Modern Modular Synthesizers

Where does Yampa enter the picture?

• Music can be seen as a hybrid phenomenon. Thus interesting to explore a hybrid approach to programming music and musical applications.

• Yampa’s programming model is very reminiscent of programming modular synthesizers:

• Fun application! Useful for teaching?
What have we done?

Framework for programming modular synthesizers in Yampa:
- Sound-generating and sound-shaping modules
- Additional supporting infrastructure:
  - Input: MIDI files (musical scores), keyboard
  - Output: audio files (.wav), sound card
  - Reading SoundFont files (instrument definitions)
- Status: proof-of-concept, but decent performance.

Example 1: Sine oscillator

oscSine :: Frequency → SF CV Sample
oscSine f0 = proc cv → do
  let f = f0 * (2 ** cv)
  phi ← integral ← 2 * pi * f
  returnA ← sin phi

constant 0 >> oscSine 440

Example 2: Vibrato

constant 0
>> oscSine 5.0
>> arr (*0.05)
>> oscSine 440

Example 3: 50’s Sci Fi

sciFi :: SF () Sample
sciFi = proc () → do
  und ← arr (*0.2) ≪ oscSine 3.0 ≪ 0
  swp ← arr (+1.0) ≪ integral ≪ −0.25
  audio ← oscSine 440 ≪ und + swp
  returnA ← audio
Envelope Generators (1)

\[ envGen :: CV \rightarrow [(\text{Time}, CV)] \rightarrow (\text{Maybe Int}) \rightarrow SF (\text{Event } ()) (CV, \text{Event } ()) \]
\[ envEx = envGen 0 [(0.5, 1), (0.5, 0.5), (1.0, 0.5), (0.7, 0)] \]
\[ (\text{Just 3}) \]

Envelope Generators (2)

How to implement?
Integration of a step function yields suitable shapes:

Envelope Generators (3)

\[ afterEach :: [(\text{Time}, b)] \rightarrow SF a (\text{Event } b) \]
\[ hold :: a \rightarrow SF (\text{Event } a) a \]
\[ steps = afterEach [(0.7, 2), (0.5, -1), (0.5, 0), (1, -0.7), (0.7, 0)] \]
\[ \gg\gg \text{hold 0} \]

Envelope Generators (4)

Envelope generator with predetermined shape:

\[ envGenAux :: CV \rightarrow [(\text{Time}, CV)] \rightarrow SF a CV \]
\[ envGenAux l0 tls = afterEach trs \gg\gg \text{integral} \gg\gg \text{arr} (+l0) \]

where
\[ (r0, trs) = \text{toRates} l0 \text{ tls} \]
Envelope Generators (5)

Envelope generator responding to key off:

\[ \text{envGen} :: CV \rightarrow [(Time, CV)] \rightarrow (Maybe \text{ Int}) \rightarrow SF \text{ (Event () (CV, Event ()]} \]

\[ \text{envGen l0 tls (Just } n) = \]

\[ \text{switch (.proc notoff } \rightarrow \text{ do} \]

\[ l \leftarrow \text{envGenAux l0 tls1 } \rightarrow () \]

\[ \text{returnA } \leftarrow ((l, \text{noEvent}), \text{notoff } 'tag' \ l)) \]

\[ (\lambda l \rightarrow \text{envGenAux l tls2} \& \& \text{after (sum (map fst tls2)) ()}) \]

\[ \text{where} \]

\[ (tls1, tls2) = \text{splitAt n tls} \]

Example 4: Bell

\[ \text{bell} :: \text{Frequency } \rightarrow SF () (\text{Sample, Event}) \]

\[ \text{bell } f = \text{proc () } \rightarrow \text{ do} \]

\[ m \leftarrow \text{oscSine} (2.33 \ast f) \leftarrow 0 \]

\[ \text{audio } \leftarrow \text{oscSine } f \leftarrow 2.0 \ast m \]

\[ (\text{ampl, end}) \leftarrow \text{envBell} \leftarrow \text{noEvent} \]

\[ \text{returnA } \leftarrow (\text{audio } \ast \text{ampl, end}) \]

Example 5: Tinkling Bell

\[ \text{tinkle} :: SF () \text{ Sample} \]

\[ \text{tinkle} = \text{(repeatedly 0.25 84} \]

\[ \gg \gg \text{constant ()} \]

\[ \& \& \text{arr (fmap (bell } \circ \text{midiNoteToFreq}) \]

\[ \gg \gg \text{rSwitch (constant 0}) \]

Example 6: Playing a C-major scale

\[ \text{scale} :: SF () \text{ Sample} \]

\[ \text{scale} = (\text{afterEach } [(0.0, 60), (2.0, 62), (2.0, 64), (2.0, 65), (2.0, 67), (2.0, 69), (2.0, 71), (2.0, 72)] \]

\[ \gg \gg \text{constant ()} \]

\[ \& \& \text{arr (fmap (bell } \circ \text{midiNoteToFreq}) \]

\[ \gg \gg \text{rSwitch (constant 0}) \]

\[ \& \& \text{after 16 ()} \]
Example 7: Playing simultaneous notes

```haskell
mysterySong :: SF () (Sample, Event ()
mysterySong = proc t → do
  t ← tinkle ()
  m ← mystery ()
  return A ← (0.4 * t + 0.6 * m)
```

A polyphonic synthesizer (1)

Sample-playing monophonic synthesizer:
- Read samples (instrument recordings) from SoundFont file into internal table.
- Oscillator similar to sine oscillator, except table lookup and interpolation instead of computing the sine.

SoundFont synthesizer structure:

A polyphonic synthesizer (2)

Exploit Yampa’s switching capabilities to:
- create and switch in a mono synth instance in response to each note on event;
- switch out the instance in response to a corresponding note off event.

Switched-on Yampa?

Software and paper: [www.cs.nott.ac.uk/~ggg](http://www.cs.nott.ac.uk/~ggg)