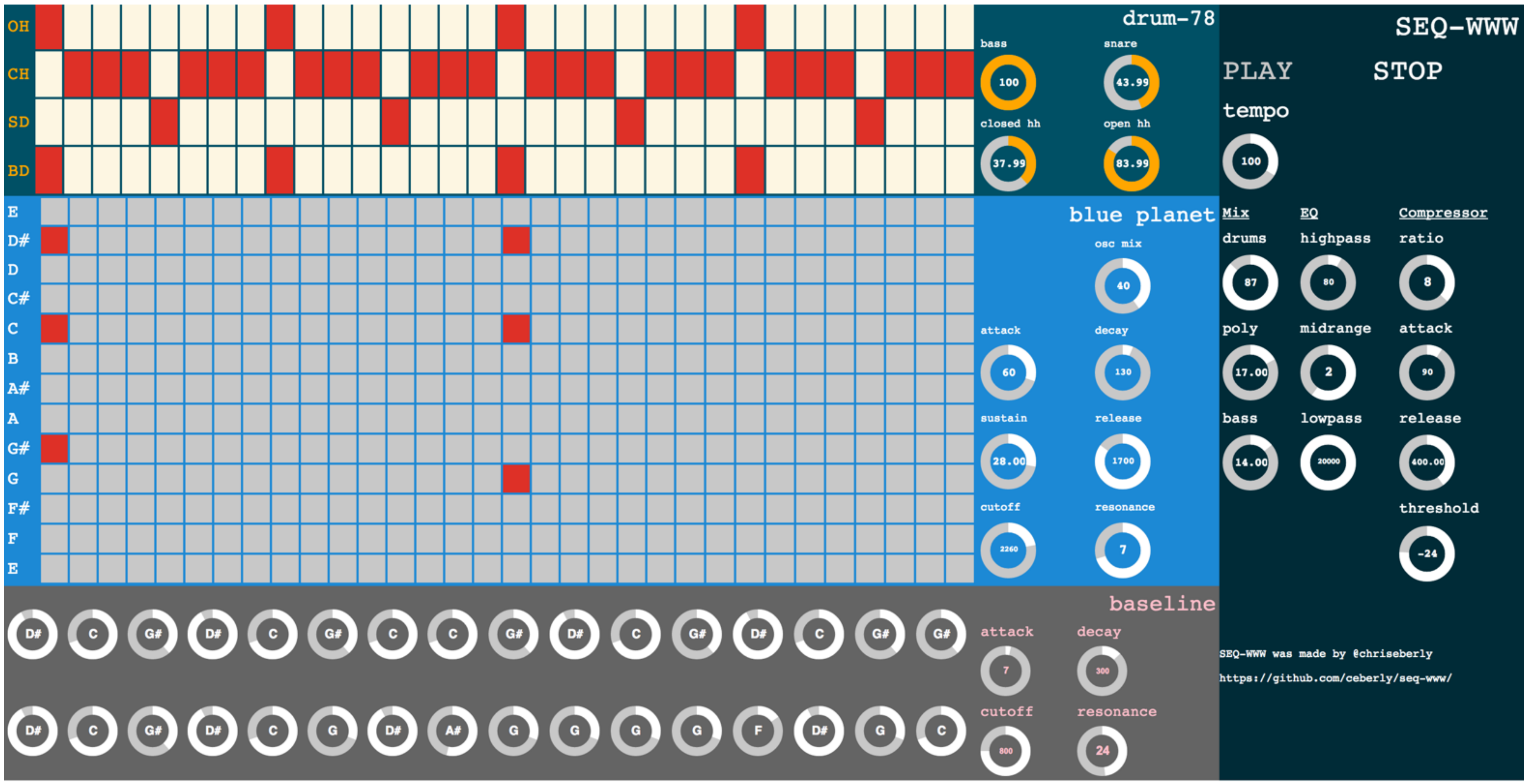
Analog style synthesis with the web audio API

