COCOON

Sonic College 2024
Jakob Schmid

What is COCOON?

A puzzle adventure game

Geometric Interactive

Director:

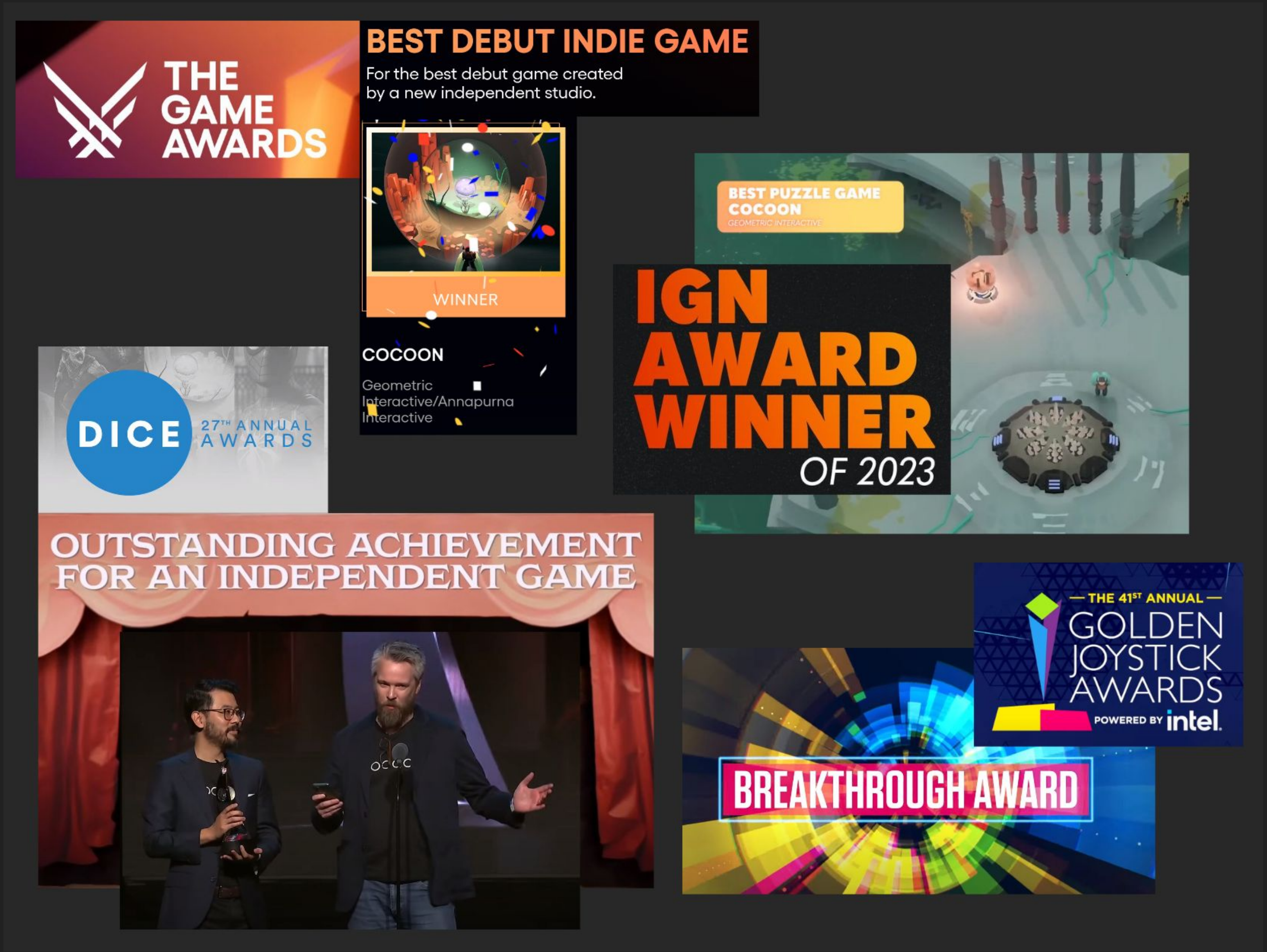
Jeppe Carlsen

Art director:

Erwin Kho

Production time: 6.5 years

Accolades


THE GAME AWARDS
BEST DEBUT INDIE GAME
For the best debut game created by a new independent studio.
WINNER
COCOON
Geometric Interactive/Annapurna Interactive
BEST PUZZLE GAME
COCOON
GEOMETRIC INTERACTIVE
IGN AWARD WINNER OF 2023
DICE 27TH ANNUAL AWARDS
OUTSTANDING ACHIEVEMENT FOR AN INDEPENDENT GAME
THE 41ST ANNUAL GOLDEN JOYSTICK AWARDS POWERED BY intel
BREAKTHROUGH AWARD

COCOON Audio Team

Audio direction / music:

Jakob Schmid

Sound design:

Julian Lentz
Mikkel Anttila

- both from Sonic College!

Music Concept

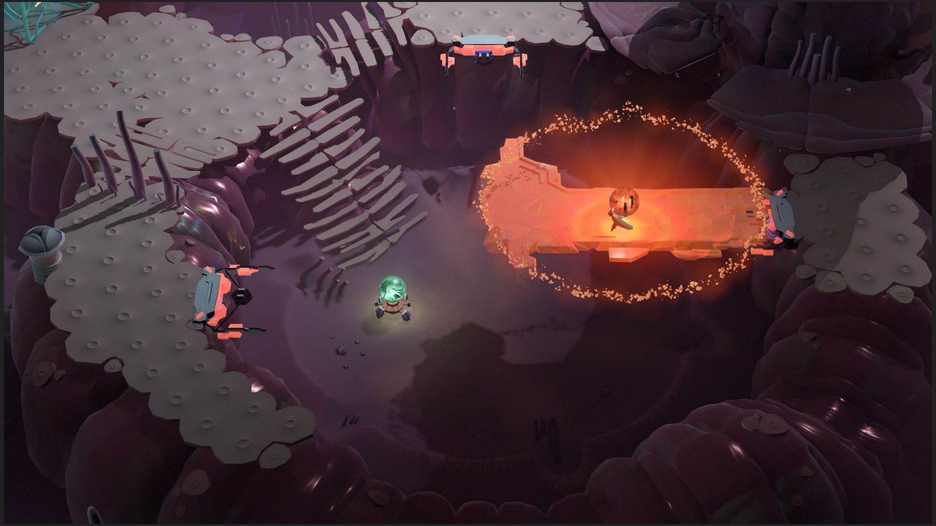
Generative music using real-time synthesis

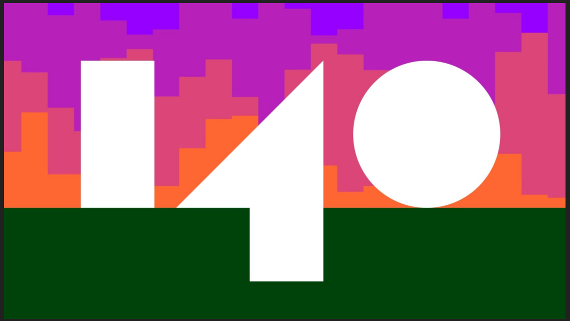
- Loop free during ‘thinking breaks’
- Unique soundtrack for each player

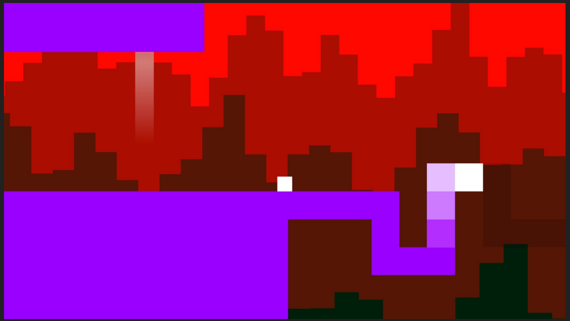
Sound Design Concept

Synthetic sound design - no recorded sound!

