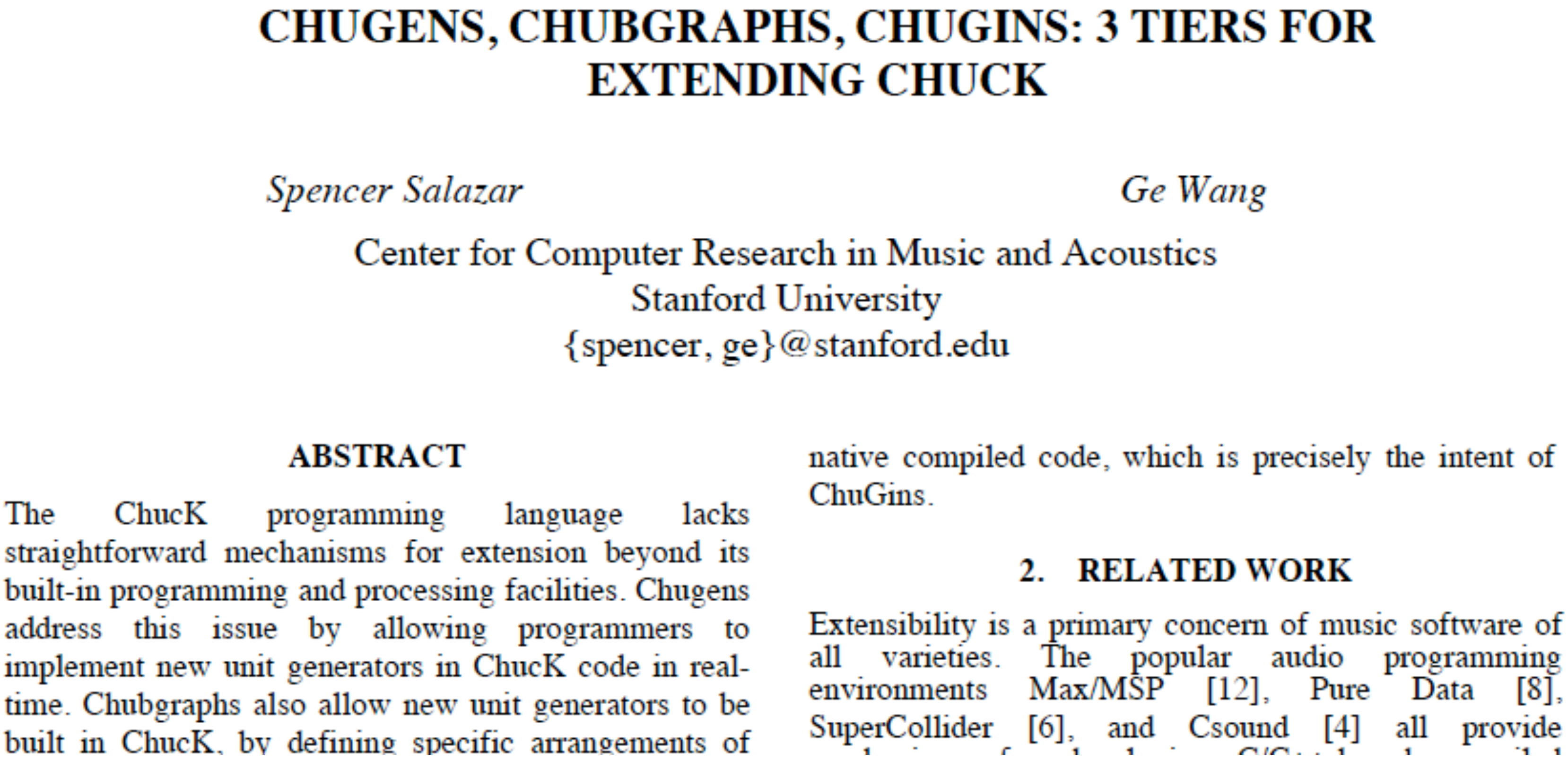
Extending ChucK

Perry Cook, Princeton, SMule, Chuck Co-creator, ...