Chris Eberly - Gobbler engineering -
chris.eberly@gobbler.com - @chriseberly

seqwww.io



https://github.com/ceberly/seq-www

Subtractive synthesis in 20 or so lines

```
window.AudioContext = window.AudioContext || window.webkitAudioContext;
var context = new AudioContext();

function run(context, now, attack, release, cutoff, resonance) {
  var lp = context.createBiquadFilter();
  var osc = context.createOscillator();
  var env = context.createGain();

  var freq = 120.0;

  osc.connect(env);
  env.connect(lp);
  lp.connect(context.destination);

  lp.frequency.value = cutoff;
  lp.Q.value = resonance;

  osc.start(now);
  osc.type = "sawtooth";
  env.gain.value = 0;

  env.gain.setTargetAtTime(1, now, attack / 1000.0);
  env.gain.setTargetAtTime(0, now + attack/1000.0, release/1000.0);
}
```

Synthesizing TR-808 style cymbals - design

CY

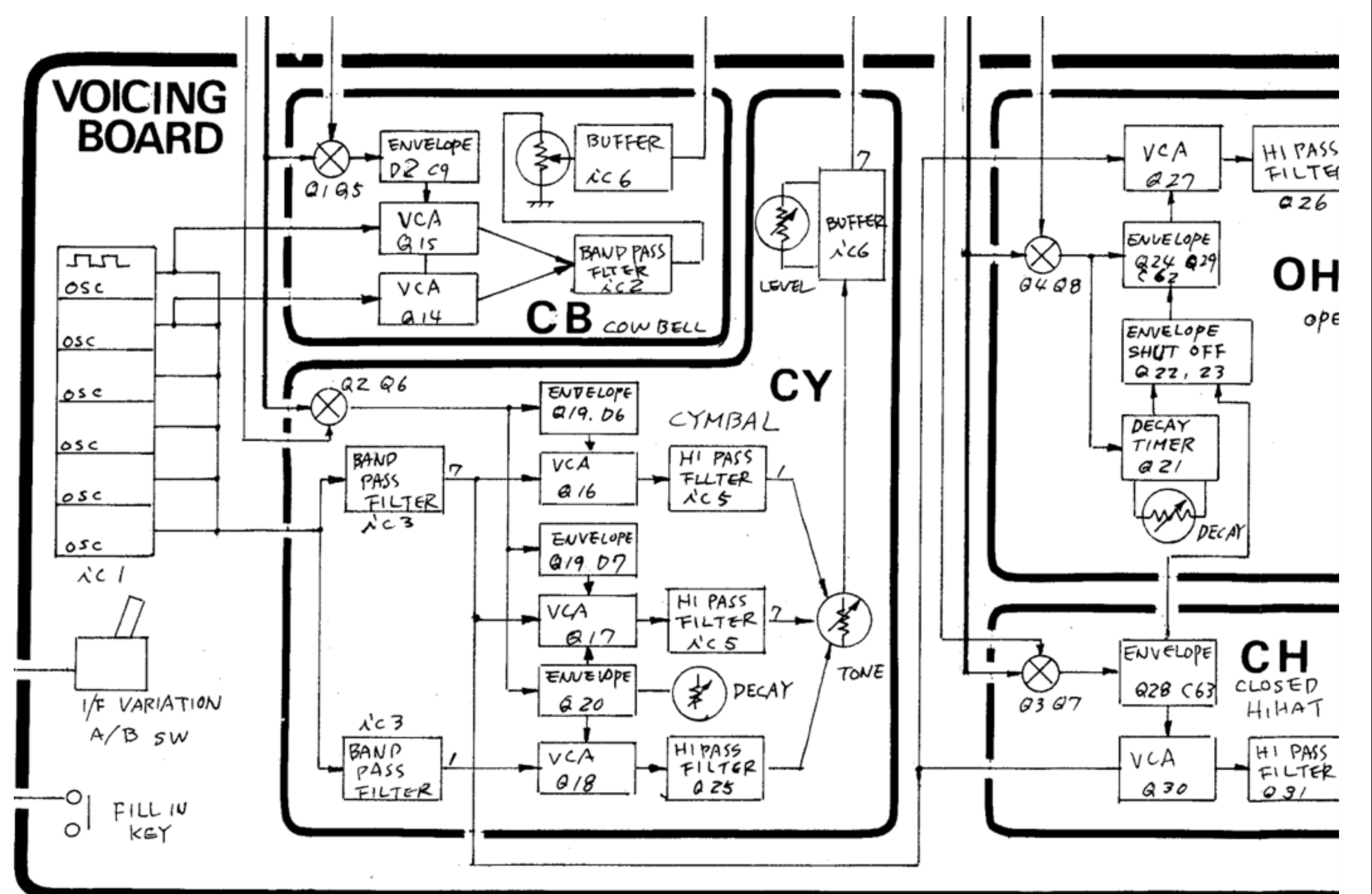
The combined square wave outputs of six Schumitt triggers including two for CB generator is separated into high and low range components by two filters composed of IC3. The high range component from pin 7 of IC3 is further separated into two frequency ranges. The output of the gate Q16 has the highest frequency component of this sound generator. Its decay time is short. The output of Q17 is in a frequency range slightly lower than the above output, and its decay time is controllable.
These three signals with different frequency ranges are outputted with their level ratio controlled by VR4.

