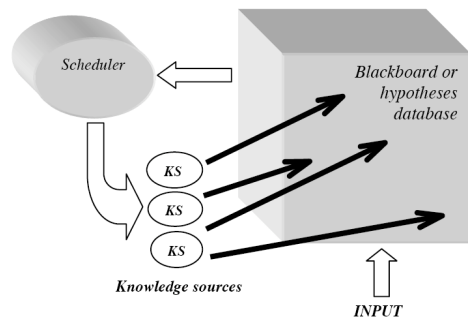


MEDIA ARTS & TECHNOLOGY PROGRAM

202

202

BlackBoard Systems

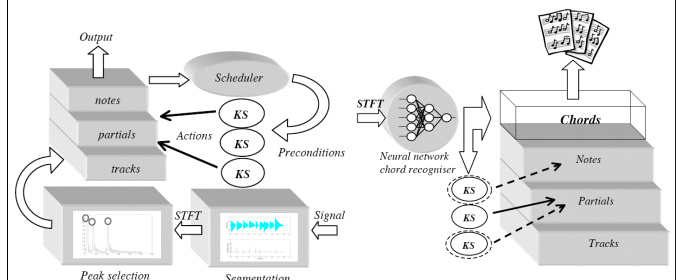


MEDIA ARTS & TECHNOLOGY PROGRAM

203

203

Bello's Examples



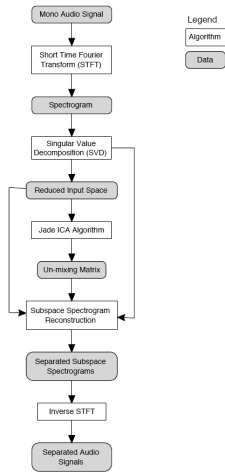
MEDIA ARTS & TECHNOLOGY PROGRAM

204

204

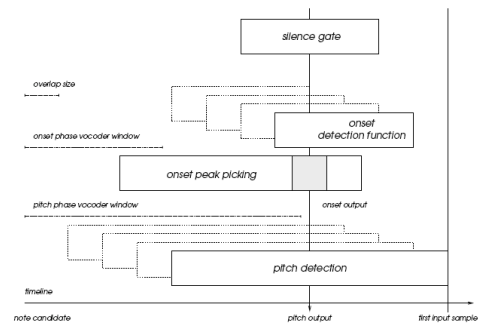
Multi-stage Systems (again)

- Incorporate statistical and heuristic methods (possibly using large data stores) after cross-domain multi-stage feature extraction



205

Windowed Multi-stage



206

Coding

- PCA/ICA
- Clustering
- NNets
- Calculating confusion matrices

207

Summary

- Data Reduction, Smoothing
- Princ./Indep. Component Analysis
- Audio Segmentation and Musical Form
- Grouping, Clustering and Classification

208

What I Really Want ...

Applications



209

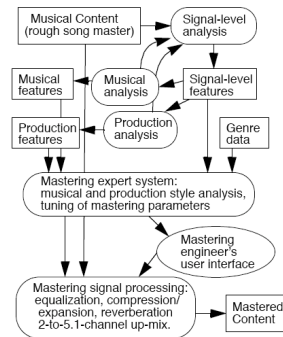
Topic 4: Databases & Applications

- MIR Databases
 - Handling of Large or Dynamic Feature Vectors
- Application Requirements and Design
 - Searching, Indexing, and Playlist generation
 - Audio Summarization and Thumb-nailing
 - Content Matching and Finger-printing
 - Data Clustering and Genre Classification
 - Other Applications: Mapping Systems

210

In the Reader

- Accompaniment
- Transcription
- Segmentation
- Classification
- Instrument ID
- Mapping experts
- Fingerprinting



211

Code Exercises

- “Real world” applications using FMAK, MARSYAS, aubio, libxtract, etc. (extending sing_along)
- Search
 - Query by humming
 - Recommender (preference matching)
- Players
 - Search and indexing
 - Playlist generation

212

Application Flow, System Integration

- Normally a given (based on user requirements)
- Special case: general-purpose MDB design or MDB frameworks (e.g., Marsyas, FASTLab)
- Analysis engine architecture
- Query generation & on-the-fly analysis
- Result pruning and browsing

213

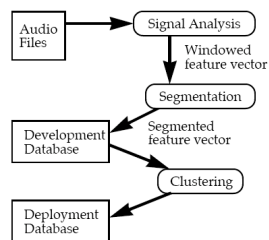
Application Design

- Source content and metadata formats
- Low-level features and signal analysis
- High-level derived features and statistics
- Perceptual mapping and data reduction
- Segmentation and feature vector pruning
- Back-end database issues

214

Handling of Large or Dynamic Feature Vectors

- FV segmentation and pruning
- Application-required data selection
- Analysis-on-demand
- DB winnowing, PCA



215

Application Requirements and Design

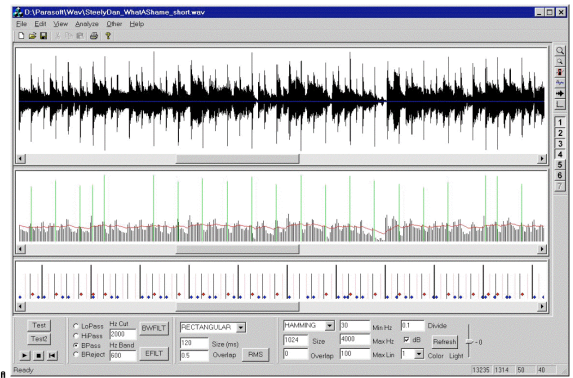
- Query criteria
- Clustering dimensions
- Fingerprint keys
- Requirements of players, mappers
- Flavors of clustering/classification
- Enhanced tools and classification

