Chapter 21. Meeting 21, Languages: Synthesis with Code

21.1. Announcements

- Music Technology Case Study Final due Today, 24 November
- Sonic System Project Report due Thursday, 3 December
- Last quiz: Thursday, 3 December

21.2. Quiz Review

- ?

21.3. Modern Music-N

- 1985: Csound
- Command-line program
- Currently available for many platforms, with many interfaces
- http://www.csounds.com

21.4. Csound: Score, Orchestra, and CSD Files

- Orchestra (a .orc file) defines synthesis processing and interconnections (instruments)
- Score (a .sco file) defines control information (events and parameters)
- A CSD file combines the score and orchestra into a single file delimited by XML-style markup
  - Outermost: <CsoundSynthesizer> </CsoundSynthesizer>
  - Orchestra: <CsInstruments> </CsInstruments>
  - Score: <CsScore> </CsScore>

21.5. Csound: Orchestra Syntax: Instrument Blocks

- A quasi programming language, closer in way to an assembly language

Orchestra procedures can be extended with Python (Ariza 2008)

- Consists of statements, functions, and opcodes
- Opcodes are unit generators
- Comments: start with a semicolon
- Instruments in blocks, named with a number:
    - Start marker: “instr” and a number
    - End marker: “endin”
    - Trivial example instrument definition

```
instr 100
   ; comments here!
endin
```

21.6. Csound: Orchestra Syntax: Signals

- Signals are created and interconnected (patched) within instrument blocks
- Signals: cary streams of amplitude values as numbers within the dynamic range (16 bit audio uses integers from -32768 to 32768)
- Signal paths (like patch cords) are named variables
- Signal variables can be at different rate resolutions depending on the first letter of the variable name
    - a-rate: Audio, name starts with an “a” (e.g. aNoise)
    - k-rate: Control signals, name starts with an “k” (e.g. kEnvl)
    - i-rate: Initialization values, name starts with an “i” (e.g. iFq)
    - Example

```
aNoise   random   -12000, 12000
```

- Opcodes: unit generators
    - Syntax uses spaces and commas to delineate:

      Provide variable, opcode, and space-separated parameter arguments

```
destinationSignal  opcodeName   arg1, arg2, ...
```

    - Example: “random”; takes two arguments: min and max
    - Example: “outs”: takes two arguments: two signals
- Example

```
instr 100
    aNoise   random   -12000, 12000
             outs     aNoise, aNoise
endin
```

21.7. Csound: Score Syntax

- A list of events and parameters given to instruments in the orchestra
- Provide a space-separated list of at least three values on each line:

```
i  instrumentNumber  startTime  duration  p4  p5  ...
i  instrumentNumber  startTime  duration  p4  p5  ...
i  instrumentNumber  startTime  duration  p4  p5  ...
```

- Additional parameters (called p-fields) can be added after duration and provided to the instrument in the score
- Example: Two events for instrument 23 lasting two seconds, starting at 0 and 5 seconds

```
i  23  0.0  2.0
i  23  5.0  2.0
```

21.8. Csound: Rendering an Audio File

- Call the CSD file with the csound command-line application on the CSD file to render audio

```
csound -d -A noise.csd -o out.aif
```

- Provide a “flag” to indicate type of audio output
    - -A (aiff output)
    - -W (wave output)
- Give sampling rate, control rate, and number of channels in a header

```
sr     = 44100
kr     = 4410
ksmps  = 10
nchnls = 2
```

- Example: a noise instrument
- Example: tutorial-a-01.csd

```
<CsoundSynthesizer>
<CsInstruments>
sr     = 44100
ksmps  = 10
nchnls = 2

instr 100
   aNoise   random  -12000, 12000
            outs    aNoise, aNoise
endin

</CsInstruments>
<CsScore>
i  100  0    2
i  100  3    2
i  100  6    2
</CsScore>
</CsoundSynthesizer>
```

21.9. Csound: GEN Routines and Wave Tables

- Some opcodes require a wave table identification number as an argument
- Wave tables are created with GEN routines in the score file, before events are listed
- Example: a GEN routine used to create a 16384 point sine wave as a wave table

```
f  99  0  16384  10  1
```

- Oscillators require a wave table to provide a shape to oscillate

  The oscili opcode oscillates (and interpolates) any shape given in the f-table argument

```
aSrc    oscili      amplitude, frequency, functionTable
```