OH

The high frequency range component signal obtained by the above 1/2 IC3 is gated by Q27 and supplied to the buffer IC7 through the filter Q26. When the CLOSED HI-HAT (CH) is triggered while the OH circuit is activated, Q23 turns on by the voltage applied through R173. At this moment, the decay time of the OH circuit terminates.

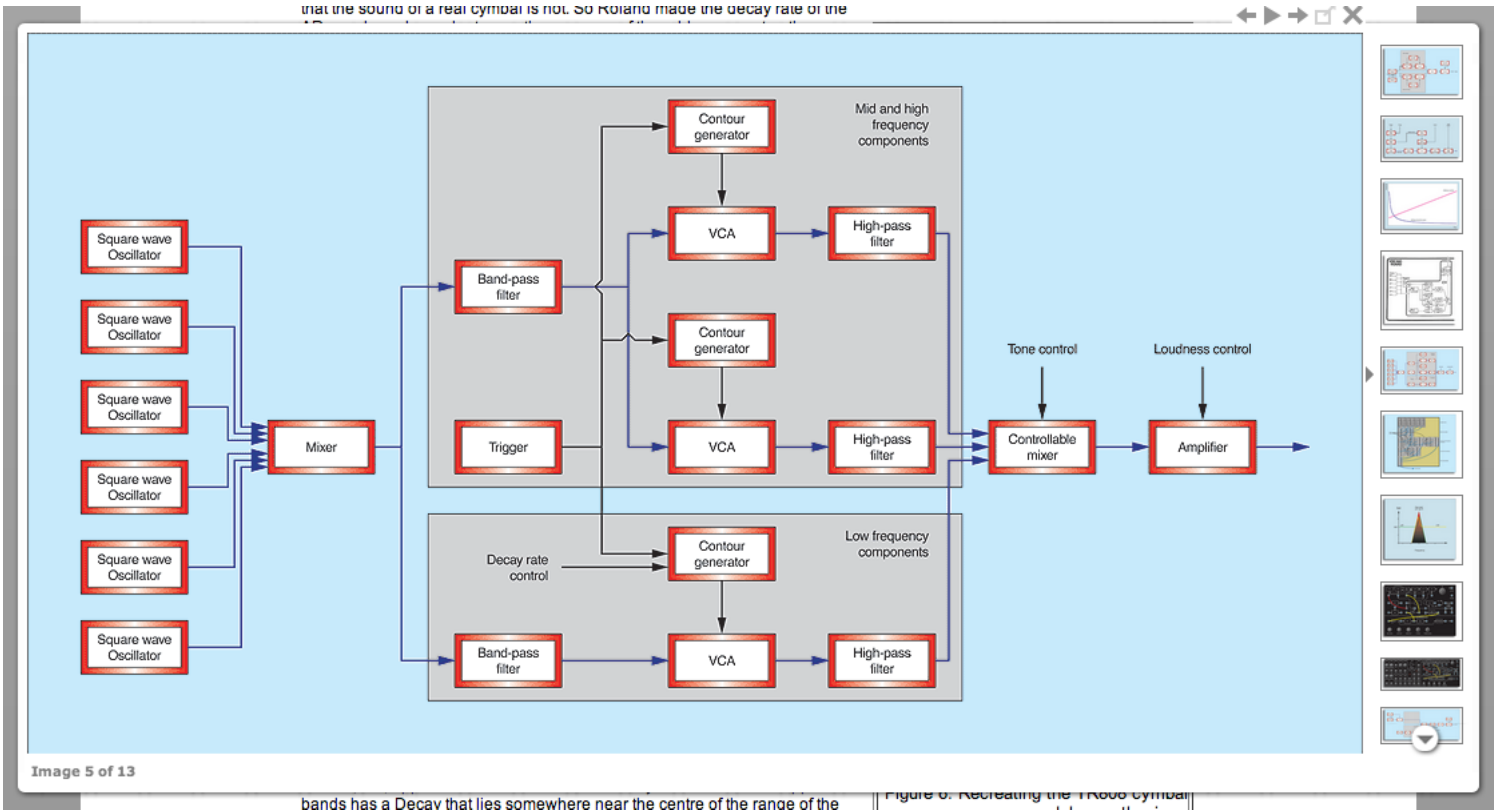
CH

This shares the same sound source with the OH. The signal is gated by Q30 and supplied to the filter Q31 and the buffer IC7 (1/2).



