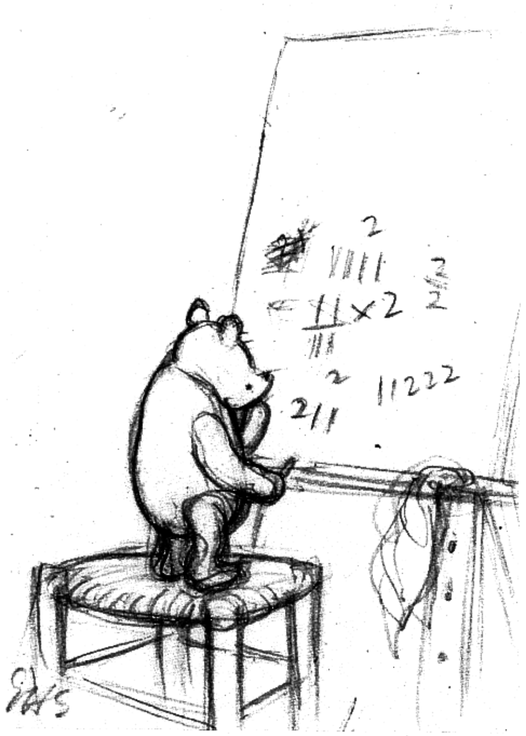
Q: A Functional Programming Language for Multimedia Applications

Albert Gräf

Dept. of Music-Informatics

Quick overview

- What?
- Why?
- The Library
- MIDI, Audio and OSC Interfaces
- Demo
- Conclusion



What?

```
sqr X = X*X;
```

- A *functional programming language* based on *term rewriting*.

```
sqr 2+sqr (2+3) ⇒ 2*2+sqr (2+3)
  ⇒ 4+sqr (2+3) ⇒ 4+sqr 5
  ⇒ 4+5*5 ⇒ 4+25 ⇒ 29
```

- Programs are collections of algebraic *equations*.

```
gcdiv X Y = gcdiv Y X if Y>X;
          = gcdiv Y (X mod Y) if Y>0;
          = X otherwise;
```