- Example: two instruments, a noise and a sine instrument
- Example: tutorial-a-02.csd

```
<CsoundSynthesizer>
<CsInstruments>
sr     = 44100
ksmps  = 10
nchnls = 2

instr 100
   aNoise   random      -12000, 12000
            outs        aNoise, aNoise
endin

instr 101
   aSine    oscili      12000, 800, 99
            outs        aSine, aSine
endin

</CsInstruments>
<CsScore>
f  99   0    16384  10  1
```

```
i  100  0    2
i  100  3    2
i  100  6    2

i  101  2    6
</CsScore>
</CsoundSynthesizer>
```

21.10. Csound: Scaling and Shifting Signals

- Example: using a scaled sine wave as an envelope of noise
- Assignment (=) and operators (+, *) permit mixing and scaling signals
- Example: tutorial-a-03.csd

```
<CsoundSynthesizer>
<CsInstruments>
sr     = 44100
ksmps  = 10
nchnls = 2

instr 102

   aEnvl    oscili      .5, 6.85, 99
   aEnvl    = aEnvl + .5


   aNoise   random      -12000, 12000
   aNoise   = aNoise * aEnvl
            outs        aNoise, aNoise
endin
</CsInstruments>
<CsScore>
f  99   0    16384  10  1
i  102  0    2
i  102  3    2
</CsScore>
</CsoundSynthesizer>
```

21.11. Csound: Adding Parameters to Score and Orchestra

- pN (p1, p2, p3, p4, ...) variables in orchestra permit additional parameter values to be provided from the score to the instrument
- Design of instruments in the orchestra requires choosing what parameters are exposed in the score
- Example: tutorial-a-04.csd

```
<CsoundSynthesizer>
<CsInstruments>
sr     = 44100
ksmps  = 10
nchnls = 2
```

```
instr 102

   iDur = p3
   iTrem = p4

   aEnvl    oscili      .5, iTrem, 99
   aEnvl    = aEnvl + .5

   aNoise   random      -12000, 12000
   aNoise   = aNoise * aEnvl
            outs        aNoise, aNoise
endin
</CsInstruments>
<CsScore>
f  99   0    16384  10  1
i  102  0    2   6.2  ; fourth parameter is frequency of sine envelope
i  102  3    2   23
i  102  6    2   45.6
</CsScore>
</CsoundSynthesizer>
```

21.12. Csound: Adding Filters

- Numerous opcodes exist to explore a wide range of common synthesis tools
- Low pass filter

```
aDst    lowpass2     aSrc, cutoffFrequency, resonance
```

- Can create a control signal to adjust a lowpass filter cutoff frequency, and applying that lowpass filter to noise
- Example: tutorial-a-05.csd

```
<CsoundSynthesizer>
<CsInstruments>
sr     = 44100
ksmps  = 10
nchnls = 2

instr 102
   iDur = p3
   iTrem = p4
   iFilterRate = p5

   aEnvl    oscili      .5, iTrem, 99
   aEnvl    = aEnvl + .5

   aNoise   random      -12000, 12000
   aNoise   = aNoise * aEnvl

   kCutoff  oscili      .5, iFilterRate, 99
   kCutoff  = kCutoff + .5
   kCutoff  = kCutoff * 8000 + 900

   aPost    lowpass2    aNoise, kCutoff, .85
            outs        aPost, aPost
endin
</CsInstruments>
<CsScore>
```

```
f  99   0    16384  10  1
i  102  0    2   6.2   .85
i  102  3    2   23    .65
i  102  6    2   45.6  .50
</CsScore>
</CsoundSynthesizer>
```

21.13. Csound: A Classic Synthesizer

- A class subtractive synth sound with detuned oscillators, an LPF with modulated cutoff, and an ADSR envelope

- Voltage controlled oscillator (vco): anlogue modelled digital oscillator

```
aOsc    vco     amp, cps, waveShape, pulseWidth, functionTable
```