source: http://ericarcher.net/wp-content/uploads/2009/11/808-svc-man.pdf

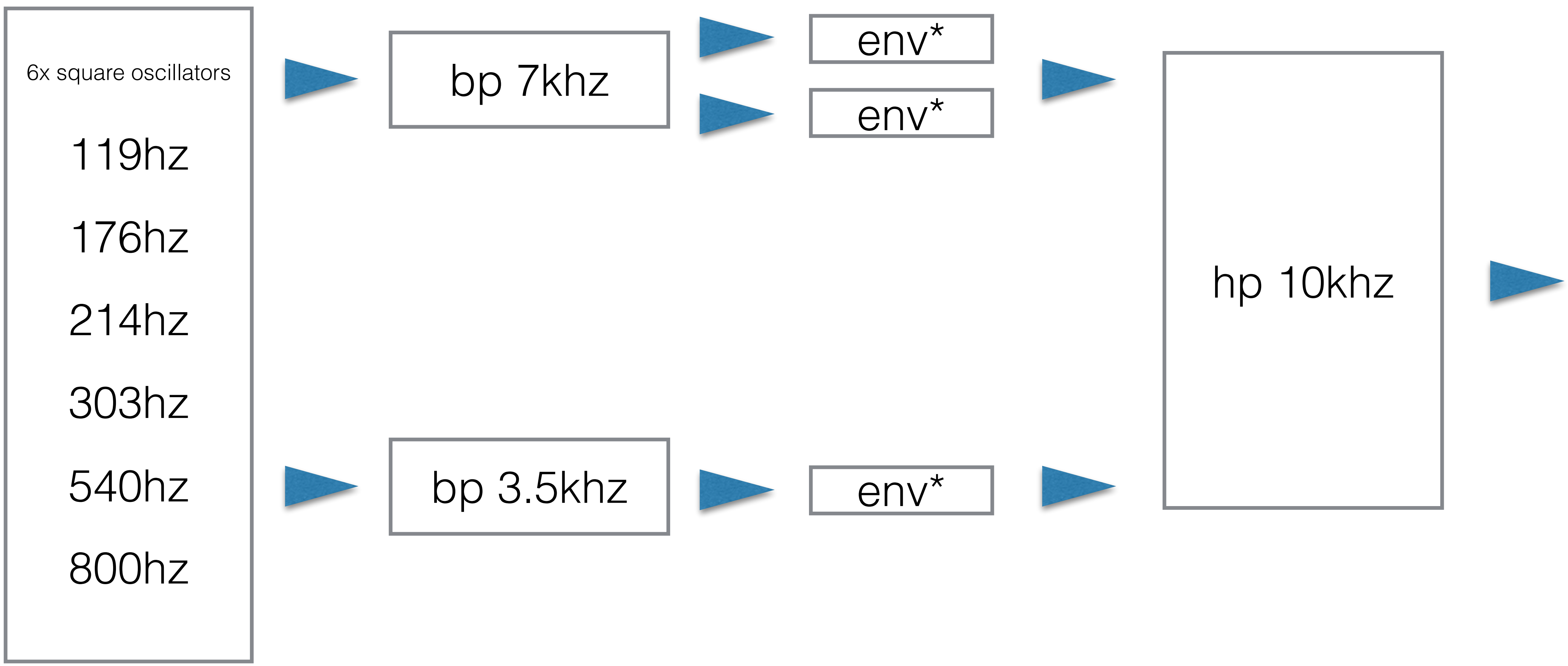
Synthesizing TR-808 style cymbals - design



source: http://www.soundonsound.com/sos/Jul02/articles/synthsecrets0702.asp

Synthesizing TR-808 style cymbals - design

here's how seqwww does it:



* The previous slide refers to this as a “Contour generator”. Our version simply controls attack and decay times.