CHUGENS, CHUBGRAPHS, CHUGINS: 3 TIERS FOR EXTENDING CHUCK

Spencer Salazar *Ge Wang*

Center for Computer Research in Music and Acoustics
Stanford University
{spencer, ge}@stanford.edu

ABSTRACT

The ChucK programming language lacks straightforward mechanisms for extension beyond its built-in programming and processing facilities. Chugens address this issue by allowing programmers to implement new unit generators in ChucK code in real-time. Chubgraphs also allow new unit generators to be built in ChucK, by defining specific arrangements of

native compiled code, which is precisely the intent of ChuGins.

2. RELATED WORK

Extensibility is a primary concern of music software of all varieties. The popular audio programming environments Max/MSP [12], Pure Data [8], SuperCollider [6], and Csound [4] all provide

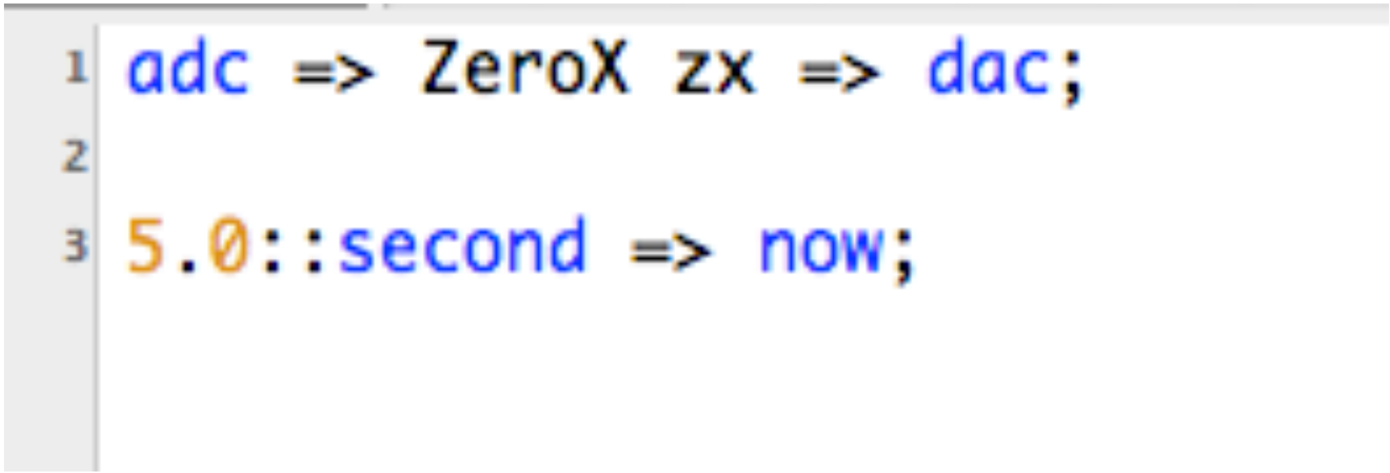
Signal Processing Native in ChucK

```
// adc => blackhole;              // DON'T DO THIS! (it stays connected)
adc => Gain input => blackhole;  // lets us look at individual samples
Impulse output => dac;           // Lets us write out individual samples

while (1)  {
    if (input.last() > 0.9) <<< now, input.last(), "LOUD!!" >>>;
    input.last() => output.next;      // copy input to output if you like
    1.0 :: samp => now;               // every sample
}
```

Zero Crossing Native in ChucK (raw language)

```
// adc => blackhole;             // DON'T DO THIS! (it stays connected)
adc => Gain input => blackhole;  // lets us look at individual samples
Impulse output => dac;           // Lets us write out individual samples

0 => int ZCs;       // place to count our zeros
0.0 => float lastIn;
0 => int counter;
22050 => int FRAME; // How often to update

while (1)  {
    if (lastIn < 0.0 & input.last() >= 0.0) {
        1 +=> ZCs;
    }
    1 +=> counter;
    if (counter == FRAME) {
        <<< "Number of Zero Crossings:", ZCs >>>;
        0 => counter;
        0 => ZCs;
    }
    input.last() => lastIn => output.next; // input to output (if needed)
    1.0 :: samp => now;               // every sample
}
```