216

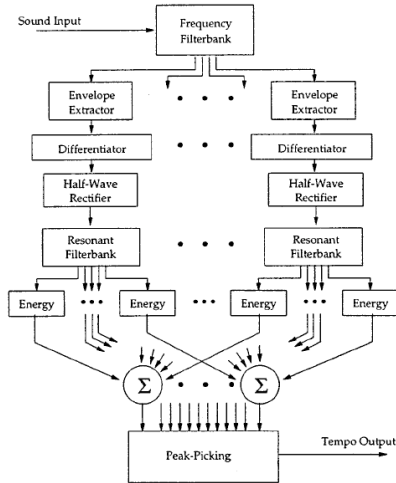
Beat Detection and Tempo Analysis

- Given note-on detector:
 - Use a peak detector
 - Use autocorrelation to find regularity
 - Use context and heuristics to get from strong beat to the "1"
- Adaptive/real-time foot-tappers
- Automatic accompaniment

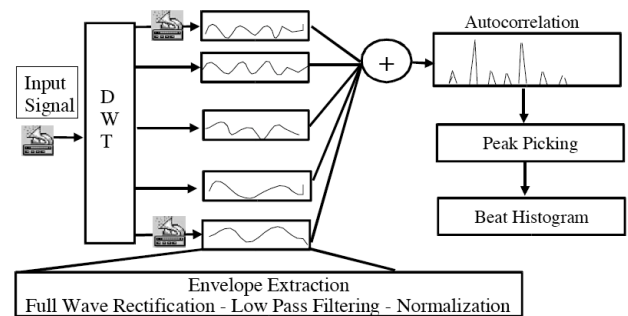
Hierarchical Tempo Extraction



Schreier's Tempo Follower



Self-similarity for Beat Extraction



Meter Estimation Systems

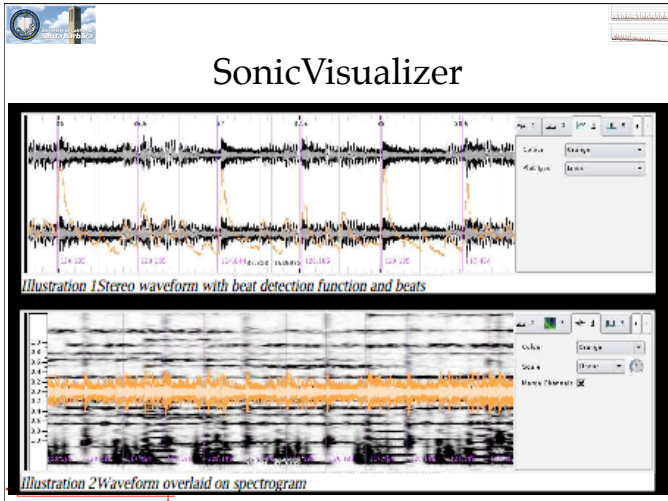
Table 1. Characteristics of some meter estimation systems.

Reference	Input	Output	Technique
Temperley & Sleator (1999)	MIDI	Meter, time quantization	Rule-based approach; implementation of the preference rules in (Lerdahl et al., 1983)
Dixon (2001)	MIDI, audio	Tactus	First find periods using IOI histogram, then phases using multiple agents
Raphael (2001)	MIDI, audio	Tactus, time quantization	Probabilistic generative model for onset times; MAP estimation (Viterbi)
Cemgil & Kappen (2003)	MIDI	Tactus, time quantization	Probabilistic generative model for onset times; sequential Monte Carlo methods
Goto & Murakawa (1995, 1997)	Audio	Meter	Extract onset components; IOI histogram; multiple tracking agents
Scheirer (1998)	Audio	Tactus	Bank of comb filters to analyze periodicity of power envelopes at six subbands
Laroche (2001)	Audio	Tactus, swing	Extract discrete onsets; maximum-likelihood estimation
Sethares & Staley (2001)	Audio	Meter	Calculate RMS-energies at 1/3-octave subbands; apply a periodicity transform
Gouyon et al. (2002)	Audio	Tatum	First find periods (IOI histogram), then phases by matching isochronous pattern
Klapuri et al. (to appear)	Audio	Meter	Measure degree of accentuation; bank of comb filters; probabilistic model