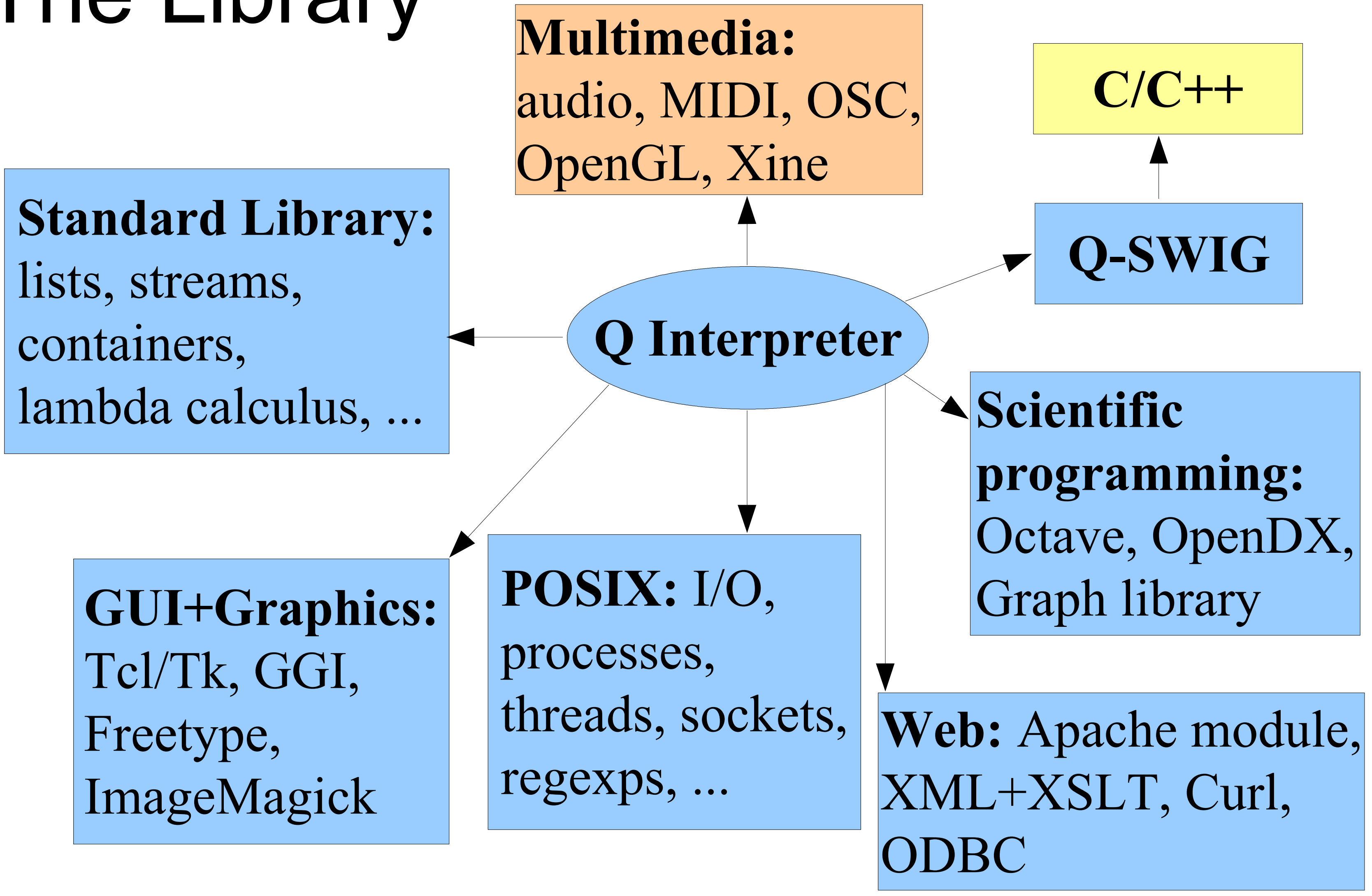
- Executing a program means to *evaluate an expression*.

```
qsort []     = [];
qsort [X|Xs] = qsort (filter (<X) Xs) ++
  [X] ++ qsort (filter (>=X) Xs);
```

Why?

- Started as a (master) research project on pattern matching techniques for term rewriting.
- Idea was to turn this into a simple, practical programming language (ca. 1991).
- Turned out quite different from both ML and Haskell. Simpler. Interpreted. Dynamic typing.
  $\Rightarrow$ “functional scripting language”
- Multimedia facilities in other modern-style FPLs were missing when I needed them, decided to do my own.

The Library


Multimedia: audio, MIDI, OSC, OpenGL, Xine
C/C++
Standard Library: lists, streams, containers, lambda calculus, ...
Q-SWIG
Q Interpreter
Scientific programming: Octave, OpenDX, Graph library
GUI+Graphics: Tcl/Tk, GGI, Freetype, ImageMagick
POSIX: I/O, processes, threads, sockets, regexps, ...
Web: Apache module, XML+XSLT, Curl, ODBC

MIDI Interface

- based on Grame's *MidiShare*
- dynamic routing and realtime processing of MIDI messages
- algebraic `MidiMsg` type; sequences are represented as lists
- standard MIDI file support

MIDI Interface

```
import midi;

/* register a MidiShare client and establish I/O connections */
def REF = midi_open "Transpose",
  IO = midi_client_ref "MidiShare/ALSA Bridge",
  _ = midi_connect IO REF || midi_connect REF IO;

/* transpose note on and off messages, leave other messages unchanged */
transp K (note_on CH N V)
                = note_on CH (N+K) V;
transp K (note_off CH N V)
                = note_off CH (N+K) V;
transp K MSG    = MSG otherwise;

/* the following loop repeatedly reads a message, transposes it and
   immediately outputs the transformed message */
transp_loop K   = midi_send REF 0 (transp K MSG) || transp_loop K
                    where (_,_,_,MSG) = midi_get REF;
```

