ChucK => A Concurrent, On-the-fly Audio Programming Language

Ge Wang *Perry R. Cook

Department of Computer Science
(*Also Music)
Princeton University

ICMC 2003
What is ChucK?

• General-purpose audio programming language
  – Strongly typed and strongly timed
  – Object oriented
  – Platform independent
  – Designed from the ground up
    • Not based on a single existing language

• A New Programming Model

• Solves a number of problems
Audio Programming

• Considerations
  – Development
    • Flexibility & Programmability
    • Elegant mapping of audio concepts to language
    • Levels of abstraction
    • Timing (and Concurrency?)
Audio Programming

• **Considerations**
  - Development
    • Flexibility & Programmability
    • Elegant mapping of audio concepts to language
    • Levels of abstraction
    • Timing (and Concurrency?)
  - Run-time
    • Real-time
    • Usability / control
    • Performance
Audio Programming

• Considerations
  – Development
    • Flexibility & Programmability
    • Elegant mapping of audio concepts to language
    • Levels of abstraction
    • Timing (and Concurrency?)
  – Run-time
    • Real-time
    • Usability / control
    • Performance

• Problem:
  DIFFICULT TO INCORPORATE INTO A SINGLE LANGUAGE
Question: Can we design one language that supports:

• Representation of audio concepts
  – reason about flow from code

• Both high **AND** low levels of abstraction
  – Data and time
  – reason about time directly from code

• Concurrency
  – can be useful (but hard!)
    • Timing
    • Synchronization
    • Performance
Problem 1: Representation

dacOut.tick(biQuad.tick(adsr.tick() * noise.tick()));

if( timeCount % 44 == 0 )
{
    if( midi.nextMessage() == __SK_NoteOn_ )
    {
        (midi.getByteThree() ? adsr.noteOn() : adsr.noteOff());
        biQuad.setFreq( MD2Freq(midi.getByteTwo()) );
    }
}
Solution 1: Representation

noise3 => ADSR => biquad1 => dac;
midi => (ADSR, biquad1);
Solution 1: Representation
• **Simple chuck**: \( x \implies y \);
• **Chain chuck**: \( w \implies x \implies y \implies z \);
• **Nested chuck**: \( w \implies (x \implies y) \implies z \);
• **Cross chuck**: \( w \implies (x, y) \);
  \( (w, v) \implies (x, y, z) \);

• **Un-chuck**: \( x \iff y \iff z \);
• **Re-chuck**: \( x \iff y \);
• **Action-oriented**

• **Overloaded on Types**
  - Defined for primitives
  - Unit generators
  - System mechanisms

• **Chuck operator helps represent flow**
Problem 2: Level of Control

• What is the appropriate level of control for...
  – Data? (bits, bytes, objects, ...)
  – Time & Control rate? (sample, ms, ...)

• Solution: provide many levels of control!
  – Work at multiple levels of abstraction for both data and time
Problem 2: Level of Control

noise => env => biQuad => dac;

while ( true )
{
    500 + (300 * sin(now*FC)) => biQuad.freq;
    100:ms => now;
}

Demo 1
ChucK Timing Constructs

• **dur** is a native type
  - **Units:**
    • samp, ms, second, minute, hour, day, week
  - **Arithmetic:**
    • \(3:\text{second} + 100:\text{ms} \Rightarrow \text{dur} \text{ quarter};\)

• **time** is a native type
  - **now** keyword holds current ChucK time
  - **Arithmetic:**
    • \(5:\text{second} + \text{now} \Rightarrow \text{time} \text{ later};\)
Time Example

4:second + now => time later;
while( now < later )
{
    now => stdout;
    1:second => now;
}

> chuck foo.ck
0
1
2
3
Advancing Time

• Time stands still until you “advance” it

• Two semantics for advancing time
  – Chuck to now
    1:second => now;
  – Wait on event
    midi_event => midi_handler;

• You are responsible for keeping up with time

• Timing embedded in program flow
How It Works

Execution Unit

Audio Engine

Scheduler

Process (running)

Process (waiting)

DAC

now

future
Dynamic Control Rate

"snare" => sndbuf sbuf => dac;

50 => int r;
0 => float a;

while( true )
{
    0 => sbuf;
    70 + (30 * sin(a)) => r;
    .1 +=> a;
    r::ms => now;
}
Consequences of Timing Mechanism

• Consistent, deterministic notion of time
• Strong correspondence of program flow and timing
• Control rate
  – Arbitrary
  – Dynamic
  – Sample-Synchronous

Timing model makes chuck more powerful by allowing deterministic concurrency...
Problem 3: Concurrent Audio Programming

noise => env => biQuad => dac;

0.0 => float v;
while (true)
{
    sin(v*FC) => biQuad.freq;
    1.0 +=> v;
    100:ms => now;
}

while (true)
{
    if (midiin.noteOn() )
        midiin.vel => env;
    13:ms => now;
}
Concurrency is hard!

• Synchronization and timing
• Overhead
• Programming model
  – expressing concurrency in code
Programming with Shreds

• **Threads**
  - Preemptive / non-deterministic
  - Use a fork()
  - No timing guarantees

• **Shreds**
  - Deterministic shred of computation
  - Use a spork()
  - Sample-synchronous precision
Properties of Shreds

• Resemble non-preemptive threads
• User-level constructs (no kernel interaction)
  – High-performance
    • Context switch
    • Synchronization / scheduling
• Automatically synchronized by time!
• Managed by the ChucK shreduler
• Deterministic execution
How it Works

Execution Unit

Audio Engine

Shred

Shreduler

Shred

Shred

DAC

now

future
CONTROL RATES AND SHREDS

• Determined by programmer per shred

• Chuck control rates are:
  - Arbitrary
  - Dynamic
  - Simultaneous

3-shred example:

```plaintext
500::ms => now;
"bass" => sndbuf sb => dac;
while( 1 )
{
  0 => sb;
  500::ms => now;
}

750::ms => now;
"snare" => sndbuf sb => dac;
while( 1 )
{
  0 => sb;
  500::ms => now;
}

625::ms => now;
"hatcl" => sndbuf sb => dac;
while( 1 )
{
  0 => sb;
  250::ms => now;
}
```

Demo 3
Consequences of Concurrency with Timing

- Possible to write truly parallel, sample-synchronous audio code
- Can work at low and high level
  - Fine granularity == power and control
  - Arbitrary granularity == flexibility and EFFICIENCY
- Program on-the-fly...
On-the-fly Programming

• **Goal:** add/modify/remove parts of the program as it is running

• **Experimentation & performance possibilities**

• **Use the shred mechanism**
  - Add and remove shreds from the virtual machine
  - Use networking, GUI, or command line
ChucK Virtual Machine

Code ("foo.ck", "bar.ck")

Process

shred → shred → shred

On-the-fly compiler

Shreduler

Audio Engine

Execution Unit

I/O Manager
Acknowledgements

Thanks => ( Andrew Appel, David August, Brian Kernighan, Kai Li, David Walker, Dan Trueman, Amal Ahmed, Lujo Bauer, Limin Jia, Jared Ligatti, Kedar Swadi, Spyros Triantafyllis, Ari Lazier );

ChucK is free from the authors:

http://chuck.cs.princeton.edu/
http://soundlab.cs.princeton.edu/
QUESTIONS?

http://chuck.cs.princeton.edu/
http://soundlab.cs.princeton.edu/