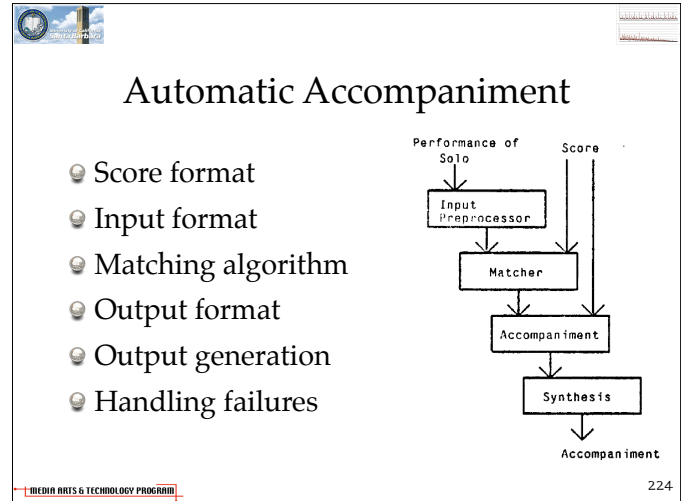
Aubio NoteOnset Detection

```
// run pvoc & rms onset detector
aubio_pvoc_do (pv,ibuf, fftgrain);
aubio_onsetdetection(o,fftgrain, onset);

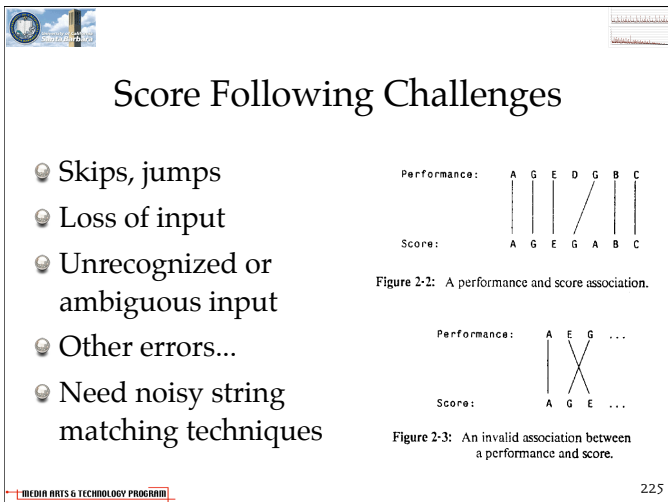
// moving mean adaptive thresh. peak pick
isonset = aubio_peakpick_pimrt(onset,parms);
if (isonset) { // if at note onset
  if (aubio_silence_detection(ibuf, threshold2) == 1)
    isonset=0;
  else // play a sound on beat onset
    for (pos = 0; pos < overlap_size; pos++)
      obuf->data[0][pos] = woodblock->data[0][pos];
}
```