Zero Crossing UG in ChucK



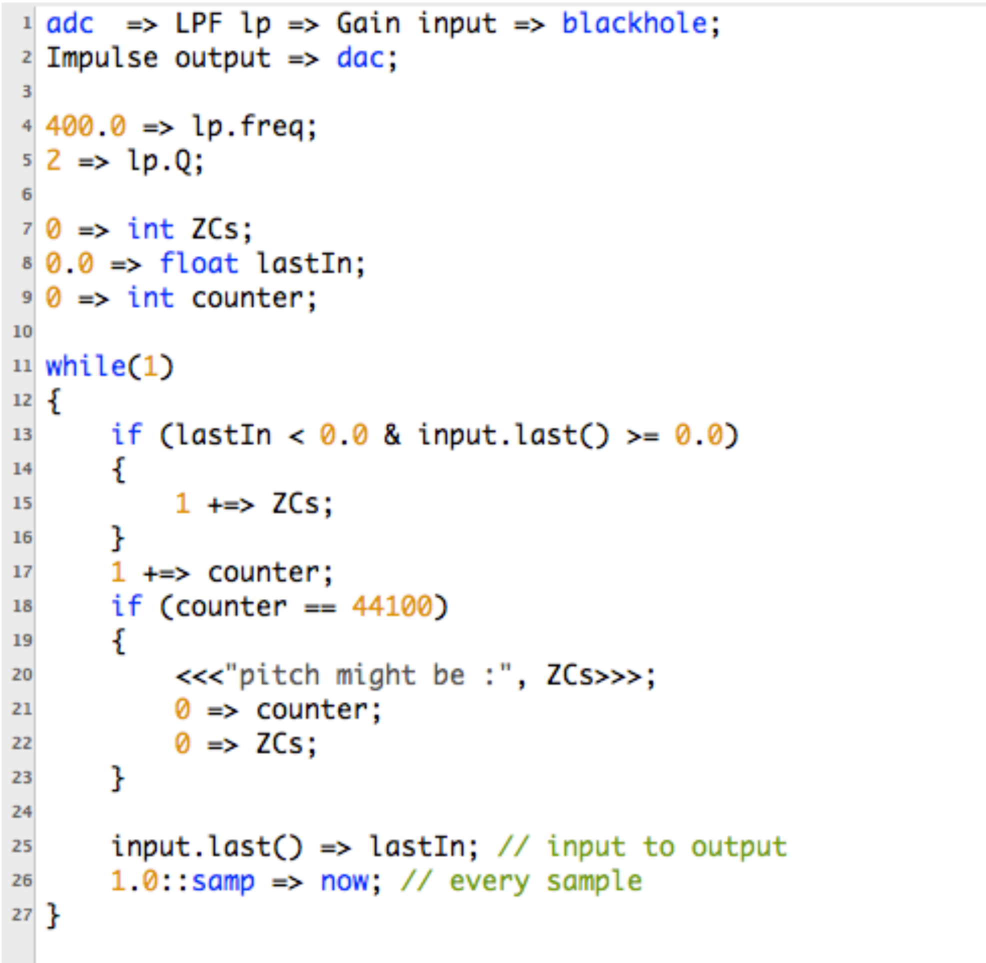
```
adc => ZeroX zx => dac;

5.0::second => now;
```

This puts out a +1.0 for positive zero crossing, and -1.0 for negative (not too useful except to make rasty sounds)

ZeroX to count Zero Xings

```
adc => ZeroX z => FullRect f => OnePole p => blackhole; // basic patch
0.0 => float myZeroes;          // This holds the last zero crossing count

spork ~ ZC(0.100);

while (1)       {
    100.0 :: second => now;  // do other stuff here
}

fun void ZC(float howoften)     {
    howoften*44100 - 1 => float wait;
    while (1)   {
        wait :: samp => now;
        p.last() => myZeroes;
    <<< "Zero Crossings = ", myZeroes >>>;
        0.0 => p.a1;
        0.0 => p.b0;
        1 :: samp => now;
        -1.0 => p.a1;   // this is dangerous, but we
        1.0 => p.b0;    // know what we're doing
    }
}
```