- ADSR envelope

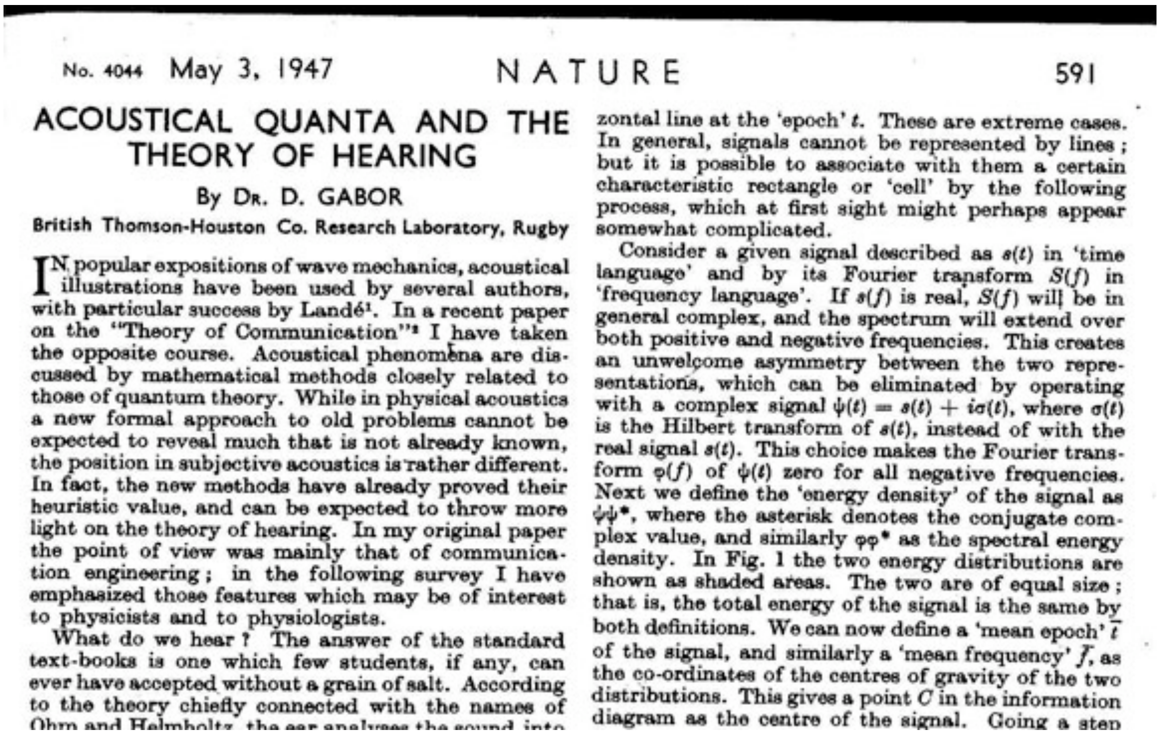
```
kEnvel    adsr     attack, decay, sustainLevel, release
```

- Example: tutorial-a-06.csd

```
<CsoundSynthesizer>
<CsInstruments>
sr     = 44100
ksmps  = 10
nchnls = 2

instr 103

   iDur = p3
   iAmp  = ampdbfs(p4)
   iPitch = cpsmidinn(p5)
   iFilterRate = p6

   kCutoff  oscili      .5, iFilterRate, 99
   kCutoff  = kCutoff + .5
   kCutoff  = kCutoff * 4000 + 400

   aOscA    vco         iAmp, iPitch, 2, .5, 99
   aOscB    vco         iAmp, iPitch*.499, 1, .5, 99

   aPost    lowpass2   aOscA+aOscB, kCutoff, 1.2
   kEnvl    adsr      .1*iDur, .2*iDur, .8, .2*iDur
            outs        aPost*kEnvl, aPost*kEnvl
endin
</CsInstruments>
<CsScore>
f  99   0    16384  10  1
i  103  0    2   -12    52  .5
i  103  3    2   -18    51  1.2
i  103  6    4   -24    48  3
</CsScore>
</CsoundSynthesizer>
```

21.14. Granular Synthesis: History

- Isaac Beekman (1588-1637): 1616: corpuscular theory of sound: sound cuts air
- 1947: Gabor proposes acoustical quanta: like photons for sound (1947)



No. 4044 May 3, 1947 NATURE 591

ACOUSTICAL QUANTA AND THE THEORY OF HEARING

By Dr. D. GABOR

British Thomson-Houston Co. Research Laboratory, Rugby

IN popular expositions of wave mechanics, acoustical illustrations have been used by several authors, with particular success by Landé[1]. In a recent paper on the "Theory of Communication"[2] I have taken the opposite course. Acoustical phenomena are discussed by mathematical methods closely related to those of quantum theory. While in physical acoustics a new formal approach to old problems cannot be expected to reveal much that is not already known, the position in subjective acoustics is rather different. In fact, the new methods have already proved their heuristic value, and can be expected to throw more light on the theory of hearing. In my original paper the point of view was mainly that of communication engineering; in the following survey I have emphasized those features which may be of interest to physicists and to physiologists.