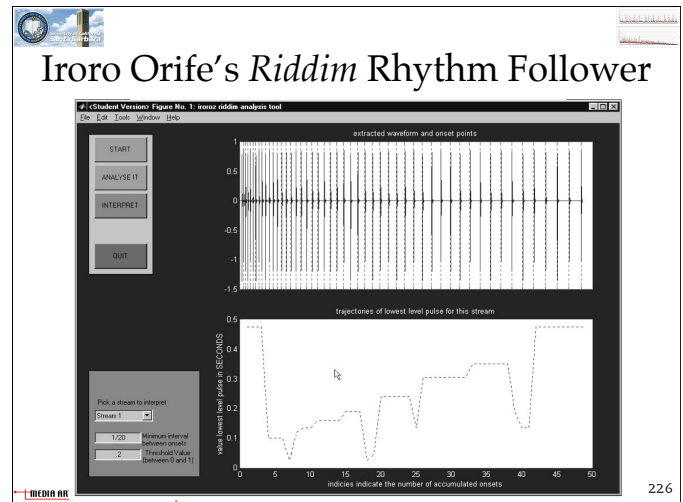
223



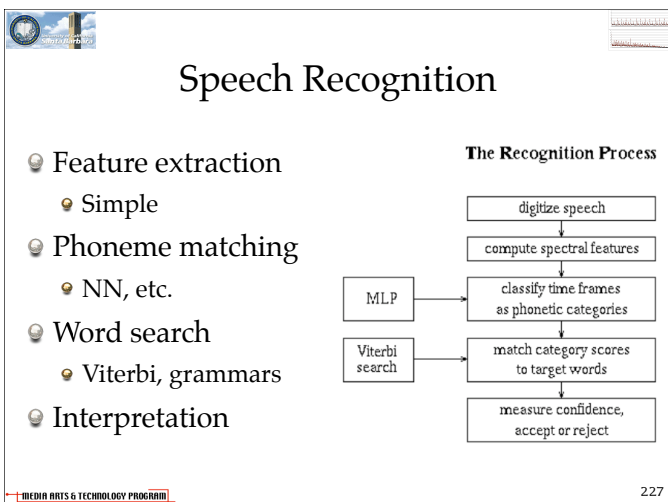
224



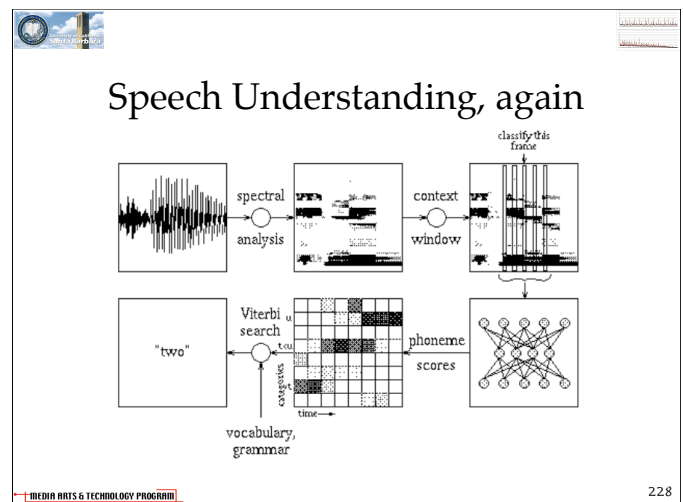
225



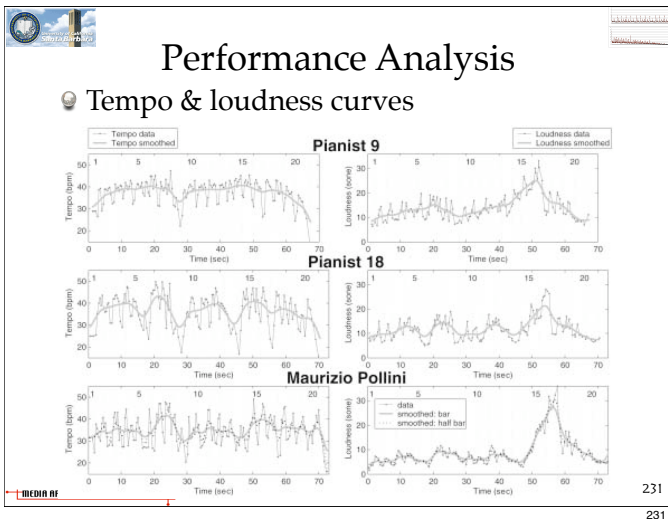
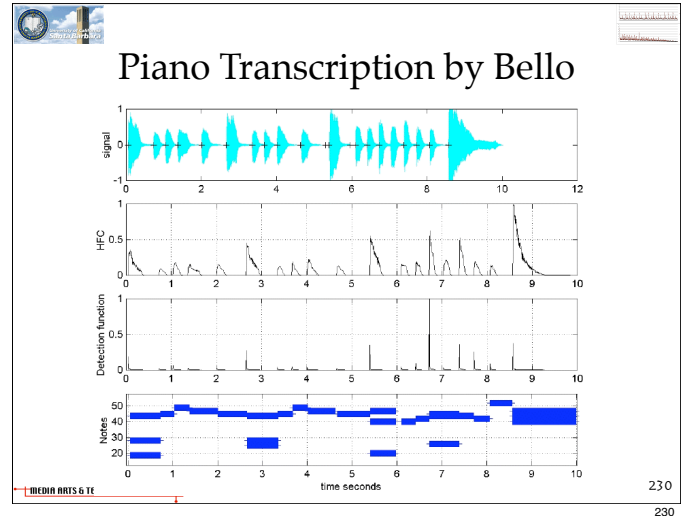
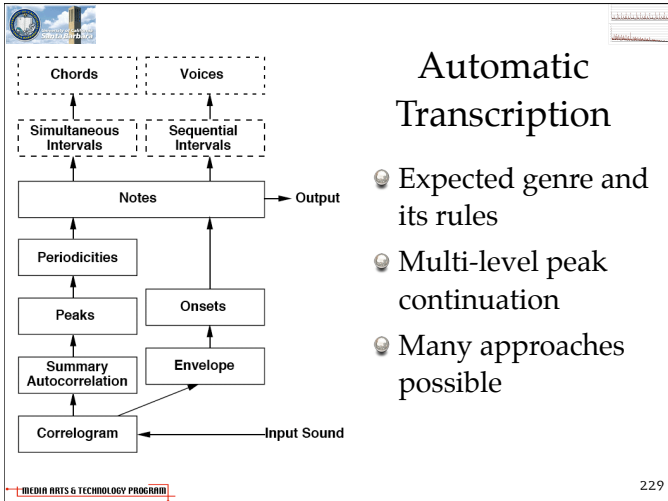
226



227



228



- ### Performance Rule Extraction
- Given a set of performance rules
 - Lerdahl & Jackendoff
 - Segment and analyze performance into tempo-phrasing functions
 - Search for a weighting of the rules to match the performance
 - Learn a performer's style

- ### Code Exercises
- Mapping systems
 - Accompaniment
 - Processing
 - Annotation, transcription
 - Metadata extraction
 - Speech understanding, music transcription
 - Extended tools
 - Smart editing?

- ### Searching, Indexing, and Players
- The most common MDB application!
 - Players often suffer from poor (or no) derived metadata or sophisticated indexing
 - Some recent systems are changing this...
 - There are many hard issues
-
- The screenshot shows the moodlogic application interface. It features a sidebar with a 'View' menu (My Music, MP3 Player, Messages, Instant Mix, Mellow Mix, 70s Disco Party, Classic Rock, Romantic 80s, Like Chemical Bros) and a main area with 'Genre' and 'Artist' filters. The 'Genre' filter is set to 'Aggressive | Upbeat | Happy | Romantic | Mellow | Sad'. The 'Artist' filter is set to 'Blonde - Atonic', 'Blonde - Dreaming', and 'Blonde - Hanging On The Telephone'. A 'mix' button is visible at the bottom right.