Synthesizing TR-808 style cymbals - filter section

```
// high pass filter connected to the two cymbal gates
var chHP = context.createBiquadFilter();
chHP.connect(chGate);
chHP.connect(ohGate);
chHP.type = "highpass";
chHP.frequency.value = 10000;

// high pass filter is fed by two gain elements that act as envelope generators.
var chHGain1 = context.createGain();
chHGain1.connect(chHP);
chHGain1.gain.value = 0;

var chHGain2 = context.createGain();
chHGain2.connect(chHP);
chHGain2.gain.value = 0;

var chHBP = context.createBiquadFilter();
chHBP.type = "bandpass";
chHBP.frequency.value = 7000;
chHBP.Q.value = 12;
chHBP.connect(chHGain1);
chHBP.connect(chHGain2);

// low bandpass filter
var chLBGain = context.createGain();
chLBGain.connect(chHP);
chLBGain.gain.value = 0;

var chLBP = context.createBiquadFilter();
chLBP.type = "bandpass";
chLBP.frequency.value = 3500;
chLBP.Q.value = 12;
chLBP.connect(chLBGain);
```

Synthesizing TR-808 style cymbals - oscillator section

```
var chOsc1 = context.createOscillator();
chOsc1.type = "square";
chOsc1.frequency.value = 303;
chOsc1.connect(chLBP);
chOsc1.connect(chHBP);
chOsc1.start(0);

var chOsc2 = context.createOscillator();
chOsc2.type = "square";
chOsc2.frequency.value = 176;
chOsc2.connect(chLBP);
chOsc2.connect(chHBP);
chOsc2.start(0);

var chOsc3 = context.createOscillator();
chOsc3.type = "square";
chOsc3.frequency.value = 214;
chOsc3.connect(chLBP);
chOsc3.connect(chHBP);
chOsc3.start(0);

var chOsc4 = context.createOscillator();
chOsc4.type = "square";
chOsc4.frequency.value = 119;
chOsc4.connect(chLBP);
chOsc4.connect(chHBP);
chOsc4.start(0);

var chOsc5 = context.createOscillator();
chOsc5.type = "square";
chOsc5.frequency.value = 540;
chOsc5.connect(chLBP);
chOsc5.connect(chHBP);
chOsc5.start(0);

var chOsc6 = context.createOscillator();
chOsc6.type = "square";
chOsc6.frequency.value = 800;
chOsc6.connect(chLBP);
chOsc6.connect(chHBP);
chOsc6.start(0);
```

Synthesizing TR-808 style cymbals - envelope section

```
this.DrumMachine.CH.Trigger = function(at) {
      ohGate.gain.value = 0; // silence open hat
      chGate.gain.value = 1; // open closed hat gate

      chHGain1.gain.setTargetAtTime(1 / 3, at, 0);
      chHGain1.gain.setTargetAtTime(0, at + .01, .01);

      chHGain2.gain.setTargetAtTime(1 / 3, at, 0);
      chHGain2.gain.setTargetAtTime(0, at + .02, .01);

      chLBGain.gain.setTargetAtTime(1 / 3, at, 0);
      chLBGain.gain.setTargetAtTime(0, at + .02, .01);
};

this.DrumMachine.OH.Trigger = function(at) {
      chGate.gain.value = 0; // silence closed hat
      ohGate.gain.value = 1; // open open hat gate

      chHGain1.gain.setTargetAtTime(1 / 3, at, 0);
      chHGain1.gain.setTargetAtTime(0, at + .5, 1);

      chHGain2.gain.setTargetAtTime(1 / 3, at, 0);
      chHGain2.gain.setTargetAtTime(0, at + .5, 1);

      chLBGain.gain.setTargetAtTime(1 / 3, at, 0);
      chLBGain.gain.setTargetAtTime(0, at + .5, 1);
};
```