- Fits aesthetics of generative music
- Fits Erwin's art style: artificial but alive
- Familiar process from '140'

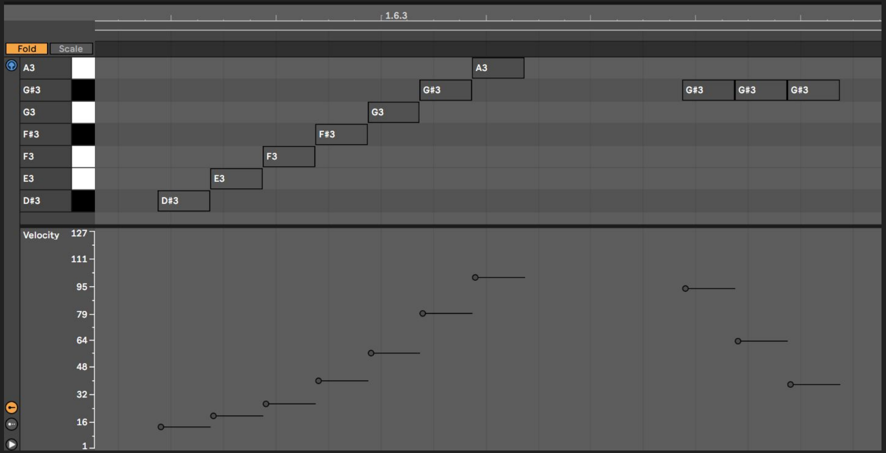


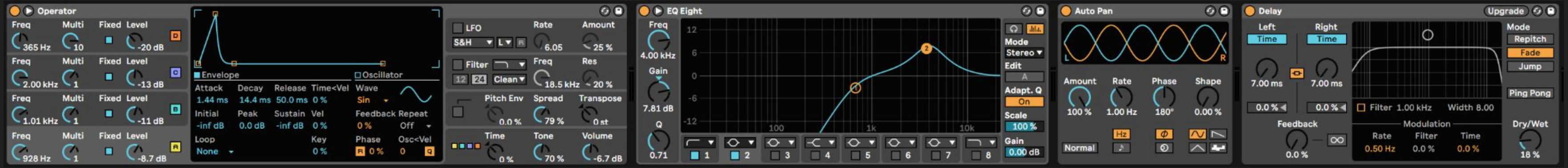




Synthetic Sound Design Experiments

Frogs, footsteps, portals





Operator

Freq	Multi	Fixed	Level
365 Hz	10		-20 dB
2.00 kHz	1		-13 dB
1.01 kHz	1		-11 dB
928 Hz	1		-8.7 dB

Envelope: Attack 1.44 ms, Decay 14.4 ms, Release 50.0 ms, Time<Vel 0 %; Initial -inf dB, Peak 0.0 dB, Sustain -inf dB, Vel 0 %; Loop None, Key 0 %

Oscillator: Wave Sin; Feedback 0 %, Repeat Off; Phase 0 %, Osc<Vel 0

LFO S&H L, Rate 6.05, Amount 25 %

Filter 12 24 Clean, Freq 18.5 kHz, Res 20 %

Pitch Env 0.0 %, Spread 79 %, Transpose 0 st

Time 0 %, Tone 70 %, Volume -6.7 dB

EQ Eight: Freq 4.00 kHz, Gain 7.81 dB, Q 0.71; Mode Stereo, Edit A, Adapt. Q On, Scale 100 %, Gain 0.00 dB

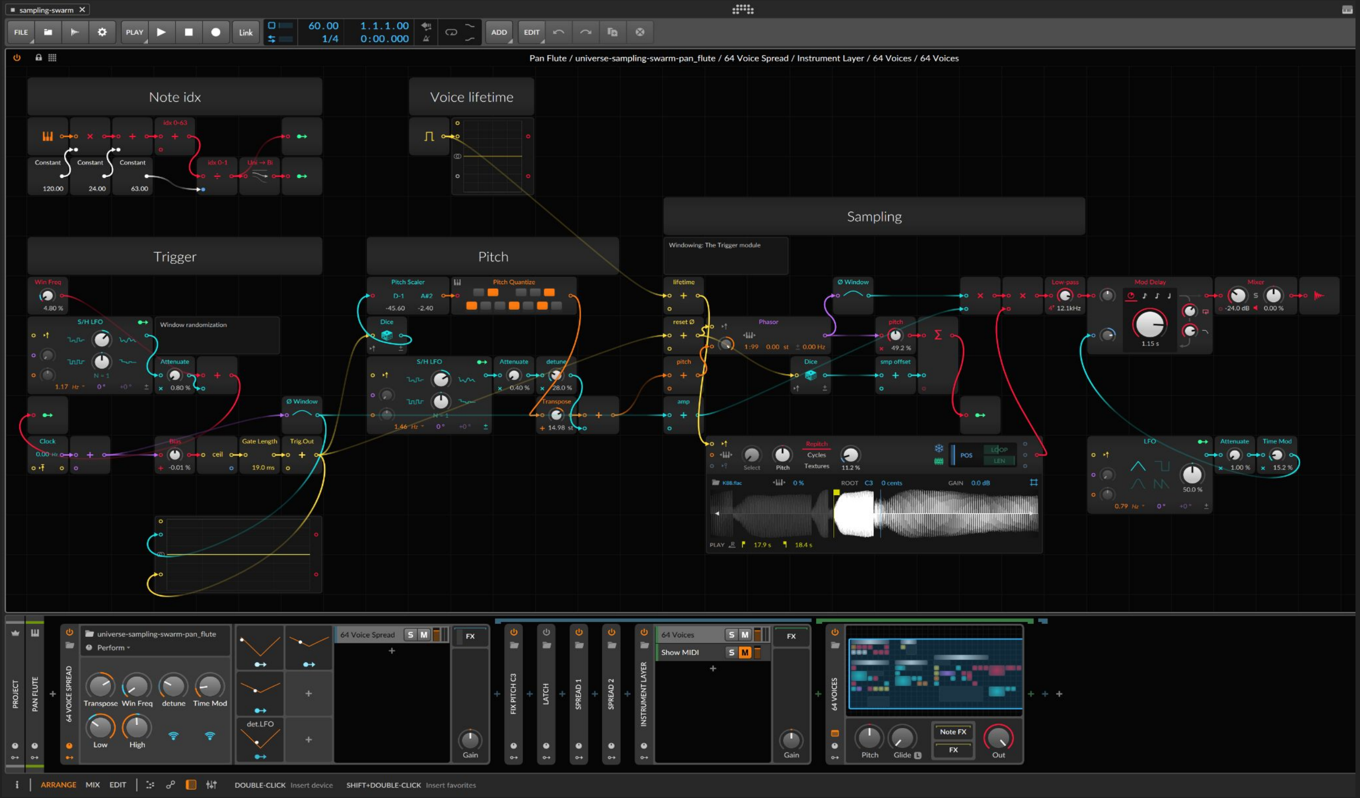
Auto Pan: Amount 100 %, Rate 1.00 Hz, Phase 180°, Shape 0.00 %, Normal

Delay: Left Time 7.00 ms, 0.0 %; Right Time 7.00 ms, 0.0 %; Feedback 0.0 %; Filter 1.00 kHz, Width 8.00; Modulation Rate 0.50 Hz, Filter 0.0 %, Time 0.0 %; Mode Repitch, Fade, Jump, Ping Pong; Dry/Wet 18 %