Pandora



235

235

Player Requirements

- Preference-matching recommender systems (my personal #1 wish) (worth \$)
- Automatic playlist generation (see above)
- Both require a powerful feature vector, genre classifier (possibly unlabeled), and user preference statistics (good style distance metrics)

236

236

Search, Match, MIR

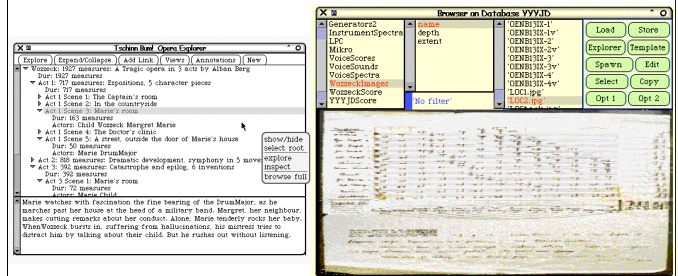
- Search in music databases based on text feature vectors is common (GraceNote, on-line stores)
- "Query by humming" in melody databases depends on melody extraction (batch and run-time)
- Timbral similarity measures are found (MuscleFish system)

237

237

Searching Musical Structures

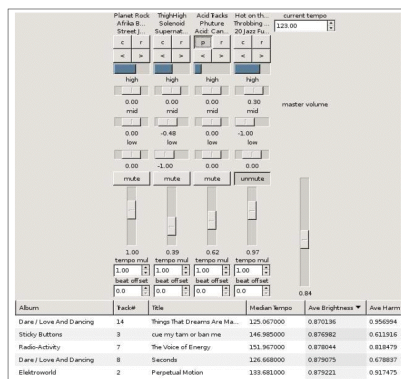
- The Siren opera browser for *Wozzeck*



238

238

DataJockey, A. Norman

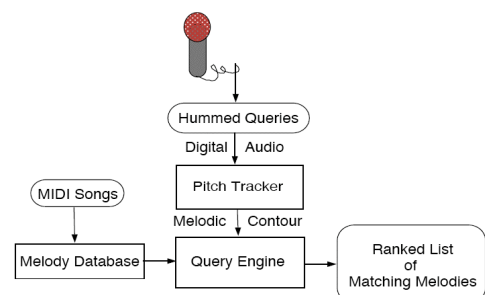


239

239

Query-by-Humming

- Limited examples abound



240

240

QbH Framework

- Input analysis
 - Thresholding, note-onset detection, pitch tracking
- Generation of search string / vector
 - Approx. rhythmic pattern
 - Interval direction / distance
- Noisy / fuzzy string matching techniques

241

241

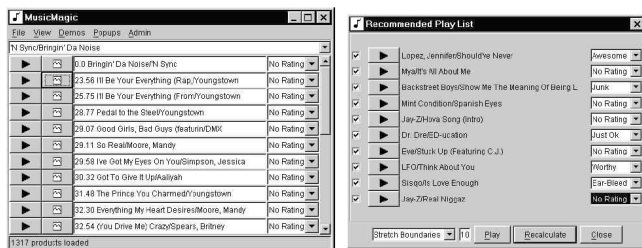
Matching Requirements

- As above, these generally require
 - Large (segmented and pruned) feature vectors
 - Support utilities (clusterer, classifier, run-time analysis for queries)
 - Run-time search on complex feature set
 - Possible pruning of large return data sets

242

242

MusicMagic Playlists



243

243

FoafingtheMusic

(<http://foafing-the-music.iaa.upf.edu>)

- LiveJournal, Weblogger, Friend of a Friend (FOAF) profile
- <http://www.last.fm>, AudioScrobbler, music lists
- WebJay for performance
- Generates playlists, plays...

Getting music recommendations from your FOAF profile
Articles found: [Bugs & Anwar](#), [Social Distortion](#), [The Misfits](#), [The Pogues](#)

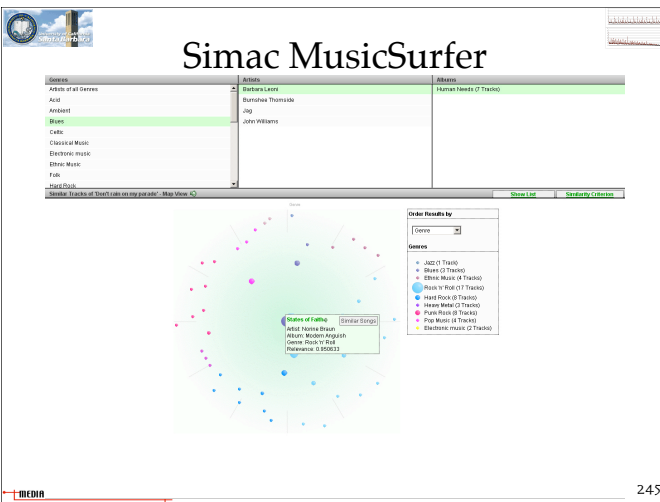
Some recommendations that you might like...

- The Bloody Irish Boys - Columbia, Dead Kennedys (1970,1980)
- Fear (1970,1980,1990,2000)
- Black Flag - Sire Records, US (1970,1980)
- The Circle Jerks (1980,1990)
- The Cramps - Sire Records, US (1980,1990,2000)
- Joe Strummer - Atlantic, US (1980,1990,2000)
- The Dragons (1990,2000)
- The Dictators (1970,1980,2000)
- The Germs (1970,1980)
- The Grays CA - Columbia, Jello Biafra US (1980,1990,2000)
- Demented Are Us (1980,1990,2000)
- Automatic 7 (1990,2000)
- The Stranglers (1970,1980,1990,2000)
- The Godfathers - London, (1980,1990)
- Frankie Flattops (1990)
- Chemical People (1980,1990)
- Clearview77 - RoughTrade