		AMPLITUDE		FREQUENCY			DECAY TIME		
		NORMAL	ACCENT	LOW	MID	HIGH	SHORT	MID	LONG
		Vpp	Vpp	ms (Hz)	ms (Hz)	ms (Hz)	ms	ms	ms
BD		3.5	10	—	18 (56)	—	50	300	800
SD	H	3	10	—	2.1 (476)	—	—	60	—
	L				4.2 (238)				
LC		3.5	12	6.1 (165)	5.4 (185)	4.5 (220)	—	180	—
LT		3.5	12	12.5 (80)	11.1 (90)	10 (100)	—	200	—
MC		3	10	4 (250)	3.6 (280)	3.2 (310)	—	100	—
MT		3	11	8.3 (120)	7.4 (135)	6.3 (160)	—	130	—
HC		3.5	12	2.7 (370)	2.5 (400)	2.2 (455)	—	80	—
HT		3.5	12	6.1 (165)	5.4 (185)	4.5 (220)	—	100	—
C		2.5	8	—	0.4 (2500)	—	—	25	—
RS	H	3	10	—	0.6 (1667)	—	—	10	—
	L				2.2 (455)				
M		3	5	—	—	—	25	—	35
CP		6	2	—	—	—	—	100	—
CB	H	3.5	12	—	1.25 (800)	—	—	50	—
	L				1.85 (540)				
CY		3.5	7	—	—	—	350	800	1200
OH		3.5	7	—	—	—	90	450	600
CH		3	6	—	—	—	—	50	—

values are typical and variable

source: http://ericarcher.net/wp-content/uploads/2009/11/808-svc-man.pdf

Web audio oscillator implementation in WebKit

- OWNERS
- OfflineAudioCompletionEvent.cpp
- OfflineAudioCompletionEvent.h
- OfflineAudioCompletionEvent.idl
- OfflineAudioContext.cpp
- OfflineAudioContext.h
- OfflineAudioContext.idl
- OfflineAudioDestinationNode.cpp
- OfflineAudioDestinationNode.h
- OscillatorNode.cpp
- OscillatorNode.h
- OscillatorNode.idl
- PannerNode.cpp
- PannerNode.h
- PannerNode.idl
- PeriodicWave.cpp
- PeriodicWave.h

```
while (n--) {
    unsigned readIndex = static_cast<unsigned>(virtualReadIndex);
    unsigned readIndex2 = readIndex + 1;

    // Contain within valid range.
    readIndex = readIndex & readIndexMask;
    readIndex2 = readIndex2 & readIndexMask;

    if (hasSampleAccurateValues) {
        incr = *phaseIncrements++;

        frequency = invRateScale * incr;
        m_periodicWave->waveDataForFundamentalFrequency(frequency, lowerWaveData, higherWaveData, tableInterpolationFactor);
    }

    float sample1Lower = lowerWaveData[readIndex];
    float sample2Lower = lowerWaveData[readIndex2];
    float sample1Higher = higherWaveData[readIndex];
    float sample2Higher = higherWaveData[readIndex2];

    // Linearly interpolate within each table (lower and higher).
    float interpolationFactor = static_cast<float>(virtualReadIndex) - readIndex;
```

https://code.google.com/p/chromium/codesearch#chromium/src/third_party/WebKit/Source/modules/webaudio/&q=createoscillator&sq=package:chromium