▸ frog

Real-time Synthesized Music

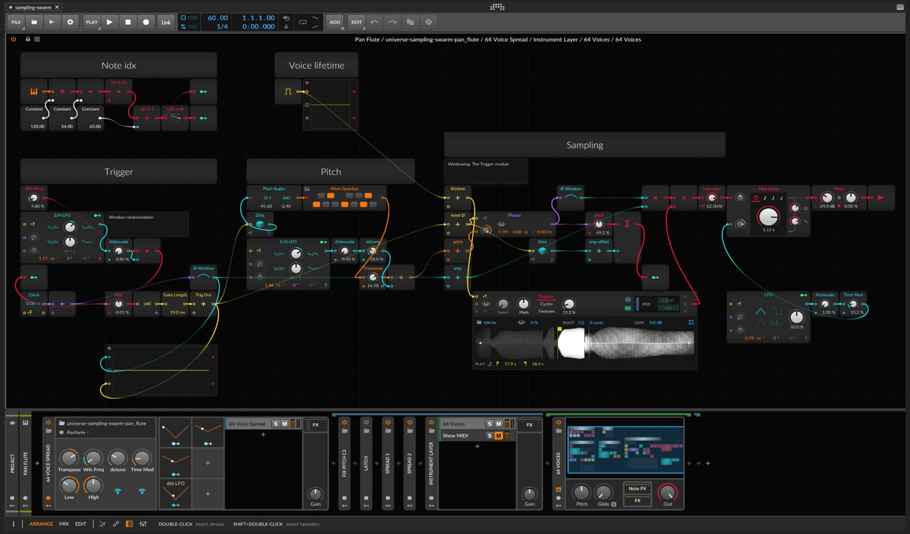
Bitwig Granular Swarm Experiment



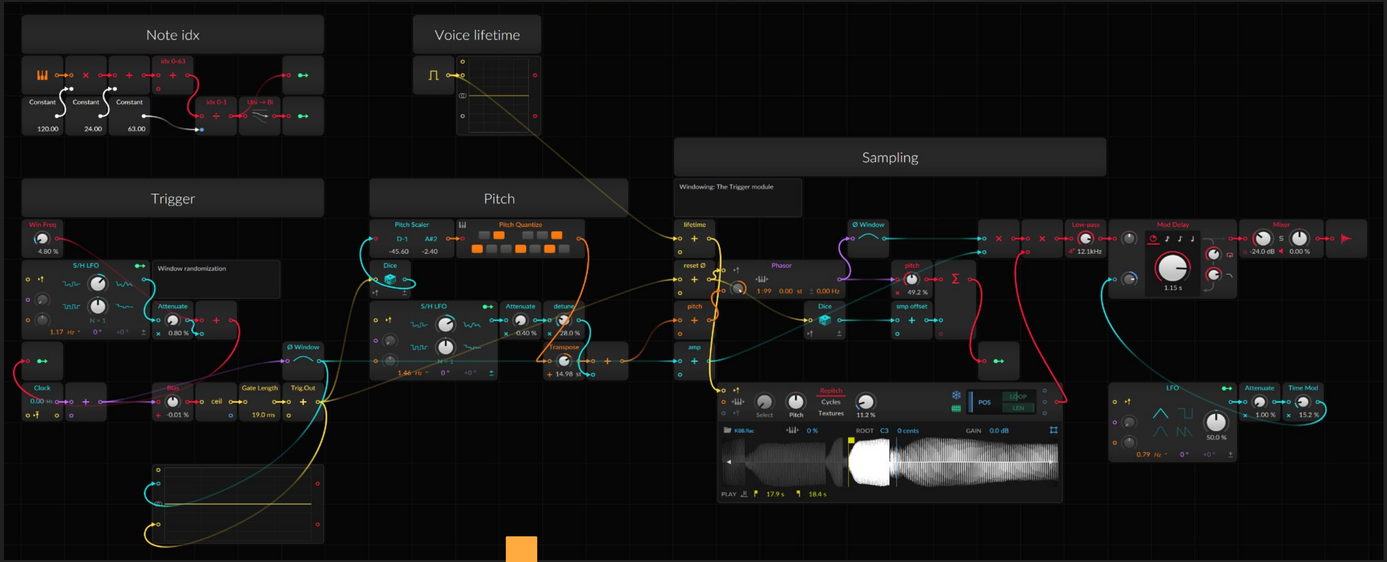
sampling-swarm
Pan Flute / universe-sampling-swarm-pan_flute / 64 Voice Spread / Instrument Layer / 64 Voices / 64 Voices
Note idx
Voice lifetime
Sampling
Trigger
Pitch
Windowing: The Trigger module
Window randomization
universe-sampling-swarm-pan_flute
Transpose
Win Freq
detune
Time Mod
Low
High
64 Voice Spread
64 Voices
Show MIDI
ARRANGE
MIX
EDIT
DOUBLE-CLICK Insert device
SHIFT+DOUBLE-CLICK Insert favorites

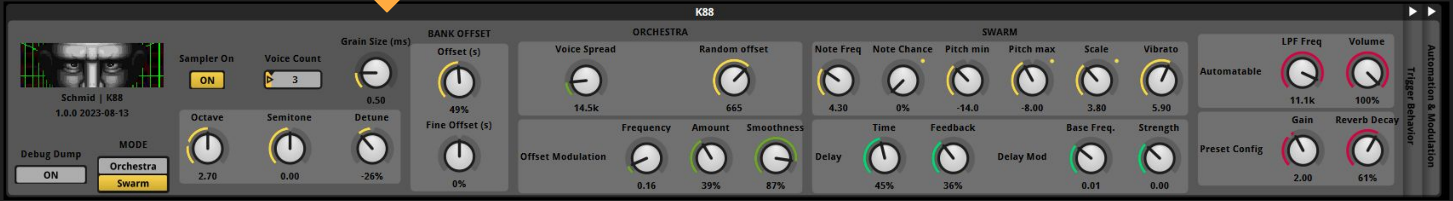
▸ bitwig-swarm

What if this was <u>in</u> the Game?



From Bitwig Prototype to FMOD Plugin





K88

Schmid | K88
1.0.0 2023-08-13

Sampler On: ON
Voice Count: 3
Grain Size (ms): 0.50
Octave: 2.70
Semitone: 0.00
Detune: -26%
Debug Dump: ON
MODE: Orchestra / Swarm

BANK OFFSET
Offset (s): 49%
Fine Offset (s): 0%

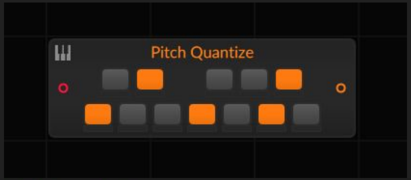
ORCHESTRA
Voice Spread: 14.5k
Random offset: 665
Offset Modulation — Frequency: 0.16, Amount: 39%, Smoothness: 87%

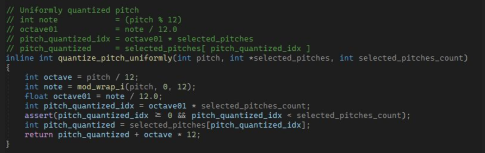
SWARM
Note Freq: 4.30
Note Chance: 0%
Pitch min: -14.0
Pitch max: -8.00
Scale: 3.80
Vibrato: 5.90
Delay — Time: 45%, Feedback: 36%
Delay Mod — Base Freq.: 0.01, Strength: 0.00

Automatable — LPF Freq: 11.1k, Volume: 100%
Preset Config — Gain: 2.00, Reverb Decay: 61%

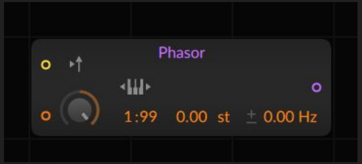
Trigger Behavior
Automation & Modulation

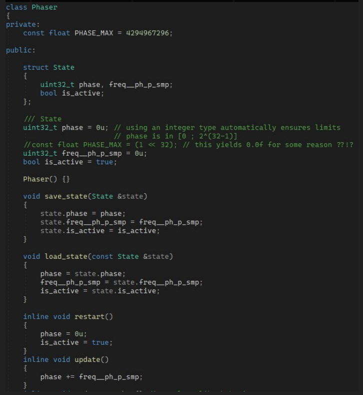
Translate Components to C++



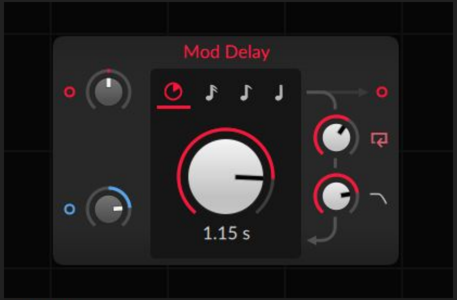


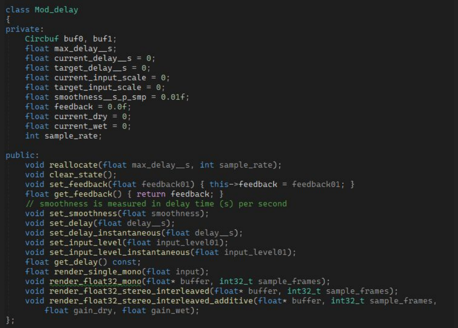
```cpp
// Uniformly quantized pitch
// int note            = (pitch % 12)
// octave01            = note / 12.0
// pitch_quantized_idx = octave01 * selected_pitches
// pitch_quantized     = selected_pitches[ pitch_quantized_idx ]
inline int quantize_pitch_uniformly(int pitch, int *selected_pitches, int selected_pitches_count)
{
    int octave = pitch / 12;
    int note = mod_wrap_i(pitch, 0, 12);
    float octave01 = note / 12.0;
    int pitch_quantized_idx = octave01 * selected_pitches_count;
    assert(pitch_quantized_idx >= 0 && pitch_quantized_idx < selected_pitches_count);
    int pitch_quantized = selected_pitches[pitch_quantized_idx];
    return pitch_quantized + octave * 12;
}
```