244

244

Simac MusicSurfer



245

245

Simac Matching Criteria

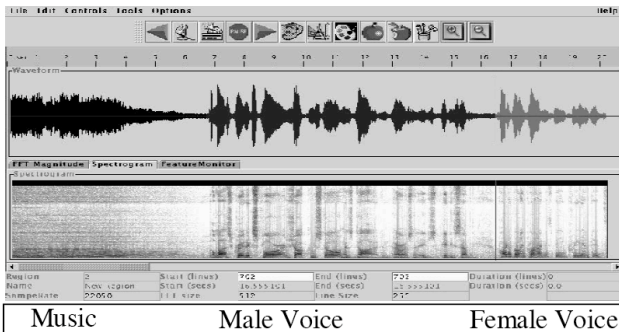
- Timbre
- Rhythm
- Genre Probability
- Danceability
- Dynamic Complexity
- BPM
- Meter
- Tonality Key (C, C#, D, D..) (Major / Minor), Tonal Strength

246

246

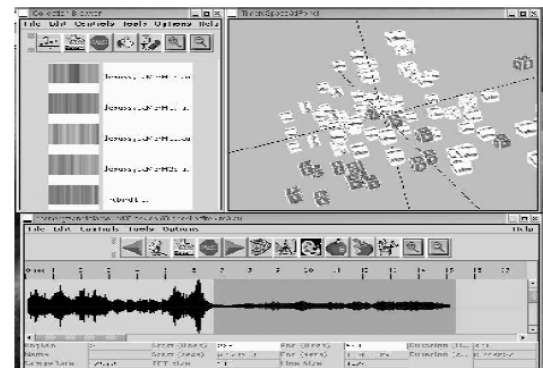
Content-Aware Tools

- What would you want a “smart editor” to do?



247

TimbreSpace Browser (GT, PRC)



248

Audio Summarization and Thumb-nailing

- Audio thumb-nailing assumes good segmentation, and location of the most typical short segment
 - Choose most-frequently repeated section (is that appropriate?)
 - Choose first incidence of common section
 - Some applications require a single note or a common signal window?

249

Content Matching, Finger-printing

- Important for indexing and stream-following (e.g., segmentation of video)
- Genre classification (speech, music, effect) useful for segmentation and indexing
- Exact finger-printing for digital rights management (some systems in use)
- Still difficult to avoid false positives and not be easily spoofed

250

Fingerprinting Systems

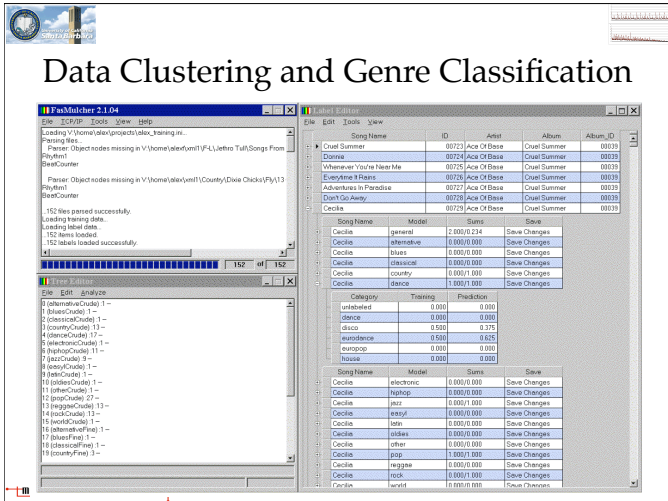
- Is there a smaller-dimension unique identifier that's robust in the face of processing, editing, and even spoofing?
- Should it be based on:
 - Segmentation -- what about edited versions?
 - Averaged timbre of a selected section?
 - Spectral variety measures?
 - Voice or solo instrument signature?