Web audio oscillator implementation in WebKit

- OfflineAudioContext.cpp
- OfflineAudioContext.h
- OfflineAudioContext.idl
- OfflineAudioDestinationNode.cpp
- OfflineAudioDestinationNode.h
- OscillatorNode.cpp
- OscillatorNode.h
- OscillatorNode.idl
- PannerNode.cpp
- PannerNode.h
- PannerNode.idl
- PeriodicWave.cpp
- PeriodicWave.h
- PeriodicWave.idl
- RealtimeAnalyser.cpp
- RealtimeAnalyser.h
- ScriptProcessorNode.cpp
- ScriptProcessorNode.h
- ScriptProcessorNode.idl
- WaveShaperDSPKernel.cpp
- WaveShaperDSPKernel.h
- WaveShaperNode.cpp
- WaveShaperNode.h
- WaveShaperNode.idl
- WaveShaperProcessor.cpp

```cpp
// Convert into time-domain wave buffers.
// One table is created for each range for non-aliasing playback at different playback rates.
// Thus, higher ranges have more high-frequency partials culled out.
void PeriodicWave::createBandLimitedTables(const float* realData, const float* imagData, unsigned numberOfComponents)
{
    float normalizationScale = 1;

    unsigned fftSize = m_periodicWaveSize;
    unsigned halfSize = fftSize / 2;
    unsigned i;

    numberOfComponents = std::min(numberOfComponents, halfSize);

    m_bandLimitedTables.reserveCapacity(m_numberOfRanges);

    for (unsigned rangeIndex = 0; rangeIndex < m_numberOfRanges; ++rangeIndex) {
        // This FFTFrame is used to cull partials (represented by frequency bins).
        FFTFrame frame(fftSize);
        float* realP = frame.realData();
        float* imagP = frame.imagData();

        // Copy from loaded frequency data and scale.
        float scale = fftSize;
        vsmul(realData, 1, &scale, realP, 1, numberOfComponents);
        vsmul(imagData, 1, &scale, imagP, 1, numberOfComponents);

        // If fewer components were provided than 1/2 FFT size, then clear the remaining bins.
        for (i = numberOfComponents; i < halfSize; ++i) {
            realP[i] = 0;
            imagP[i] = 0;
        }

        // Generate complex conjugate because of the way the inverse FFT is defined.
```

https://code.google.com/p/chromium/codesearch#chromium/src/third_party/WebKit/Source/modules/webaudio/&q=createoscillator&sq=package:chromium

Web audio oscillator implementation in WebKit

OfflineAudioContext.cpp
OfflineAudioContext.h
OfflineAudioContext.idl
OfflineAudioDestinationNode.cpp
OfflineAudioDestinationNode.h
OscillatorNode.cpp
OscillatorNode.h
OscillatorNode.idl
PannerNode.cpp
PannerNode.h
PannerNode.idl
PeriodicWave.cpp
PeriodicWave.h
PeriodicWave.idl
RealtimeAnalyser.cpp
RealtimeAnalyser.h
ScriptProcessorNode.cpp
ScriptProcessorNode.h
ScriptProcessorNode.idl
WaveShaperDSPKernel.cpp
WaveShaperDSPKernel.h
WaveShaperNode.cpp
WaveShaperNode.h
WaveShaperNode.idl
WaveShaperProcessor.cpp

```
    // Generate complex conjugate because of the way the inverse FFT is defined.
    float minusOne = -1;
    vsmul(imagP, 1, &minusOne, imagP, 1, halfSize);

    // Find the starting bin where we should start culling.
    // We need to clear out the highest frequencies to band-limit the waveform.
    unsigned numberOfPartials = numberOfPartialsForRange(rangeIndex);

    // Cull the aliasing partials for this pitch range.
    for (i = numberOfPartials + 1; i < halfSize; ++i) {
        realP[i] = 0;
        imagP[i] = 0;
    }
    // Clear packed-nyquist if necessary.
    if (numberOfPartials < halfSize)
        imagP[0] = 0;

    // Clear any DC-offset.
    realP[0] = 0;

    // Create the band-limited table.
    OwnPtr<AudioFloatArray> table = adoptPtr(new AudioFloatArray(m_periodicWaveSize));
    m_bandLimitedTables.append(table.release());

    // Apply an inverse FFT to generate the time-domain table data.
    float* data = m_bandLimitedTables[rangeIndex]->data();
    frame.doInverseFFT(data);

    // For the first range (which has the highest power), calculate its peak value then compute normalization scale.
    if (!rangeIndex) {
        float maxValue;
        vmaxmgv(data, 1, &maxValue, m_periodicWaveSize);
```

https://code.google.com/p/chromium/codesearch#chromium/src/third_party/WebKit/Source/modules/webaudio/&q=createoscillator&sq=package:chromium

Thanks!

https://github.com/ceberly/seq-www

Chris Eberly - Gobbler engineering -
chris.eberly@gobbler.com - @chriseberly