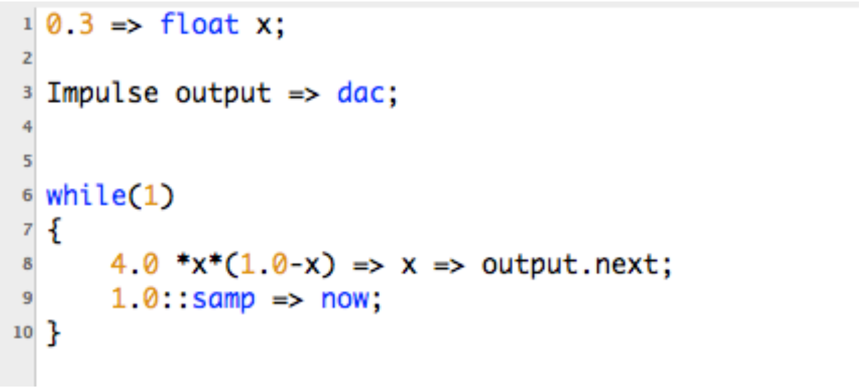
Zero Crossing in ChucK (with LP Filter)



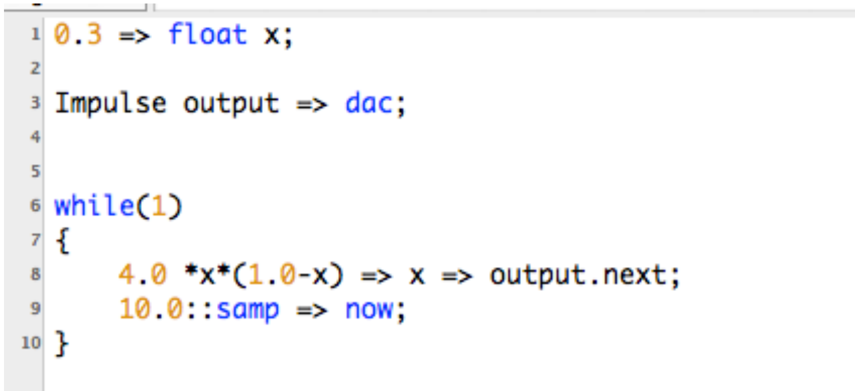
```
adc  => LPF lp => Gain input => blackhole;
Impulse output => dac;

400.0 => lp.freq;
2 => lp.Q;

0 => int ZCs;
0.0 => float lastIn;
0 => int counter;

while(1)
{
    if (lastIn < 0.0 & input.last() >= 0.0)
    {
        1 +=> ZCs;
    }
    1 +=> counter;
    if (counter == 44100)
    {
        <<<"pitch might be :", ZCs>>>;
        0 => counter;
        0 => ZCs;
    }

    input.last() => lastIn; // input to output
    1.0::samp => now; // every sample
}
```

Can add more filters in series

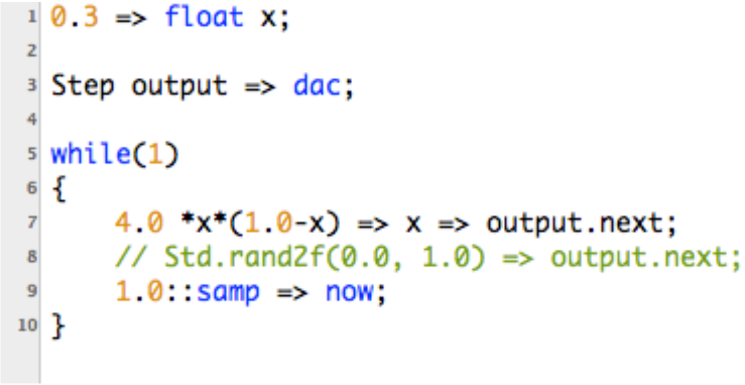
DIY Native Chaos Noise Unit Generator



```
0.3 => float x;

Impulse output => dac;


while(1)
{
    4.0 *x*(1.0-x) => x => output.next;
    1.0::samp => now;
}
```