251

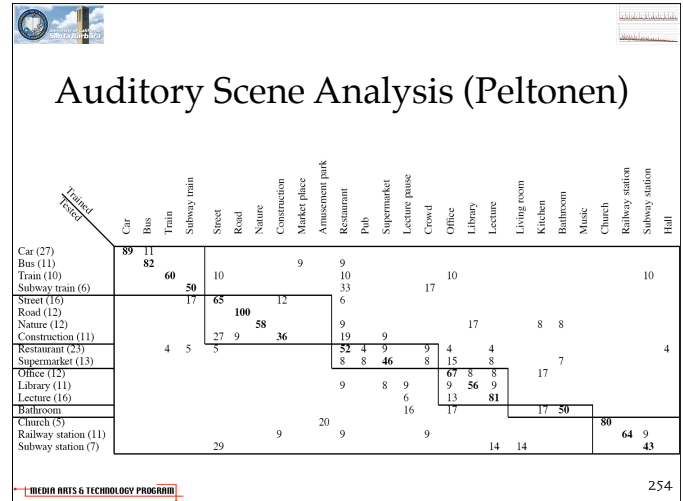
Music Thumbnail Examples

- Mark Sandler, Queen's, London

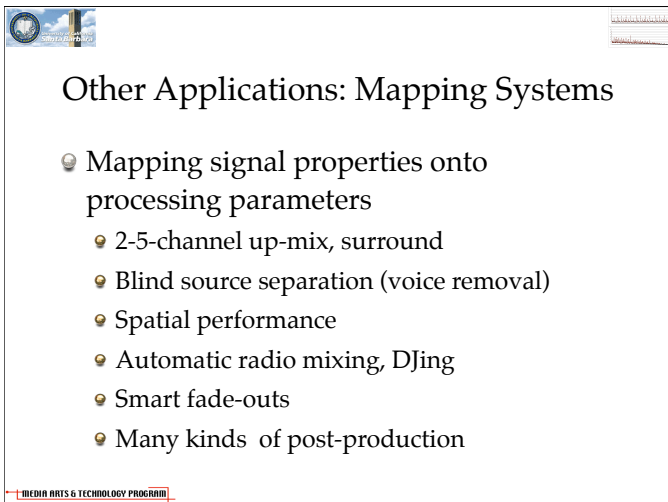
252



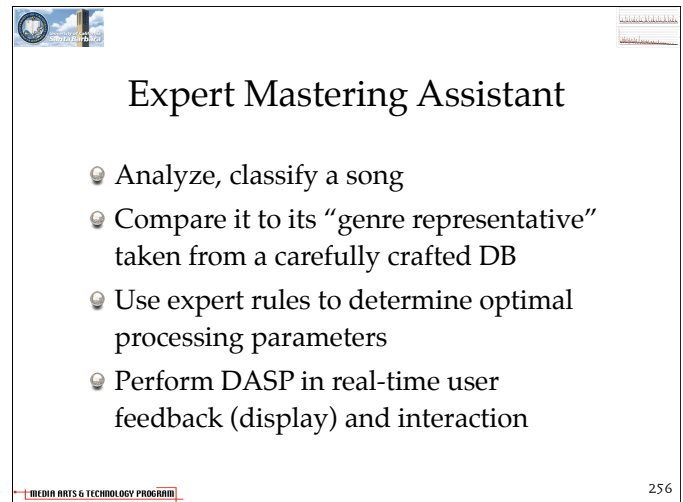
253



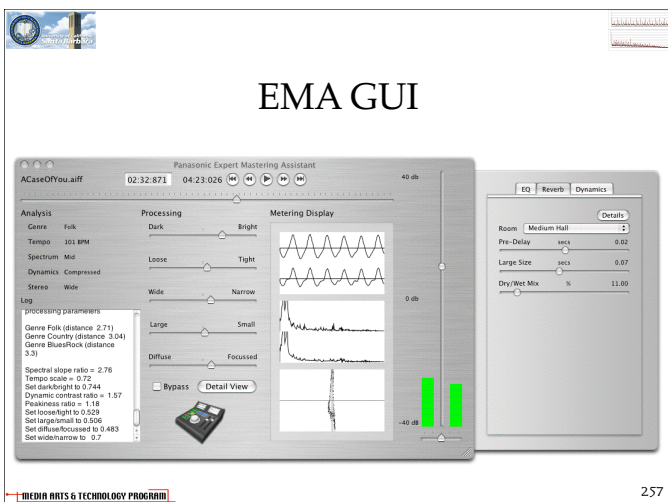
254



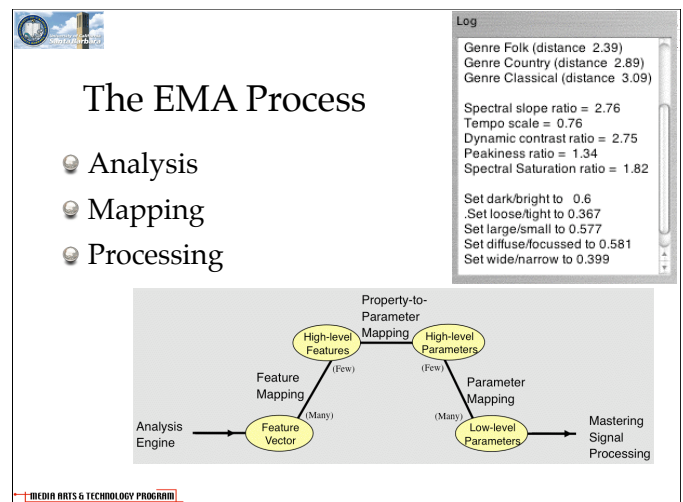
255



256



257



258

EMA Processing Examples

Title	Fortuna Imperatrix	Scudetti Allegro	Take Five	A Case of You	My Sweet Lord	El Cucarro	Drunken Hearted	Back In The USSR	Heiss
Artist	Mundi						Man	Rock	
	Carl Orff	Horevitz	Dave Brubeck	Joni Mitchell	George Harrison	Buena Vista Social Club	Robert Johnson	The Beatles	Nana Hagen
Genre 1	Classical (2.1)	Classical (2.26)	Jazz (4.49)	Folk (2.39)	Classical (4.11)	Rock/Pop (2.11)	Opera (10.8)	Rock/Pop (3.76)	Rock (2.44)
Genre 2	Rock (2.25)	Other (2.44)	Other (6.12)	Country (2.89)	Other (5.04)	Rock/Pop (2.96)	Classical (10.9)	Rock (4.2)	Rock/Pop (2.57)
Genre 3	Classical (2.32)	Classical (2.82)	Classical (6.76)	Classical (0.09)	Rock (3.36)	Other (3.08)	Rap/R&B (11.8)	Punk (4.54)	Classical (2.82)
Detailed Analysis Properties									
Spect Slope	2.33	1.71	5.09	2.76	2.07	2.56	1.83	2.06	3.79
TempoScale	0.354	0.522	0.344	0.76	0.92	0.804	0.803	0.795	0.496
DynContrast	3.31	2.38	6.59	2.75	1.22	4.71	29.4	1.77	1.29
Peakiness	1.61	1.35	1.83	1.34	1.86	0.934	1.43	3.64	1.06
Spect. Saturation	1.53	1.32	2.49	1.82	1.15	1.48	0.961	1.71	2.79
High-level Processing Parameters									
Dark/bright	0.464	0.156	0.6	0.6	0.335	0.578	0.215	0.328	0.6
Loose/tight	0.361	0.357	0.454	0.367	0.313	0.487	0.563	0.337	0.316
Large/small	0.905	0.449	0.682	0.577	0.397	0.486	0.263	0.56	0.753
Diffuse/focused	0.531	0.451	0.737	0.561	0.426	0.461	0.312	0.675	0.722
Wide/narrow	0.3	0.7	0.363	0.399	0.544	0.3	0.3	0.639	0.3