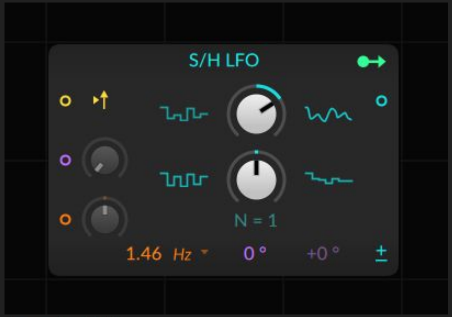


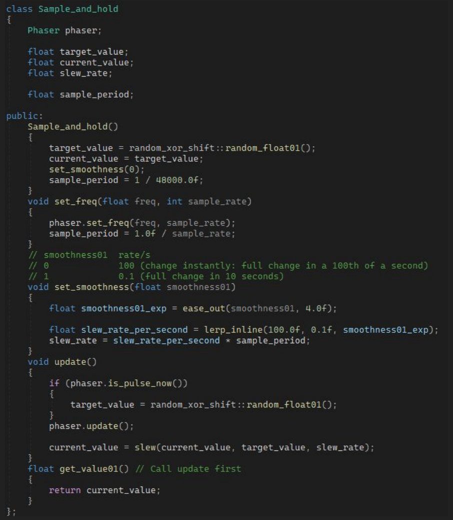
```cpp
class Phaser
{
private:
    const float PHASE_MAX = 4294967296;

public:

    struct State
    {
        uint32_t phase, freq__ph_p_smp;
        bool is_active;
    };

    /// State
    uint32_t phase = 0u; // using an integer type automatically ensures limits
                         // phase is in [0 ; 2^(32-1)]
    //const float PHASE_MAX = (1 << 32); // this yields 0.0f for some reason ??!?
    uint32_t freq__ph_p_smp = 0u;
    bool is_active = true;

    Phaser() {}

    void save_state(State &state)
    {
        state.phase = phase;
        state.freq__ph_p_smp = freq__ph_p_smp;
        state.is_active = is_active;
    }

    void load_state(const State &state)
    {
        phase = state.phase;
        freq__ph_p_smp = state.freq__ph_p_smp;
        is_active = state.is_active;
    }

    inline void restart()
    {
        phase = 0u;
        is_active = true;
    }
    inline void update()
    {
        phase += freq__ph_p_smp;
    }
```





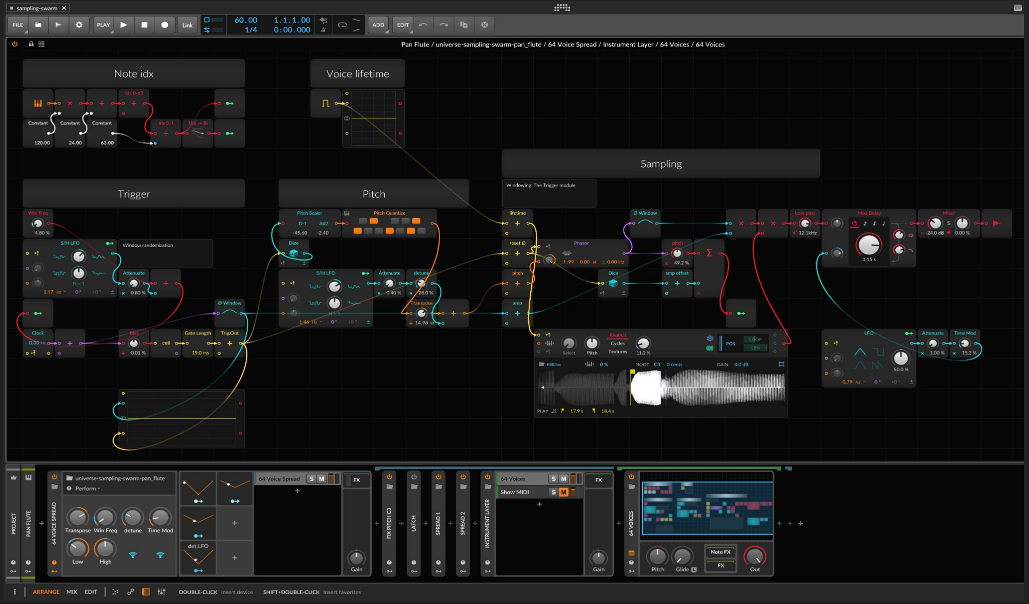
```cpp
class Mod_delay
{
private:
    Circbuf buf0, buf1;
    float max_delay__s;
    float current_delay__s = 0;
    float target_delay__s = 0;
    float current_input_scale = 0;
    float target_input_scale = 0;
    float smoothness__s_p_smp = 0.01f;
    float feedback = 0.0f;
    float current_dry = 0;
    float current_wet = 0;
    int sample_rate;

public:
    void reallocate(float max_delay__s, int sample_rate);
    void clear_state();
    void set_feedback(float feedback01) { this->feedback = feedback01; }
    float get_feedback() { return feedback; }
    // smoothness is measured in delay time (s) per second
    void set_smoothness(float smoothness);
    void set_delay(float delay__s);
    void set_delay_instantaneous(float delay__s);
    void set_input_level(float input_level01);
    void set_input_level_instantaneous(float input_level01);
    float get_delay() const;
    float render_single_mono(float input);
    void render_float32_mono(float* buffer, int32_t sample_frames);
    void render_float32_stereo_interleaved(float* buffer, int32_t sample_frames);
    void render_float32_stereo_interleaved_additive(float* buffer, int32_t sample_frames,
        float gain_dry, float gain_wet);
};
```