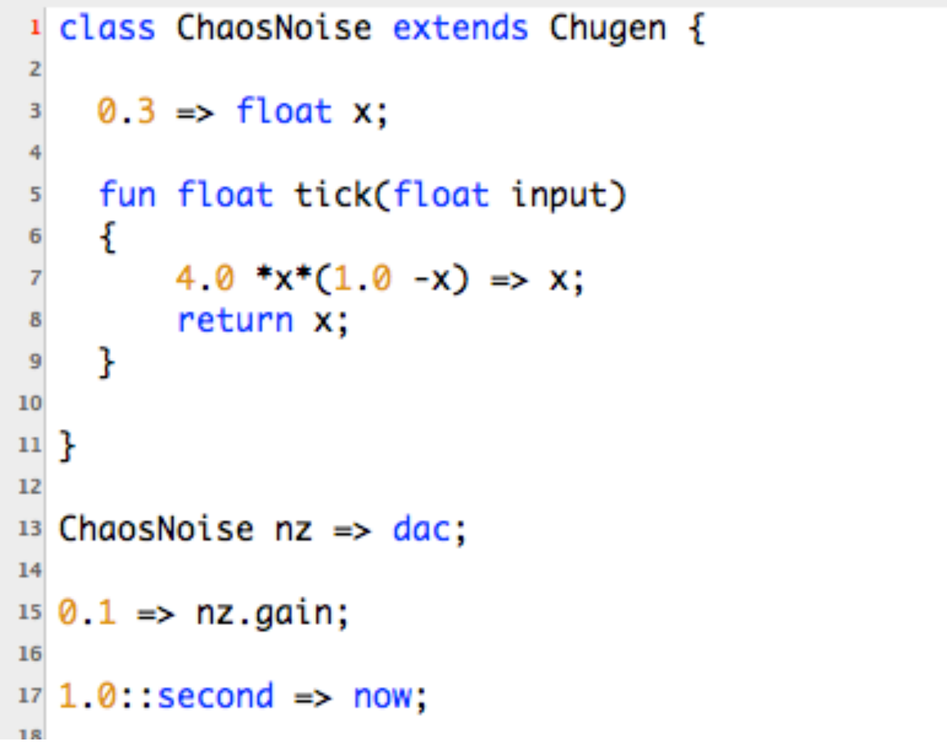


```
0.3 => float x;

Impulse output => dac;


while(1)
{
    4.0 *x*(1.0-x) => x => output.next;
    10.0::samp => now;
}
```



```
0.3 => float x;

Step output => dac;

while(1)
{
    4.0 *x*(1.0-x) => x => output.next;
    // Std.rand2f(0.0, 1.0) => output.next;
    1.0::samp => now;
}
```

Chaos Noise Chugen



```
class ChaosNoise extends Chugen {

  0.3 => float x;

  fun float tick(float input)
  {
      4.0 *x*(1.0 -x) => x;
      return x;
  }

}

ChaosNoise nz => dac;

0.1 => nz.gain;

1.0::second => now;
```

Zero Crossing Chugen

```
// ChuGen!    Now we can create new UGens, in ChucK
// to perform audio-rate processing in ChucK

class ZeroCrossings extends Chugen  {
    (1.0 :: second / 1.0 :: samp) $ int => int SRATE;
    SRATE => int frame;  // how often to update
    0.0 => float lastIn;
    0 => int ZCount;
    0 => int myZeroes;
    0 => int counter;

    fun float tick(float in)   {   // <<<===  Here's the important action
        if (lastIn < 0.0 & in >= 0.0) {
            1 +=> ZCount;
        }
        in => lastIn;
        1 +=> counter;
        if (counter >= frame) {
            ZCount => myZeroes;
            myZeroes * SRATE / frame => myZeroes; // per second (pitch)
            0 => ZCount => counter;
        }
        return in;  // might as well, right?
    }
```

continued next slide...

ZC Chugen continued

```
    fun int zeroes()  {
        return myZeroes;
    }

    fun void setFrame(int frameSamps)  {
        frameSamps => frame;
    }
}

adc => ZeroCrossings zcs => blackhole;
5000 => zcs.frame;

while (1)  {
    5000 :: samp => now;
    <<< zcs.zeroes() >>>;
}
```

end class def

now test and use it!!

ChuGen LFO

```
// Simple tabulated sine LFO
// compute and fill table only once
// P. Cook, March 2013

class LFO extends Chugen {
    128 => int VECT;
    float sine[VECT];
    0.01 => float myFreq;
    0.0 => float count;

    for (0 => int i; i < VECT; i++)
        Math.sin(i*2*pi/VECT) => sine[i];

    fun float tick(float input) {
        // here we ignore the input
        myFreq +=> count;
        while (count >= VECT)  VECT -=> count;
        count $ int => int counter;
        return sine[counter];
    }

    fun void freq(float aFreq) {
        aFreq * VECT / (1.0 :: second / 1.0 :: samp) => myFreq;
    }
}
```