259

259

Other Application Genres

See initial survey papers

260

260

Summary

- Application requirements and architecture
- Application domains
 - Search-as-application
 - Players, index, search, match
 - Accompaniment
 - Transcription
 - Mappers

261

261

Review, Advice

- Signal Analysis
 - Basic data analysis (RMS, FFT) is fast and simple
 - Advanced analysis techniques can be either expensive (TDD) or hard to interpret (Wavelets)
 - Some measures are inexpensive and powerful
 - MFCC, beat histograms
 - There are still advanced to be made...

262

262

Review 2

- Deriving higher-level features is complex
 - Smoothing, de-noising
 - Perceptual mapping
 - Segmentation
 - Feature vector statistics
- Feature vector design and weighting
- Multi-stage systems
- Applications
- Code review

263

263

Tools & Techniques

- Adaptive numerical techniques
- Dimensionality reduction and clustering
- Statistics and searching
- Many assumptions about musical form and timbres are not simple to codify, calling for heuristic programming techniques
 - Neural nets
 - Machine learning

264

264

Application Summary

- MIR for Query
 - Access, QBH
 - Finger-print, content ID
 - Playlist generation
- MIR for Indexing
 - Thumb-nailing, summarization
 - Video segmentation
- MIR for Score Following
 - Performance, mix-sync
- Other Apps galore!
 - Invisible metadata-in-media
 - MIR-based services

265

Observations

- Good musical segmentation and pruned feature vectors are key to the next level of MDB functionality
- Clustering and classification are easier given a sufficiently sophisticated feature vector
- Many applications still use quite shallow and simplistic feature extraction...

266

Where do we go from here?

- MAT 240 A-F overview
- ICMC Slide of MAT 240 spin-offs
- AlloSphere sound
- Performance systems
- DASP frameworks
- New languages
- A trip down memory lane

267

MAT 240 Results

- MA/MS Thesis Projects
 - Ramakrishnan¹, Sturm¹, McCoy², Hollerweger¹, Putnam¹, Wakefield¹, Newman¹, Smith¹, Norman, Stuckey², Medrano², Thompson¹, Hosale, Bell¹, Gujar², Voss², Schlegel², Avery², Koven², Ku², O'Reilly¹, Yeo¹, Brown², Thomas², Thall², Durrand², Romblo², Wolcott² (¹ = Academia ² = Industry)
- PhD Candidates
 - Overholt, Sturm, Wakefield, Smith, Newman, Putnam, Hosale, Bell, ...
- CSL, Siren, CRAM, AlloSphere

268

MAT 240 Creation

- Pieces
- Performances
- Videos
- Controllers
- Installations
- Publications

269



MEDIA ARTS & TECHNOLOGY PROGRAM

Two years ago, right here

**MAT 240A:
DIGITAL AUDIO
PROGRAMMING:
THE SERIES,
EPISODE 1, PART 1**

PRESENTATION SLIDES BY STEPHEN TRAVIS POPE,
STP@MAT.UCSB.EDU

270

Digital Audio Programming Topics

- Sound/music software
 - (Composition), synthesis, processing, performance
 - Sources, filters, analysis/resynthesis
 - Feature extraction, databases
- Interactive systems and controllers
- Multimedia programming
- Multi-threaded & distributed systems
- Media data streaming and I/O

271

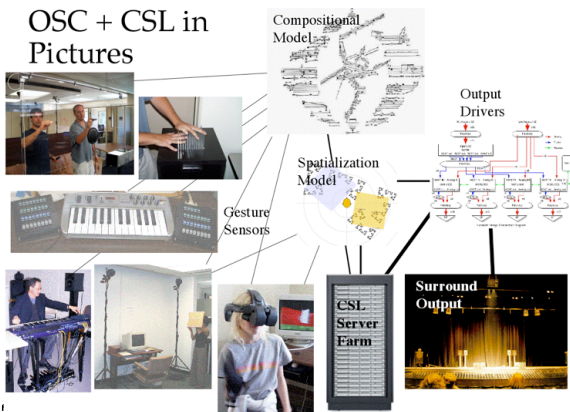
The MAT 240 Digital Audio Programming Series

- 240A: Sound I/O APIs
- 240B: Spectral transformations
- 240C: Spatial and surround sound
- 240D: Sound synthesis techniques
- 240E: Control and multi-rate processing
- 240F: Databases & Music Information Retrieval

272

Why MAT 240?

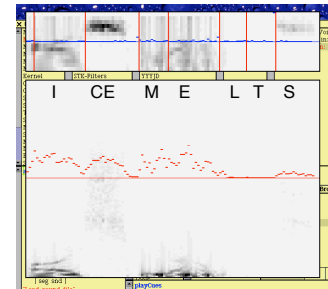
OSC + CSL in Pictures



273

MAT 240F: Digital Audio Programming: Audio Analysis and Music Information Retrieval

Stephen T. Pope
stephen@mat.ucsb.edu
Spring, 2008



274