What do we hear? The answer of the standard text-books is one which few students, if any, can ever have accepted without a grain of salt. According to the theory chiefly connected with the names of Ohm and Helmholtz, the ear analyses the sound into

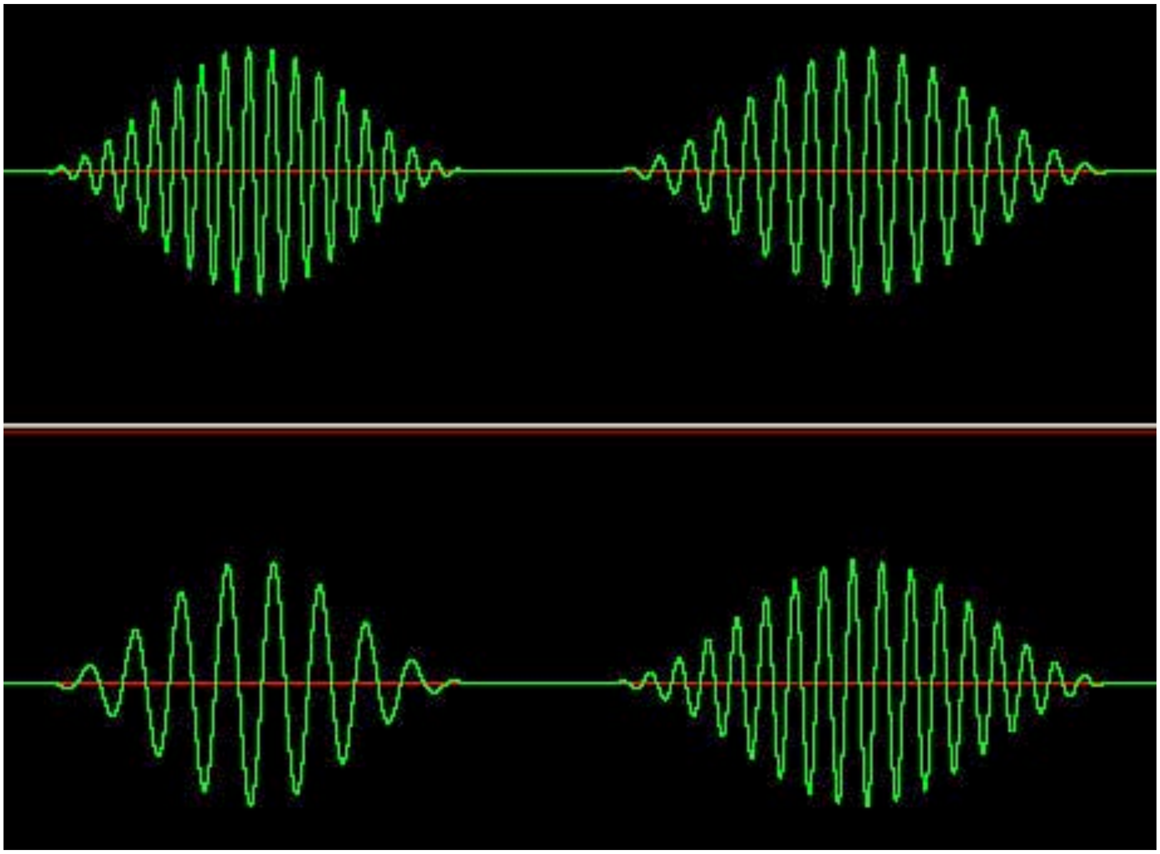
zontal line at the 'epoch' t. These are extreme cases. In general, signals cannot be represented by lines; but it is possible to associate with them a certain characteristic rectangle or 'cell' by the following process, which at first sight might perhaps appear somewhat complicated.

Consider a given signal described as $s(t)$ in 'time language' and by its Fourier transform $S(f)$ in 'frequency language'. If $s(f)$ is real, $S(f)$ will be in general complex, and the spectrum will extend over both positive and negative frequencies. This creates an unwelcome asymmetry between the two representations, which can be eliminated by operating with a complex signal $\psi(t) = s(t) + i\sigma(t)$, where $\sigma(t)$ is the Hilbert transform of $s(t)$, instead of with the real signal $s(t)$. This choice makes the Fourier transform $\varphi(f)$ of $\psi(t)$ zero for all negative frequencies. Next we define the 'energy density' of the signal as $\psi\psi^*$, where the asterisk denotes the conjugate complex value, and similarly $\varphi\varphi^*$ as the spectral energy density. In Fig. 1 the two energy distributions are shown as shaded areas. The two are of equal size; that is, the total energy of the signal is the same by both definitions. We can now define a 'mean epoch' $\bar{t}$ of the signal, and similarly a 'mean frequency' $\bar{f}$, as the co-ordinates of the centres of gravity of the two distributions. This gives a point C in the information diagram as the centre of the signal. Going a step