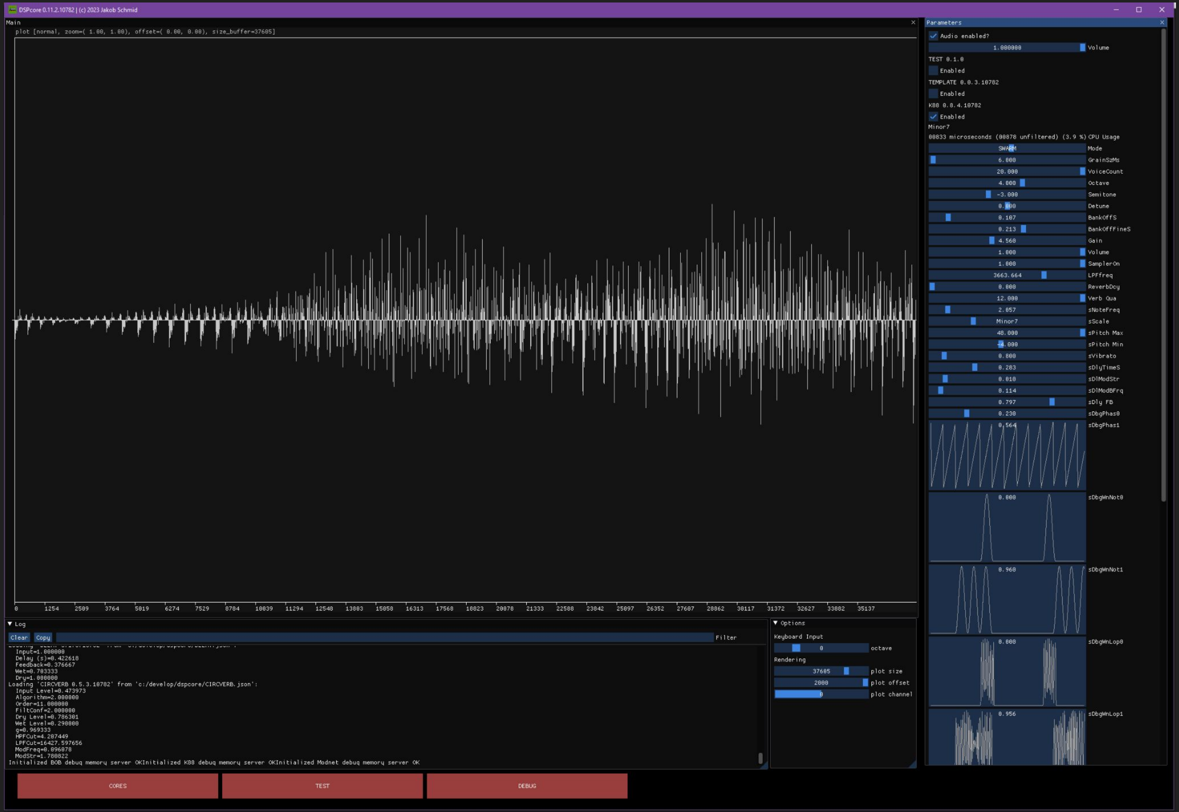
```cpp
class Sample_and_hold
{
    Phaser phaser;

    float target_value;
    float current_value;
    float slew_rate;

    float sample_period;

public:
    Sample_and_hold()
    {
        target_value = random_xor_shift::random_float01();
        current_value = target_value;
        set_smoothness(0);
        sample_period = 1 / 48000.0f;
    }
    void set_freq(float freq, int sample_rate)
    {
        phaser.set_freq(freq, sample_rate);
        sample_period = 1.0f / sample_rate;
    }
    // smoothness01  rate/s
    // 0             100 (change instantly: full change in a 100th of a second)
    // 1             0.1 (full change in 10 seconds)
    void set_smoothness(float smoothness01)
    {
        float smoothness01_exp = ease_out(smoothness01, 4.0f);

        float slew_rate_per_second = lerp_inline(100.0f, 0.1f, smoothness01_exp);
        slew_rate = slew_rate_per_second * sample_period;
    }
    void update()
    {
        if (phaser.is_pulse_now())
        {
            target_value = random_xor_shift::random_float01();
        }
        phaser.update();

        current_value = slew(current_value, target_value, slew_rate);
    }
    float get_value01() // Call update first
    {
        return current_value;
    }
};
```

Translate Patch to C++


Pan Flute / universe-sampling-swarm-pan_flute / 64 Voice Spread / Instrument Layer / 64 Voices / 64 Voices
Note idx
Voice lifetime
Sampling
Trigger
Pitch

```cpp
void Swarm::render_float32_stereo_interleaved(float* buffer, int32_t sample_frames, uint64_t clock)
{
    using namespace random_xor_shift;

    // Ensure reasonable default values
    window_size_ms = clamp(window_size_ms, min_window_ms, max_window_ms);
    bank_offset_s = clamp(bank_offset_s, min_bank_offset_s, max_bank_offset_s);

    float start_time_s = bank_offset_s - window_size_ms * 0.5f * 0.001f;
    float end_time_s   = bank_offset_s + window_size_ms * 0.5f * 0.001f;
    float start_smp = start_time_s * 44100;
    float end_smp   = end_time_s * 44100;
    int max_offset = min(44100, static_cast<int>(end_smp - start_smp));

    int idx = 0;
    for (int i = 0, count = sample_frames; i < count; ++i)
    {
        buffer[idx++] = 0;
        buffer[idx++] = 0;
    }

    float amp = sqrtf(1.0f / voice_count);

    for (int vidx = 0; vidx < voice_count; ++vidx)
    {
        float v01 = idx_to_01(vidx, voice_count);
        float pan_factor_l = pan01_to_factor_l(v01);
        float pan_factor_r = pan01_to_factor_r(v01);
        Voice_state& state = voices[vidx];

        float window_big = 0.0f;
        float window_loop = 0.0f;

        int idx = 0;
        for (int i = 0, count = sample_frames; i < count; ++i)
        {
            bool retrigger = state.note_phasor.is_pulse_now();
            if (retrigger)
            {
                // Trigger
                if (random_float01() < note_chance)
                {
                    int pitch = lerp_inline(pitch_min, pitch_max, random_float01());
                    int pitch_scale = quantize_pitch_uniformly(pitch, scale_bitfield);

                    int current_offset = random_int(0, max_offset);
                    state.start_smp = current_offset + start_smp;
                    state.end_smp = current_offset + end_smp;

                    // FIXME: Compute relative pitch, assuming waveform is C3
                    float tune = octave * 12 + semitone + detune;
                    state.current_freq = floatmidi2freq(pitch_scale + tune);

                    state.sample_phasor.restart();
                }
                // Stop
                else
                {
                    state.current_freq = -1.0f; // voice off
                }
            }

            float out0 = 0.0f;
            float out1 = 0.0f;

            bool is_voice_on = (state.current_freq > -1.0f);
            if (is_voice_on)
            {
                float freq = state.current_freq + state.pmod.get_value01() * vibrato;
                state.sample_phasor.set_freq(freq / window_size_ms, sample_rate);

                window_big = hanning_window->lookup_uint32(state.note_phasor.phase);
                window_loop = hanning_window->lookup_uint32(state.sample_phasor.phase);

                float phase01 = state.sample_phasor.saw_up01();
                float sample_idx = lerp_inline(state.start_smp, state.end_smp, phase01);

                // interpolated sample lookup
                float amp_win = window_big * window_loop;
                float out = get_interpolated_sample_decrypt(waveform, waveform_length, sample_idx) * amp_win;

                // render to buffer with panning
                out0 = out * amp * pan_factor_l;
                out1 = out * amp * pan_factor_r;
            }

            out0 += state.delay0.render_single_mono(out0);
            out1 += state.delay1.render_single_mono(out1);

            buffer[idx++] += out0;
            buffer[idx++] += out1;

            float mod = sine_table->lookup01_uint32(state.delay_mod_phasor.phase) * delay_mod_str;
            state.delay0.set_delay(mod_delay_time + mod);
            state.delay1.set_delay(mod_delay_time + mod);
            state.delay_mod_phasor.update();
            state.sample_phasor.update();
            state.note_phasor.update();
            state.pmod.update();
        }
    }
}
```

Test in custom GUI



```
plot [normal, zoom=( 1.00, 1.00), offset=( 0.00, 0.00), size_buffer=37605]
```

```
  Input=1.000000
  Delay (s)=0.422618
  Feedback=0.376667
  Wet=0.703333
  Dry=1.000000
Loading 'CIRCVERB 0.5.3.10782' from 'c:/develop/dspcore/CIRCVERB.json':
  Input Level=0.473973
  Algorithm=2.000000
  Order=11.000000
  FiltConf=2.000000
  Dry Level=0.786301
  Wet Level=0.290000
  g=0.969333
  HPFCut=4.207449
  LPFCut=16427.597656
  ModFreq=0.096078
  ModStr=1.780822
Initialized BOB debug memory server OKInitialized K88 debug memory server OKInitialized Modnet debug memory server OK
```

Options

Keyboard Input

0	octave

Rendering

37605	plot size
2000	plot offset
0	plot channel

Parameters

✓ Audio enabled?