Using the LFO Chugen

```
Step stp => Gain freq => SinOsc s => dac;
LFO lfo => freq;
1.0 => stp.next;
6.0 => lfo.freq;
0.05 => lfo.gain;
2 => s.sync;

300.0 => freq.gain;
1.0 :: second => now;
400.0 => freq.gain;
1.0 :: second => now;
500.0 => freq.gain;
1.0 :: second => now;
```

Chubgraph: build UGs using UGs

```
class WigglySine extends Chubgraph    {
    Step stp => Gain frq => SinOsc s => dac;
    LFO lfo => frq;
    1.0 => stp.next;
    6.0 => lfo.freq;
    0.05 => lfo.gain;
    2 => s.sync;
    300.0 => frq.gain;

    fun float freq( float f)  {
        f => frq.gain;
    }
}

WigglySine ws;
300.0 => ws.freq;
1.0 :: second => now;
400.0 => ws.freq;
1.0 :: second => now;
500.0 => ws.freq;
1.0 :: second => now;
```

Fuzz Chugen

```
// ChuGen
// Create new UGens by performing audio-rate processing in ChucK

class Fuzz extends Chugen      {
    3.0 => float p;
    2 => intensity;

    fun float tick(float in)    {
        Math.sgn(in) => float sgn;
        return Math.pow(Math.fabs(in), p) * sgn;
    }

    fun void intensity(float i)  {
        if(i > 1)
            1.0/i => p;
    }
}

adc => Fuzz f => dac;
2.5 => f.intensity;

while(true) 1::second => now;
```

ChubGraph - MyString

```
// Chubgraph
// Create new UGens by compositing existing UGens

class MyString extends Chubgraph    {
    // karplus + strong plucked string filter
    // Ge Wang (gewang@cs.princeton.edu)

    Noise imp => OneZero lowpass => dac;
    lowpass => DelayA delay => lowpass;

    .99999 => float R;
    1/220 => float L;
    -1 => lowpass.zero;
    220 => freq;
    0 => imp.gain;

    fun float freq( float f )     {
        1/f => L;
        L::second => delay.delay;
        Math.pow( R, L ) => delay.gain;
        return f;
    }

    fun void pluck()     {
        1 => imp.gain;
        L::second => now;
        0 => imp.gain;
        (Math.log(.0001) / Math.log(R))::samp => now;
    }
}
```

Using MyString

```
MyString s[3];
for(int i; i < s.cap(); i++) s[i] => dac;

while( true )
{
    for( int i; i < s.cap(); i++ )
    {
        Math.rand2( 60,72 ) => Std.mtof => s[i].freq;
        spork ~ s[i].pluck();
        0.25::second => now;
    }

    2::second => now;
}
```

ChubGraph - MandoPlayer

```
class MandoPlayer extends Chubgraph  {
    // Four Mando "strings", plus lots of smarts
    // by Perry R. Cook, March 2013

    Mandolin m[4];
    m[0] => JCRev rev => dac; m[0].freq(Std.mtof(55));
    m[1] => rev; m[0].freq(Std.mtof(62));
    m[2] => rev; m[0].freq(Std.mtof(69));
    m[3] => rev; m[0].freq(Std.mtof(76));
    0.02 => rev.mix;

    fun void freqs(float gString, float aString, float dString, float eString)

    fun void notes(int gNote, int aNote, int dNote, int eNote)  {
        ...
    fun void strum(dur rate) {
        ...
    fun void damp(float amount)  { // 0.0 = lots of damping, 1.0 = none
        ...
    fun void chord(string which)  {
        ...

    }

    MandoPlayer m;

    m.chord("G");
    m.strum(0.4 :: second);
    m.chord("D");  m.strum(0.4 :: second);
    m.strum(0.1 :: second);
    m.chord("G");  m.strum(0.4 :: second);
    ...
```

Chugins: Your own UG Classes!

Warnings!!