- 1960: Xenakis expands theory of screens and grains for creative sound production (Xenakis 1992)
- 1978: Curtis Roads introduces software for Granular Synthesis (Roads 1978, 1996, p. 168, 2002)

21.15. Granular Synthesis: Concepts

- Produce a stream of sounds with very short envelopes (10 to 200 ms)
- Envelopes function like windows; multiple windows are often overlapped
- Sounds may be derived from synthesized or sampled sources
- Parameters are frequently randomly adjusted (spacing, amplitude, duration)

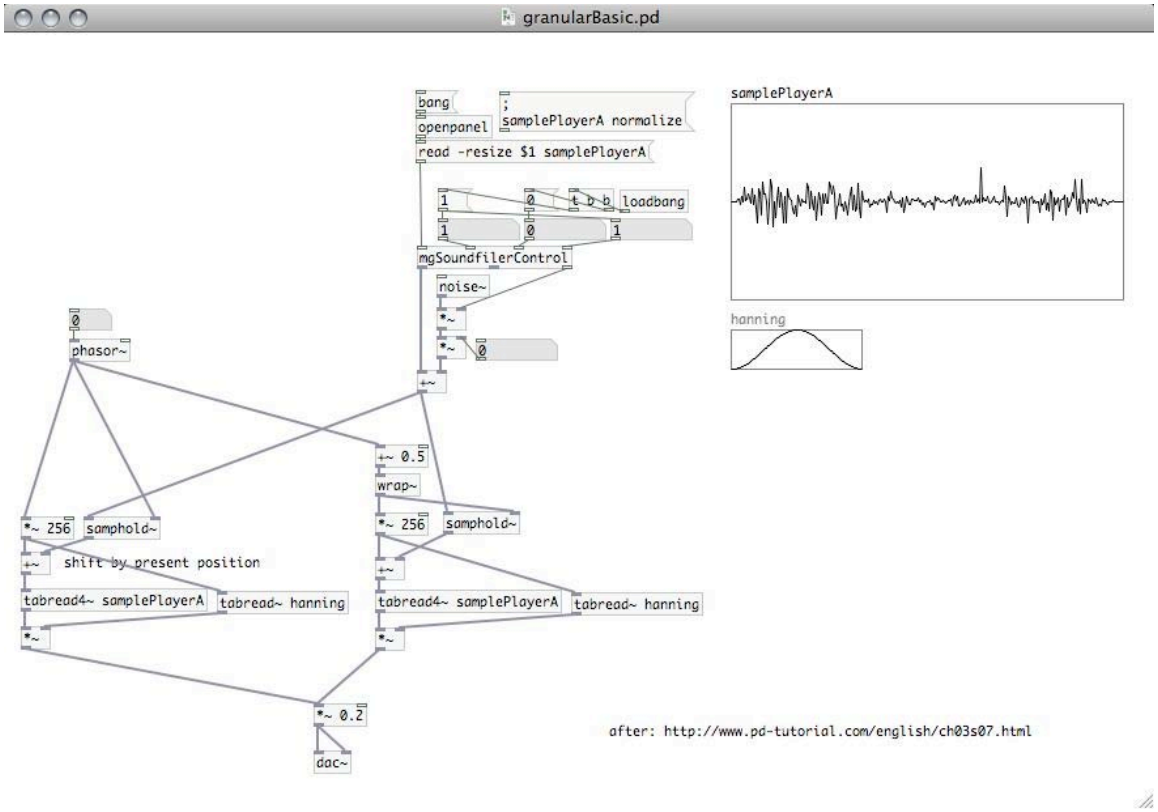


- Multiple streams are often combined combined
- Extreme control and speed suggests a procedure idiomatic to computer-based synthesis

21.16. Granular Synthesis: Pitch Shifting and Time Stretching

- The Eltro Information Rate Changer (1967)

- Four playback heads 90 degrees apart on a cylinder; when spun can make continuous contact with the tape
- By changing the direction and speed of the tape head rotation, could re-sample small bits of audio at a different speed without changing playback speed
- By changing the tape speed and the tape head rotation speed, could alter tempo without altering pitch
- Granular pitch/time shifting reads overlapping segments of an audio buffer, where each segment start position is consistent with the source playback speed, yet the reading of that segment can happen at a variable rate [demo/granularBasic.pd]