1.000000	Volume

TEST 0.1.0

Enabled

TEMPLATE 0.0.3.10782

Enabled

K88 0.8.4.10782

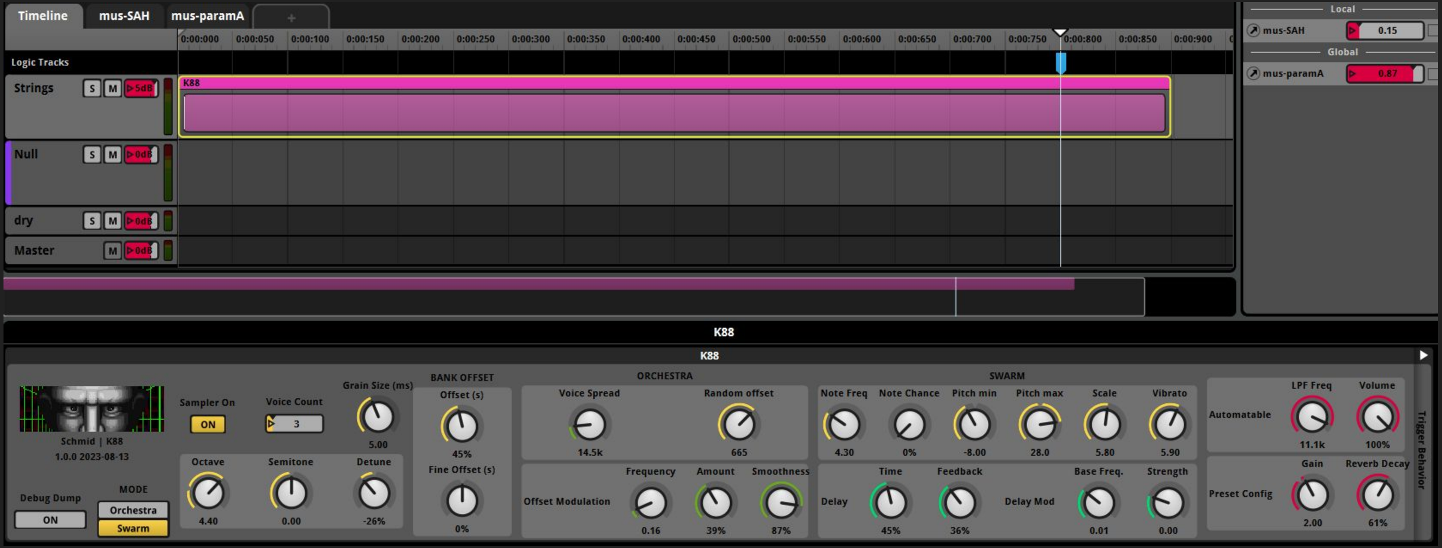
✓ Enabled

Minor7

00833 microseconds (00878 unfiltered) (3.9 %) CPU Usage

SWARM	Mode
6.000	GrainSzMs
20.000	VoiceCount
4.000	Octave
-3.000	Semitone
0.000	Detune
0.107	BankOffS
0.213	BankOffFineS
4.560	Gain
1.000	Volume
1.000	SamplerOn
3663.664	LPFfreq
0.000	ReverbDcy
12.000	Verb Qua
2.057	sNoteFreq
Minor7	sScale
48.000	sPitch Max
-4.000	sPitch Min
0.800	sVibrato
0.283	sDlyTimeS
0.010	sDlModStr
0.114	sDlModBFrq
0.797	sDly FB
0.230	sDbgPhas0
0.564	sDbgPhas1
0.000	sDbgWnNot0
0.960	sDbgWnNot1
0.000	sDbgWnLop0
0.956	sDbgWnLop1

Wrap as FMOD Plug-in Instrument



mus-SAH 0.15
mus-paramA 0.87

Strings S M ▷5dB
Null S M ▷0dB
dry S M ▷0dB
Master M ▷0dB

K88

Schmid | K88
1.0.0 2023-08-13

Sampler On: ON
Voice Count: 3
Grain Size (ms): 5.00
Octave: 4.40
Semitone: 0.00
Detune: -26%
Debug Dump: ON
MODE: Orchestra [✓ Swarm]

BANK OFFSET
Offset (s): 45%
Fine Offset (s): 0%

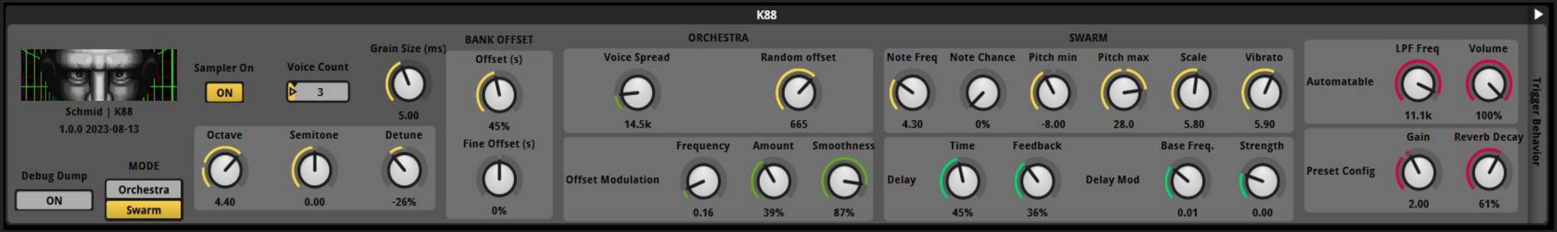
ORCHESTRA
Voice Spread: 14.5k
Random offset: 665
Offset Modulation — Frequency: 0.16, Amount: 39%, Smoothness: 87%

SWARM
Note Freq: 4.30
Note Chance: 0%
Pitch min: -8.00
Pitch max: 28.0
Scale: 5.80
Vibrato: 5.90
Delay — Time: 45%, Feedback: 36%
Delay Mod — Base Freq.: 0.01, Strength: 0.00

Automatable — LPF Freq: 11.1k, Volume: 100%
Preset Config — Gain: 2.00, Reverb Decay: 61%

▸ fmod-k88

COCOON Plugin Instruments


K88

Schmid | K88
1.0.0 2023-08-13

Sampler On: ON
Voice Count: 3
Grain Size (ms): 5.00
Octave: 4.40
Semitone: 0.00
Detune: -26%
Debug Dump: ON
MODE: Orchestra / Swarm

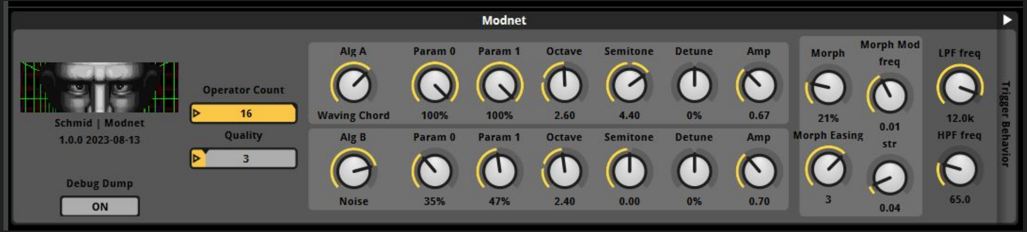
BANK OFFSET
Offset (s): 45%
Fine Offset (s): 0%

ORCHESTRA
Voice Spread: 14.5k
Random offset: 665
Offset Modulation — Frequency: 0.16, Amount: 39%, Smoothness: 87%

SWARM
Note Freq: 4.30
Note Chance: 0%
Pitch min: -8.00
Pitch max: 28.0
Scale: 5.80
Vibrato: 5.90
Delay — Time: 45%, Feedback: 36%
Delay Mod — Base Freq.: 0.01, Strength: 0.00

Automatable — LPF Freq: 11.1k, Volume: 100%
Preset Config — Gain: 2.00, Reverb Decay: 61%

Trigger Behavior


Modnet

Schmid | Modnet
1.0.0 2023-08-13

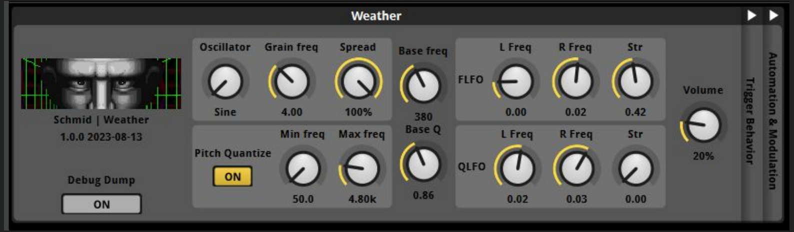
Operator Count: 16
Quality: 3
Debug Dump: ON