1) Awesome
2) Pain in the Ass
3) Must have them in right place

Coolness:

1) Lets you extend ChucK any way
2) User/Community contributed
3) (can be) Most efficient of all

Simple Examples: BitCrusher, ZCs

Useful Examples: FIR, MAUI, TDFeatures

Chugins: Simple ZCs Class

```
#include "chuck_dl.h"
#include "chuck_def.h"

#include <stdio.h>
#include <limits.h>
#include <math.h>

CK_DLL_CTOR(ZCs_ctor);
CK_DLL_DTOR(ZCs_dtor);

CK_DLL_MFUN(ZCs_setFrame);      // set frame, allocate memory (default 4410)
CK_DLL_MFUN(ZCs_getFrame);      // query frame size, in samples
CK_DLL_MFUN(ZCs_getZCs);        // get all zero crossings in frame / 2

CK_DLL_TICK(ZCs_tick);

t_CKINT ZCs_data_offset = 0;

struct ZCsData
{
    int frame;  // frame, size of buffer, update this often
    int count;  // where are we in frame, circular buffer pointer

    int zcs;     // number of positive-going ZCs in last frame
    int zcsTemp;    // temp accumulator

    float lastIn;       // last input sample
};
```

Chugins: Simple ZCs Class

```
CK_DLL_QUERY(ZCs)
{
    QUERY->setname(QUERY, "ZCs");
    QUERY->begin_class(QUERY, "ZCs", "UGen");

    QUERY->add_ctor(QUERY, ZCs_ctor);
    QUERY->add_dtor(QUERY, ZCs_dtor);

    QUERY->add_ugen_func(QUERY, ZCs_tick, NULL, 1, 1);

    QUERY->add_mfun(QUERY, ZCs_setFrame, "int", "frame");
    QUERY->add_arg(QUERY, "int", "arg");

    QUERY->add_mfun(QUERY, ZCs_getFrame, "int", "frame");

    QUERY->add_mfun(QUERY, ZCs_getZCs, "float", "ZCs");

    ZCs_data_offset = QUERY->add_mvar(QUERY, "int", "@lpc_data", false);

    QUERY->end_class(QUERY);

    return TRUE;
}
```

Chugins: Simple ZCs Class

```
CK_DLL_CTOR(ZCs_ctor)
{
    BLAH BLAH BLAH   }

CK_DLL_DTOR(ZCs_dtor)
{
   BLAH BLAH BLAH  }

CK_DLL_TICK(ZCs_tick)
{
    ZCsData * tdfdata = (ZCsData *) OBJ_MEMBER_INT(SELF, ZCs_data_offset);

    *out = in; // first things first;

    if (in > 0.0 && tdfdata->lastIn<= 0.0) {
        tdfdata->zcsTemp++;
    }

    tdfdata->count++;

    if (tdfdata->count >= tdfdata->frame) {
        tdfdata->zcs = tdfdata->zcsTemp;
        tdfdata->zcsTemp = 0;
        tdfdata->count = 0;
    }

    tdfdata->lastIn = in;

    return TRUE;
}
```

Chugins: Simple ZCs Class

```
CK_DLL_MFUN(ZCs_setFrame)
{
   ZCsData * tdfdata = (ZCsData *) OBJ_MEMBER_INT(SELF, ZCs_data_offset);
   // TODO: sanity check

   tdfdata->frame = GET_NEXT_INT(ARGS);

   RETURN->v_int = tdfdata->frame;
}

CK_DLL_MFUN(ZCs_getFrame)
{
   ZCsData * tdfdata = (ZCsData *) OBJ_MEMBER_INT(SELF, ZCs_data_offset);
   RETURN->v_int = tdfdata->frame;
}

CK_DLL_MFUN(ZCs_getZCs)
{
   ZCsData * tdfdata = (ZCsData *) OBJ_MEMBER_INT(SELF, ZCs_data_offset);
   RETURN->v_float = tdfdata->zcs;
}
```

Chugins: Class TDFeatures

(time-domain features)

at (common) frame rate .frame(int samps)(default 4410)

* +/- peaks

.ppeak() (positive peak)

.npeak() (negative peak)

.peak() (max abs peak)

* power=(Sum(x^2))/frame .power()

* RMS Power (Sqrt(power) .RMS()

* Zero Crossings:

(+going and -going)

(with +/- Hysteresis)

.ZCs (average (pos+neg)/2)

.ZCsp(positive going)

.ZCsn(negative going)

.hyst(float hystpoint)

(fraction of peaks, 0.0-1.0, default 0)

* pitch= srate * ZCs / frame .pitch() also needs .srate(int rate)

default = 44100