FM SYNTHESIS

A classic synthesis algorithm

Roger B. Dannenberg
Professor of Computer Science, Art, and Music

Frequency Modulation

- Frequency modulation occurs naturally:
  - Voice inflection, natural jitter, and vibrato in singing
  - Vibrato in instruments
  - Instrumental effects, e.g. electric guitar
  - Many tones begin low and come up to pitch
  - Loose vibrating strings go sharp as they go louder
  - Slide trombone, Theremin, voice, violin, etc. create melodies by FM (as opposed to, say, pianos)
Frequency Modulation with Nyquist

- \texttt{fmosc(basic-pitch, fm-control [, table [, phase]]))}
  \textit{fm-control} is expressed as deviation in Hz

- \texttt{hzosc(fm-control)}
  \textit{fm-control} is absolute frequency in Hz

- \texttt{snd-compose(f, g)}
  Computes f(g(t)) – if g is non-linear, frequency changes occur

FM EXAMPLES
Exploring the sound world of FM synthesis
Examples

- See Code 4 (code_4.sal)

Why FM Synthesis?

- We’ve already seen wavetable or table-lookup synthesis:
  - Very efficient
  - Create any harmonic spectrum
  - Simple frequency and amplitude control
- What’s missing?
  - Time-varying control over the spectrum
  - Inharmonic spectra
- Various Approaches:
  - Synthesize each sinusoid separately – tedious, costly
  - Filter the output of table – useful, but only harmonic output
  - FM Synthesis
FM Synthesis

- When modulation frequency is in the audio range, interesting things happen.

Mathematics of FM

- The exact amplitudes of the partials generated by FM are described by Bessel functions.
- These functions are messy, their evolution is messier, and there is no simple way to invert the functions.
- Many lives of FM:
  - 1967-1968 Invented by John Chowning, patented 1975
  - 1983-1986: Yamaha DX7 160,000 sold
  - 1990-1995: IBM PC-compatible Sound Cards
  - 2000's: FM synthesis provides polyphonic ring tones
FM and Harmonics

- Generated frequencies are:
  \[ C \pm nM \]
- Where C = “Carrier” and M = Modulator
- Simplest case: C = M
- Generated frequencies are:
  \[ C+nM \] gives us C, 2C, 3C, 4C, …
- What about negative frequencies?

FM and Harmonics (2)

Bandwidth \~ 2(D+M)
FM and Harmonics (3)

Bandwidth~2(D+M)

Classic FM brass sound

- Characterized by a rise in upper partials
- Generated by increasing depth of modulation
- Uses 1:1 Carrier:Modulation frequency

More partials over time

- See example in code_4.htm
Odd Harmonics

\[ C \pm nM \]

- Let \( M = 2C \)
- Resulting frequencies are \( C, 3C, 5C, \ldots \)
- Negative frequencies are \(-C, -3C, -5C, \ldots\)
- Try it…

Other Harmonic Schemes

\[ C \pm nM \]

- Let \( M = \frac{i}{j} \times C \), for small integers \( i \) and \( j \)
- Let \( F = C/j \), then \( M = iF \)
- \( C = jF \), \( C + M = (i+j)F \), \( C + 2M = (2i+j)F \), etc.
- All frequencies are harmonics (integer multiples) of \( F \)
- Try it…
**Inharmonic Partials**

\[ C \pm nM \]

- Let \( M = \text{not } i/j \times C \)
- Resulting frequencies are not harmonics
- Negative frequencies are not harmonics
- Try it…

**Formants**

- Resonances (especially in the vocal tract) emphasize frequencies around the resonant frequency
- We can simulate resonances (and voice) by placing a carrier near the desired resonant frequency and modulating it to create nearby harmonics:
Summary

- FM Synthesis
  - Time varying spectra
  - Low cost (simplest case is only 2 oscillators)
  - Simple parametric control
  - Musically useful results
- FM Control
  - Carrier:Modulator ratio
    - Harmonic or inharmonic spectra
    - Odd or all harmonics
    - Formants
  - Depth of modulation
    - Number of partials

See examples in code_4.sal

BEHAVIORAL ABSTRACTION

A sound event can behave differently according to the context in which it is instantiated.
Temporal Semantics and Behavioral Abstraction

- Extensions to ordinary (Lisp, SAL) semantics:
  - Behaviors
  - Evaluation environment
  - Transformations
  - Temporal combination: SEQ and SIM

Behaviors

- Nyquist sound expressions denote a whole class of behaviors
- The specific sound computed by the expression depends upon the environment
- Transformations like STRETCH and TRANSPOSE alter the behavior.
- Behaviors vs. linear transformation: when you play a longer note, you don’t simply stretch the signal! The behavior concept is critical for music.
Evaluation Environment

- To implement behavior concept, all Nyquist expressions evaluate within an environment.
- Nyquist environment includes: starting time, stretch factor, transposition, legato factor, loudness, sample rates, and more.
- Environment is “hidden” and changed or accessed using special function-like constructs.