Alg A	Param 0	Param 1	Octave	Semitone	Detune	Amp
Waving Chord	100%	100%	2.60	4.40	0%	0.67

Alg B	Param 0	Param 1	Octave	Semitone	Detune	Amp
Noise	35%	47%	2.40	0.00	0%	0.70

Morph: 21%
Morph Mod freq: 0.01
Morph Easing: 3
str: 0.04
LPF freq: 12.0k
HPF freq: 65.0

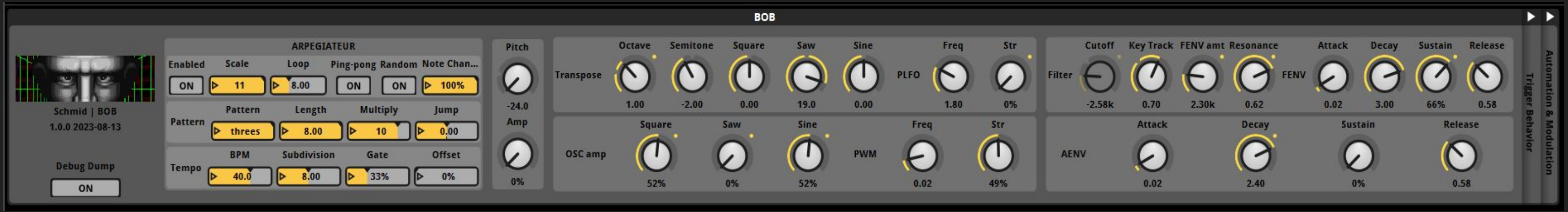
Trigger Behavior


Weather

Schmid | Weather
1.0.0 2023-08-13

Debug Dump: ON
Oscillator: Sine
Grain freq: 4.00
Spread: 100%
Base freq: 380
Base Q: 0.86
Pitch Quantize: ON
Min freq: 50.0
Max freq: 4.80k
FLFO — L Freq: 0.00, R Freq: 0.02, Str: 0.42
QLFO — L Freq: 0.02, R Freq: 0.03, Str: 0.00
Volume: 20%

Trigger Behavior
Automation & Modulation


BOB

Schmid | BOB
1.0.0 2023-08-13

Debug Dump: ON

ARPEGIATEUR
Enabled: ON
Scale: 11
Loop: 8.00
Ping-pong: ON
Random: ON
Note Chan...: 100%
Pattern — Pattern: threes, Length: 8.00, Multiply: 10, Jump: 0.00
Tempo — BPM: 40.0, Subdivision: 8.00, Gate: 33%, Offset: 0%

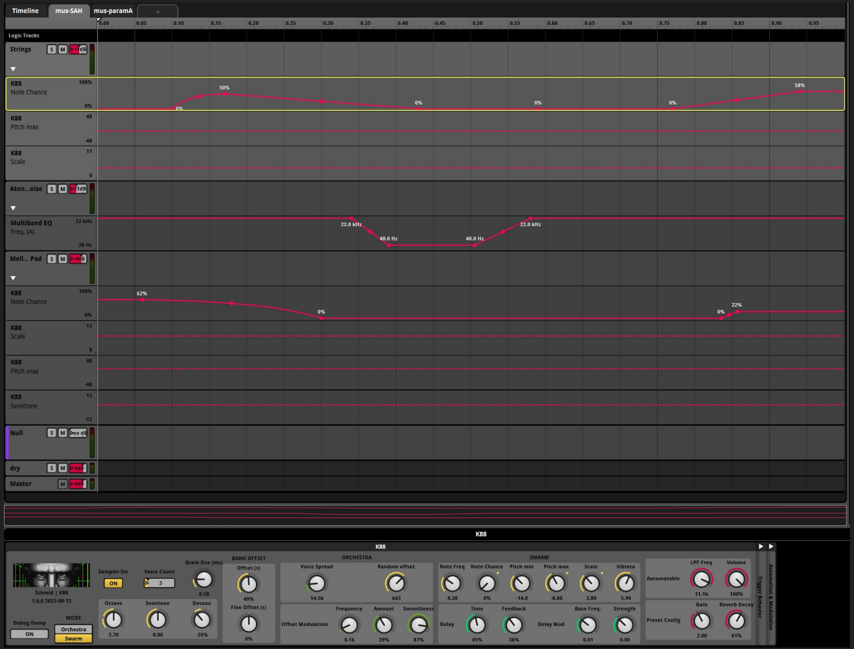
Pitch: -24.0
Amp: 0%

Transpose — Octave: 1.00, Semitone: -2.00, Square: 0.00, Saw: 19.0, Sine: 0.00
PLFO — Freq: 1.80, Str: 0%
OSC amp — Square: 52%, Saw: 0%, Sine: 52%
PWM — Freq: 0.02, Str: 49%

Filter — Cutoff: -2.58k, Key Track: 0.70, FENV amt: 2.30k, Resonance: 0.62
FENV — Attack: 0.02, Decay: 3.00, Sustain: 66%, Release: 0.58
AENV — Attack: 0.02, Decay: 2.40, Sustain: 0%, Release: 0.58