21.17. Grains in Csound

- Numerous highly-specialized, advanced opcodes are available in Csound and other synthesis languages
- "grain," "granule," (and more) for granular synthesis
- An instrument that smoothly moves from min to maximum density, granulating an audio file loaded into a wave-table
- Example: tutorial-a-07.csd

```
<CsoundSynthesizer>
<CsInstruments>
sr     = 44100
ksmps  = 10
nchnls = 2

instr 104
   iDur = p3
   iDensityMin = p4
   iDensityMax = p5

   iSnd = 98
   iBaseFq = 44100 / ftlen(iSnd)

   kDensity    line     iDensityMin, iDur, iDensityMax
   kGrainDur   line     .010, iDur, .030
```

```
   kAmpDev     line     0, iDur, 1000

   aSrc   grain  16000, iBaseFq, kDensity, kAmpDev, 0, kGrainDur, iSnd, 99, .100
          outs     aSrc, aSrc
endin
</CsInstruments>
<CsScore>
f  99   0    16384   20  1
f  98   0    1048576  1  "sax.aif" 0 0 0

i  104   0.0  10  .05 20
i  104  11.0  10   35 200
</CsScore>
</CsoundSynthesizer>
```

21.18. Listening: Curtis Roads

- Composer, computer musician, writer
- Significant early work with granular techniques
- Curtis Roads: "Now": *Line Point Cloud*

21.19. Listening: Trevor Wishart

- Sound mutations and transformations
- Interest in vocal sounds and new notations (Wishart 1996)
- Trevor Wishart, *Red Bird*, 1977

21.20. More Synthesis with Code

- A variety of low-level frameworks for DSP in C and C++: STK, openAL
- Numerous high-level languages related to Csound, often built in C/C++

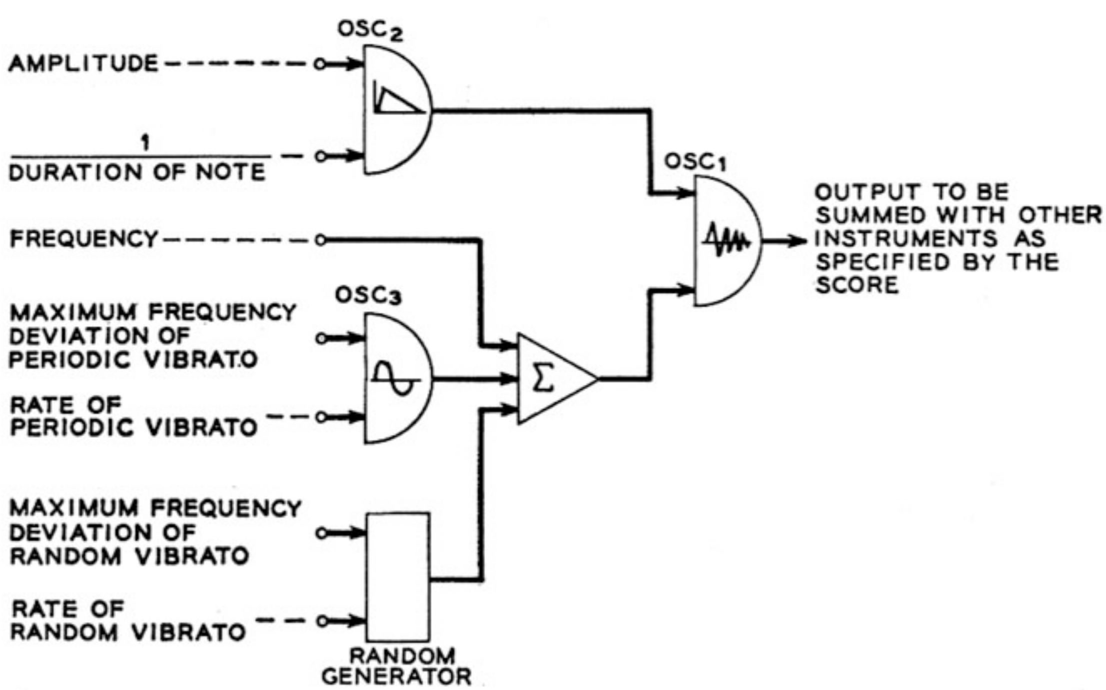
- Text-based: SuperCollider, ChucK, Nyquist, Cmix, Cmusic
- Graphic-based: PD/MaxMsp, Open Sound World, Reaktor

21.21. The Problem of Text

- Systems like Csound are powerful, but may make exploration and experimentation difficult
- Batch processing did not permit real-time, interactive systems
- Signal graph (or signal network or patching) can be spread across multiple lines of text

21.22. Signal Processing Block Diagrams

- Used in audio engineering
- Used to plan voltage-controlled synthesis systems before execution
- Used to illustrate unit generators and types of inputs and output in Music N languages
- Examples:



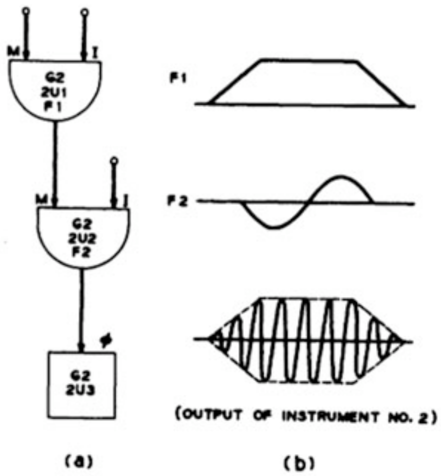


FIGURE 8
AN INSTRUMENT WITH ENVELOPE CONTROL

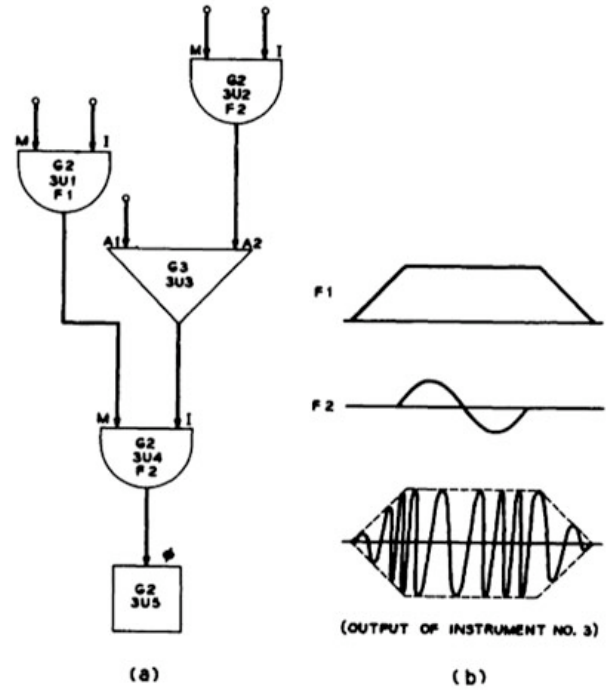


FIGURE 9
AN INSTRUMENT WITH ENVELOPE CONTROL AND VIBRATO

21.23. MaxMSP/PD

- Max is a visual programming paradigm
- Many diverse implementations: MaxMSP, jMax, Pd
- Emphasizes real-time control and signal flow design
- Emphasizes processes more than data

21.24. MaxMSP/PD: History

- 1979-1987: Miller Puckette studied with Barry Vercoe
- 1982: Puckette releases Music 500
- 1985: Working on a dedicated digital audio processor, Puckette designs a new system, keeping the Music 500 control structure; names it Max after Max Mathews's RTSKED (Puckette 1985)
- 1987: Re-rewrites Max in C for Macintosh (Puckette 1988)
- Max commercialized by David Zicarelli, fell through two companies, than reconsolidated at Cycling 74

- Puckette reprograms Max for IRCAM ISPW and NeXT Cube, and adds signal processing to Max (called Faster Than Sound (FTS))
- 1991: Max/FTS ported to other architectures
- IRCAM version becomes jMax
- Puckette reprograms system as Pure Data (PD), releases in 1997 as an open-source tool (Puckette 1997)
- Zicarelli, after PD's signal processing, creates Max Signal Processing (MSP) (Puckette 2002)
- PD-Extended offers a complete package of PD tools for all platforms

  http://puredata.info/downloads

21.25. SuperCollider: History

- Programming language and development environment for real-time signal processing
- First released in 1996 by James McCartney (McCartney 1996; McCartney 1998)
- 1999: version 2 released (Wells 1999)
- In 2002 version 3 released as an open source project

21.26. SuperCollider: Concepts

- Unit Generators are combined to produce SynthDefs
- A server-based architecture: SynthDefs live on a server and send and receive messages and signals
- A complete object-oriented language: create objects, manipulate, and reuse code
- Designed for real-time performance and experimentation
- Code can be executed piece by piece in the development environment
- Under active development and supported by a robust community

  http://supercollider.sf.net

21.27. SuperCollider: Basic Patching

- Can evaluate code interactively by selecting expressions and pressing Enter (not Return!)