Manipulating the Environment

- Example:
  \[ \text{osc(c4)} \sim 3 \]
  - Within STRETCH, all expressions see altered environment and behave accordingly
- Scoping is dynamic:
  \[
  \text{function tone() return osc(c4)} \\
  \text{play tone() \sim 3 \rightarrow \langle? \text{ second sound}\rangle}
  \]
- Transformations can be nested:
  \[
  \text{function tone() return osc(c4) \sim 2} \\
  \text{play tone() \sim 3 \rightarrow \langle? \text{ second sound}\rangle}
  \]
Manipulating the Environment

- Example:
  \[ \text{osc}(c4) \sim 3 \]
- Within STRETCH, all expressions see altered environment and behave accordingly
- Scoping is \textit{dynamic}:
  ```
  function tone() return \text{osc}(c4)
  play tone() \sim 3 \rightarrow \text{<3 second sound>}
  ```
- Transformations can be nested:
  ```
  function tone() return \text{osc}(c4) \sim 2
  play tone() \sim 3 \rightarrow \text{<6 second sound>}
  ```

Absolute Transformations

- You can override the “inherited” environment:
  ```
  function tone2() return \text{osc}(c4) \sim 2
  play tone2() \sim 100 \rightarrow \text{<2 second tone>}
  ```
- Even though TONE2 is called with a stretch factor of 100, its STRETCH-ABS transformation overrides the environment and sets it to 2
- \textit{Once sound is computed by OSC(C4), the sound is immutable, i.e. not subject to transformation!!!!!!}
The SOUND Type

- \texttt{osc(c4) \sim 2} \quad \leftarrow \text{this is an expression}
- When evaluated, \texttt{osc()} uses the environment (especially start time and stretch factor) and returns a SOUND:

![Waveform diagram with labels: Start time, Sample Rate, Terminate time, Logical stop time]

Example

- \texttt{begin}
  
  \texttt{with x = osc(c4)}
  
  \texttt{play x \sim 3} \quad \Rightarrow \quad \texttt{<? second tone>}

  \texttt{end}

- \texttt{function x() return osc(c4)}
  
  \texttt{play x() \sim 3} \quad \Rightarrow \quad \texttt{<? second tone>
Example

• begin
  with x = osc(c4)
  play x ~ 3 \rightarrow \text{<1 second tone>}
end

• function x() return osc(c4)
  play x() ~ 3 \rightarrow \text{<3 second tone>}

Transformations

• STRETCH, STRETCH-ABS (~, ~~)
• AT, AT-ABS (@, @@)
• LOUD, LOUD-ABS
• SUSTAIN, SUSTAIN-ABS
• ABS-ENV – use default environment
• See manual for others.
• Maybe we’l talk about time-varying transformations later in semester.
Practical Notes

- In practice, the most critical transformations are AT (@) and STRETCH (~), which control *when* sounds are computed and how long they are.
- Technically, transformations are not functions because they do not evaluate their arguments in the normal order: instead, they manipulate the environment, evaluate the behavior, then restore the environment.
- Implemented as macros in XLISP

SEQ

A construct for sequential behavior
SEQ

- How do we make a sequence of sounds: 
  \( \text{seq}(\text{osc}(c4), \text{osc}(d4)) \)

- Semantics:
  - Evaluate \( \text{osc}(c4) \) at default time (t=0)
  - Resulting sound has \textit{logical stop time} of 1.0
  - Evaluate \( \text{osc}(d4) \) at start time t=1.0
  - Return the sum of the results

Counterexample

- You MUST use \texttt{seq} with behavior expressions, not sound values:

  ```
  set x = \text{osc}(c4); \texttt{compute sounds}
  set y = \text{osc}(d4);
  play seq(x, y); \texttt{WRONG!!}
  
  function x() return \text{osc}(c4); \texttt{define}
  function y() return \text{osc}(d4); \texttt{behaviors}
  play seq(x(), y()); \texttt{RIGHT!!}
  ```
SIM

A construct for simultaneous behavior

- **SIM** is exactly the same as **SUM** and **+**
- SIM evaluates a list of behaviors and forms their sum (equivalent to audio mixing)
- \( \text{sim(osc(c4), osc(g4))} \)
Example Using @

\[
\begin{align*}
\text{• play sim}( & \text{osc}(c4), \\
& \text{osc}(e4) @ 0.1, \\
& \text{osc}(g4) @ 0.2, \\
& \text{osc}(b4) @ 0.3), \\
& \text{osc}(d5) @ 0.4)
\end{align*}
\]

LOGICAL STOP TIME

Decoupling the “logical” end of a sound (its duration) from the “physical” end of a sound (its articulation)
Overlap With Logical Stop Times

- `play seq(set-logical-stop(\text{osc}(c_4), 0.1),
  set-logical-stop(\text{osc}(e_4), 0.1),
  set-logical-stop(\text{osc}(g_4), 0.1),
  set-logical-stop(\text{osc}(b_4), 0.1),
  set-logical-stop(\text{osc}(d_5), 0.1))`

Scores

- We’ve seen scores already
- To evaluate a score, evaluate each sound expression with the start time and stretch factor:
  - `\{\{\text{start dur \{instr parameters\}}\} \Rightarrow \text{instr(parameters)} \sim \text{dur @ start}`
  - Note: `\text{instr()} \sim \text{dur @ start} \Leftrightarrow \text{instr()} \text{ @ (start / dur) \sim dur}`
Summary

- SOUNDS
  - Start time
  - Logical stop time
  - Physical stop time
- Functions evaluated in an *environment*
  - Dynamically scoped – inherited across calls
  - Modified by transformations
    - Stretch (~)
    - Shift (@)
- Results of functions (SOUNDS) are immutable
- Sim and Seq control constructs