Audio Interface

- `audio` module: *PortAudio* interface
- `sndfile` module: *Libsndfile* interface
- `wave` module: simple wave generation and manipulation operations, wave drawing, interface to *libsamplerate* and *FFTW*

OSC Interface

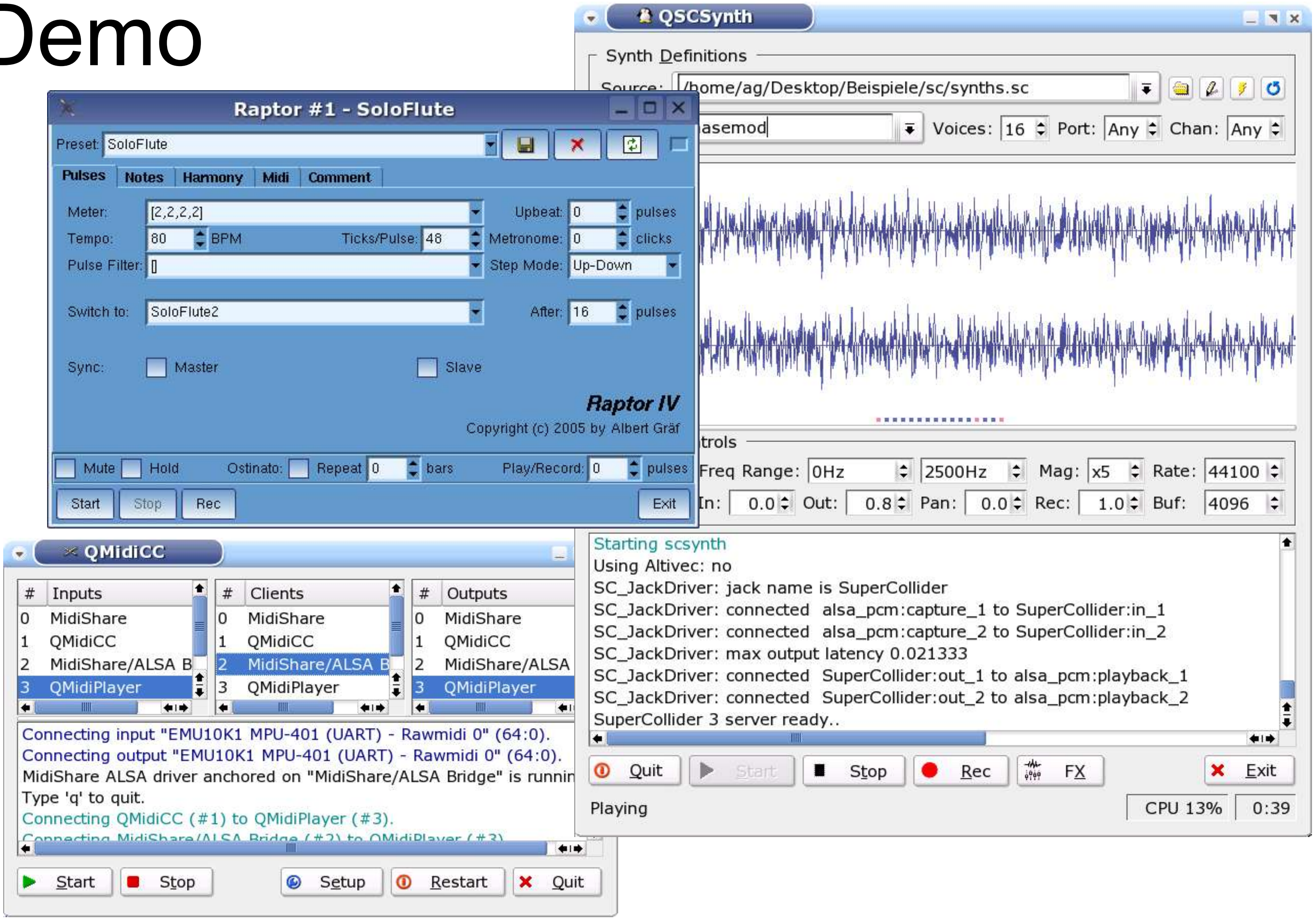
- implements Berkeley's *Open Sound Control* protocol
- all standard OSC features supported, including nested bundles
- UDP support
- special support for *SuperCollider*
- current version is written in Q; might use *liblo* in the future