- Creating noise

```
{WhiteNoise.ar(0.2)}.play
```

- Enveloping noise with a sine envelope scaled

```
{WhiteNoise.ar(0.2) * SinOsc.kr(4, mul:0.5, add:0.5)}.play
```

- Oscillating rate of envelope applied to noise

```
{
var envRate;
envRate = SinOsc.kr(0.3, mul:20, add:1.5);
WhiteNoise.ar(0.2) * SinOsc.kr(envRate, mul:0.5, add:0.5);
}.play
```

- Applying a low-pass filter

```
{
var envRate, preFilter;
envRate = SinOsc.kr(0.3, mul:20, add:1.5);
preFilter = WhiteNoise.ar(0.2) * SinOsc.kr(envRate, mul:0.5, add:0.5);
LPF.ar(preFilter, 900);
}.play
```

- Applying a low-pass filter with a cutoff frequency controlled by an oscillator; translating MIDI values to Hertz

```
{
var envRate, preFilter, cfControl;
envRate = SinOsc.kr(0.3, mul:20, add:1.5);
cfControl = SinOsc.kr(0.25, mul:0.5, add:0.5);
cfControl = (cfControl * 70) + 50;
preFilter = WhiteNoise.ar(0.2) * SinOsc.kr(envRate, mul:0.5, add:0.5);
LPF.ar(preFilter, cfControl.midicps);
}.play
```

21.28. SuperCollider: Creating SynthDefs and Sending Parameters

- Most often, SynthDefs are created and sent signals or parameters from other processes
- Create a SynthDef; create an envelope opened and closed by a gate; create LPF filtered noise; control the amplitude of the noise by the envelope; create a Task to loop through parameters for duration, sustain, and cutoff frequency scalar
- Example: tutorial-b.rtf

```
(
SynthDef(\noise, {|sus=2, ampMax=0.9, lpfCfScalar=20|
   var env, amp, gate, sigPrePan, cfControl;

   gate = Line.ar(1, 0, sus, doneAction: 2);
   env = Env.adsr(0.1*sus, 0.2*sus, 0.8, 0.1*sus, ampMax);
   amp = EnvGen.kr(env, gate);

   cfControl = SinOsc.kr(12, mul:0.5, add:0.5);
```

```
   cfControl = (cfControl * lpfCfScalar) + 40;

   sigPrePan = LPF.ar(WhiteNoise.ar(amp), cfControl.midicps);
   Out.ar(0, Pan2.ar(sigPrePan, 0.5));
}).send(s);

r = Task({
        var dur, sus, fq, delta;
        dur = Pseq([0.5, 0.5, 0.25], 6).asStream;
        sus = Pseq([0.2, 0.2, 0.2], 6).asStream;
        fq = Pseq([60, 30, 20, 40], 6).asStream; // midi pitch values

        while {delta = dur.next;
                 delta.notNil
        } {
                Synth(\noise, [sus: sus.next, lpfCfScalar: fq.next]);
                delta.yield;
        }
});
r.play()
)
```

- Adding randomized panning control and cutoff frequency scalar
- Example: tutorial-c.rtf

```
(
SynthDef(\noise, {|sus=2, ampMax=0.9, lpfCfScalar=20, pan=0.5|
   var env, amp, gate, sigPrePan, cfControl;

   gate = Line.ar(1, 0, sus, doneAction: 2);
   env = Env.adsr(0.1*sus, 0.2*sus, 0.8, 0.1*sus, ampMax);
   amp = EnvGen.kr(env, gate);

   cfControl = SinOsc.kr(12, mul:0.5, add:0.5);
   cfControl = (cfControl * lpfCfScalar) + 40;

   sigPrePan = LPF.ar(WhiteNoise.ar(amp), cfControl.midicps);
   Out.ar(0, Pan2.ar(sigPrePan, pan));
}).send(s);

r = Task({
        var dur, sus, fq, delta, pan;
        dur = Pseq([0.5, 0.5, 0.25], 6).asStream;
        sus = Pseq([0.2, 0.2, 0.2], 6).asStream;
        fq = Pshuf([60, 30, 20, 40], 6).asStream; // midi pitch values
        pan = Pshuf([0, 0.2, 0.4, 0.6, 0.8, 1], 6).asStream;

        while {delta = dur.next;
                 delta.notNil
        } {
                Synth(\noise, [sus: sus.next, lpfCfScalar: fq.next, pan: pan.next]);
                delta.yield;
        }
});
r.play()
)
```

21.29. Live Coding

- A performance practice of computer music that emphasizes the creation of code

- Computer screens are projected while code is used to build-up musical parts
- Software such as SuperCollider, Impromptu, and ChucK are used
- Live Coding with aa-cell

  YouTube (http://www.youtube.com/watch?v=OBt4PLUv2q0)