OSC Interface

```
/* note offs: set the gate of the synth to 0 and put it at the end of the
   queue */
loop P Q (_,note_on _ N 0)
                = n_set I ("gate",0) || loop P Q midiin
                    where (I,_) = P!N, P = delete P N, Q = append Q I;
                = loop P Q midiin otherwise;
loop P Q (T,note_off CH N _)
                = loop P Q (T,note_on CH N 0);

/* note ons: turn note off if already sounding, then get a new voice from
   the queue and set its gate to 1 */
loop P Q (T,note_on CH N V)
                = n_set I ("gate",0) || loop P Q (T,note_on CH N V)
                    where (I,_) = P!N, P = delete P N, Q = append Q I;
                = n_set I ("freq",FREQ,"gain",V/127,"gate",1) ||
                  loop P Q midiin
                    where [I|Q] = Q, FREQ = freq N,
                      P = insert P (N,(I,FREQ));
```

Demo



Source: /home/ag/Desktop/Beispiele/sc/synths.sc
asemod Voices: 16 Port: Any Chan: Any

Preset: SoloFlute
Meter: [2,2,2,2] Upbeat: 0 pulses
Tempo: 80 BPM Ticks/Pulse: 48 Metronome: 0 clicks
Pulse Filter: [] Step Mode: Up-Down
Switch to: SoloFlute2 After: 16 pulses
Sync: Master Slave
Copyright (c) 2005 by Albert Gräf
Mute Hold Ostinato: Repeat 0 bars Play/Record: 0 pulses

trols
Freq Range: 0Hz 2500Hz Mag: x5 Rate: 44100
In: 0.0 Out: 0.8 Pan: 0.0 Rec: 1.0 Buf: 4096

```
Starting scsynth
Using Altivec: no
SC_JackDriver: jack name is SuperCollider
SC_JackDriver: connected  alsa_pcm:capture_1 to SuperCollider:in_1
SC_JackDriver: connected  alsa_pcm:capture_2 to SuperCollider:in_2
SC_JackDriver: max output latency 0.021333
SC_JackDriver: connected  SuperCollider:out_1 to alsa_pcm:playback_1
SC_JackDriver: connected  SuperCollider:out_2 to alsa_pcm:playback_2
SuperCollider 3 server ready..
```

Playing CPU 13% 0:39

#	Inputs	#	Clients	#	Outputs
0	MidiShare	0	MidiShare	0	MidiShare
1	QMidiCC	1	QMidiCC	1	QMidiCC
2	MidiShare/ALSA B	2	MidiShare/ALSA B	2	MidiShare/ALSA
3	QMidiPlayer	3	QMidiPlayer	3	QMidiPlayer

```
Connecting input "EMU10K1 MPU-401 (UART) - Rawmidi 0" (64:0).
Connecting output "EMU10K1 MPU-401 (UART) - Rawmidi 0" (64:0).
MidiShare ALSA driver anchored on "MidiShare/ALSA Bridge" is runnin
Type 'q' to quit.
Connecting QMidiCC (#1) to QMidiPlayer (#3).
Connecting MidiShare/ALSA Bridge (#2) to QMidiPlayer (#3)
```

Conclusion

- Q: a modern-style functional programming language based on term rewriting.
- Already good support for multimedia and computer music applications.
- Future work: library support (Jack, LADSPA, DSSI, ...), high-level interfaces.
- It's free! (GPL)
- More info: q-lang.sf.net