Trigger Behavior
Automation & Modulation

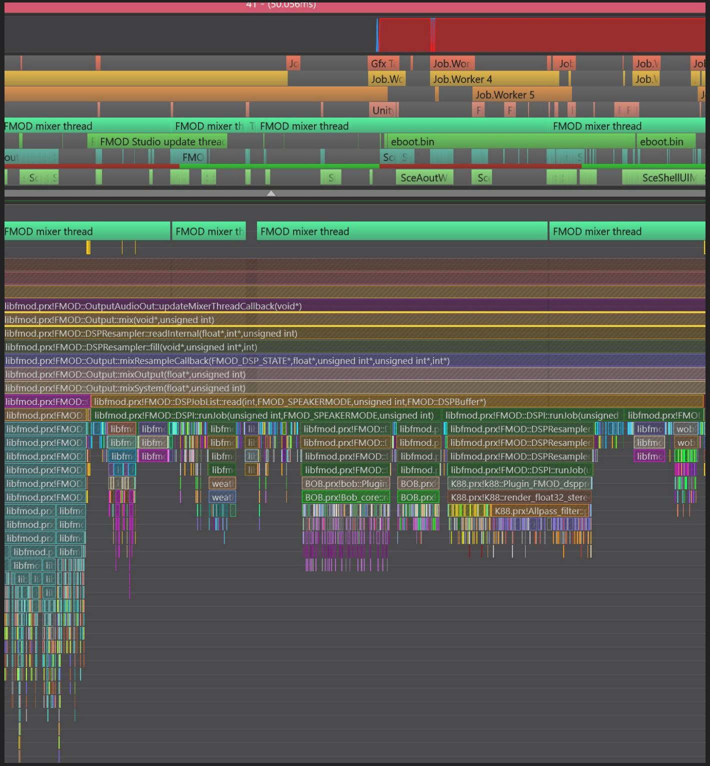
Real-time Synthesized Music in FMOD



▸ fmod-gig

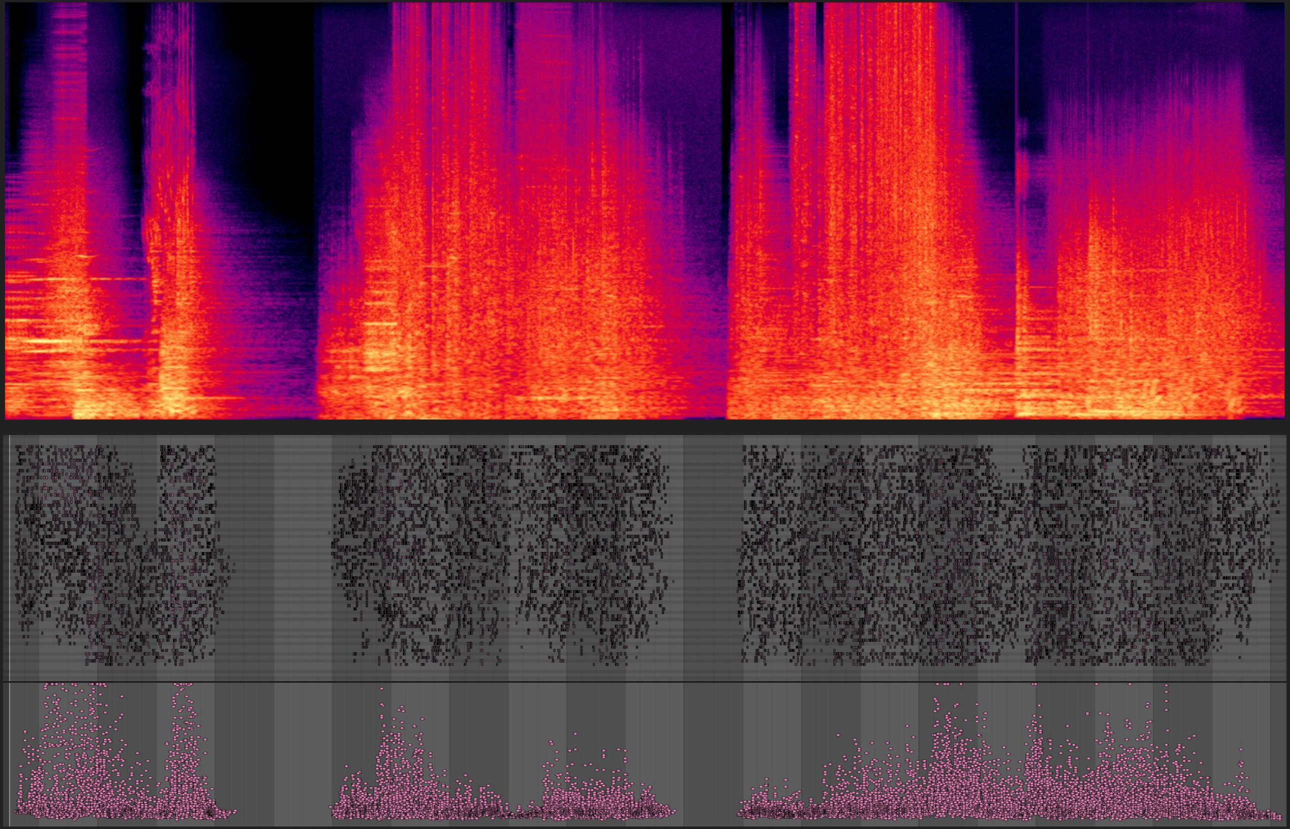
Real-time Synthesized Music on All Platforms

- Windows
- Xbox Series S|X, Xbox One
- PlayStation 5, PlayStation 4
- Nintendo Switch



FMOD mixer thread
FMOD Studio update thread
Job.Worker 4
Job.Worker 5
eboot.bin
SceAoutW
SceShellUI
libfmod.prx!FMOD::OutputAudioOut::updateMixerThreadCallback(void*)
libfmod.prx!FMOD::Output::mix(void*,unsigned int)
libfmod.prx!FMOD::DSPResampler::readInternal(float*,int*,unsigned int)
libfmod.prx!FMOD::DSPResampler::fill(void*,unsigned int*,int)
libfmod.prx!FMOD::Output::mixResampleCallback(FMOD_DSP_STATE*,float*,unsigned int*,unsigned int*,int*)
libfmod.prx!FMOD::Output::mixOutput(float*,unsigned int)
libfmod.prx!FMOD::Output::mixSystem(float*,unsigned int)
libfmod.prx!FMOD::DSPJobList::read(int,FMOD_SPEAKERMODE,unsigned int,FMOD::DSPBuffer*)
libfmod.prx!FMOD::DSPI::runJob(unsigned int,FMOD_SPEAKERMODE,unsigned int)
BOB.prx!bob::Plugin
K88.prx!k88::Plugin_FMOD_dspprocess
K88.prx!K88::render_float32_stereo
K88.prx!Allpass_filter

MIDI Vocoder: Dyson Gate

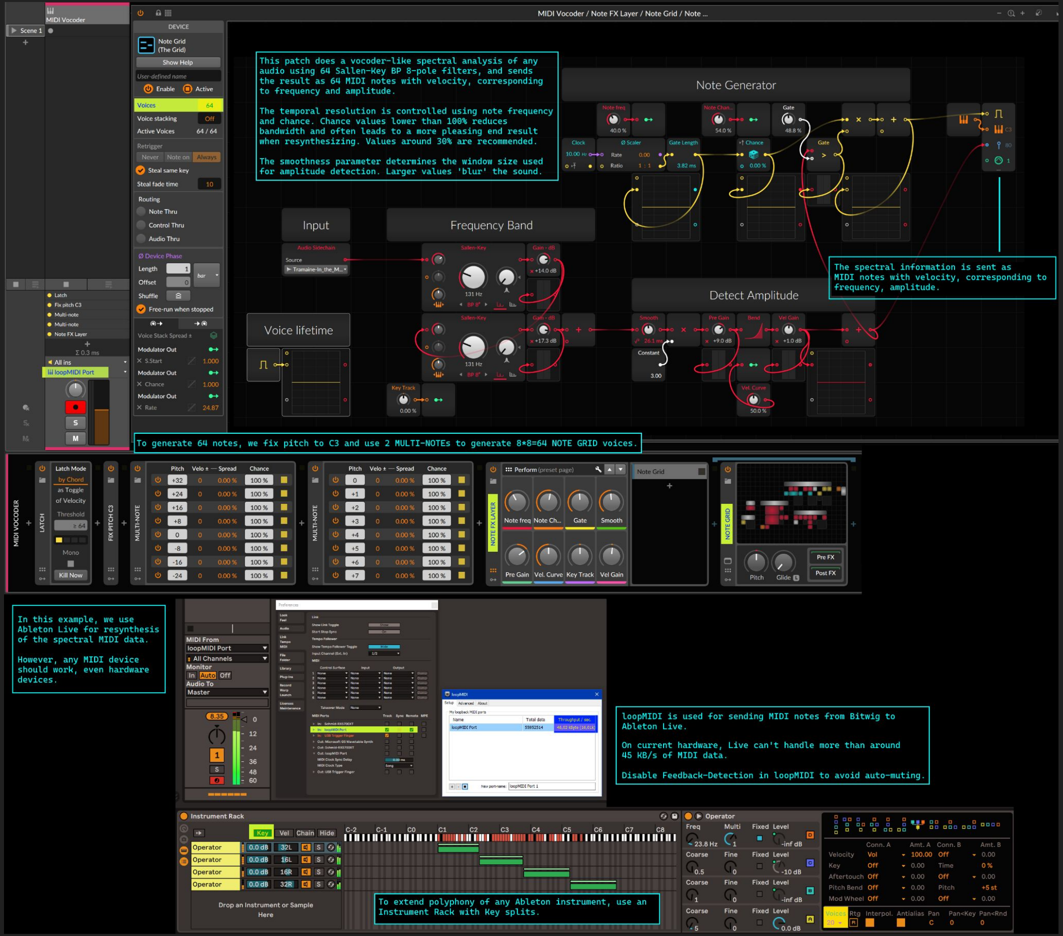


▸ `midi_vocoder-bitwig, midi_vocoder-ableton, cocoon-gate`

MIDI Vocoder

Home-made vocoder

- Bitwig audio analysis
- MIDI sent via loopMIDI
- Record MIDI in Ableton Live



This patch does a vocoder-like spectral analysis of any audio using 64 Sallen-Key BP 8-pole filters, and sends the result as 64 MIDI notes with velocity, corresponding to frequency and amplitude.

The temporal resolution is controlled using note frequency and chance. Chance values lower than 100% reduces bandwidth and often leads to a more pleasing end result when resynthesizing. Values around 30% are recommended.

The smoothness parameter determines the window size used for amplitude detection. Larger values 'blur' the sound.

The spectral information is sent as MIDI notes with velocity, corresponding to frequency, amplitude.

To generate 64 notes, we fix pitch to C3 and use 2 MULTI-NOTEs to generate 8*8=64 NOTE GRID voices.

In this example, we use Ableton Live for resynthesis of the spectral MIDI data.

However, any MIDI device should work, even hardware devices.

loopMIDI is used for sending MIDI notes from Bitwig to Ableton Live.

On current hardware, Live can't handle more than around 45 KB/s of MIDI data.

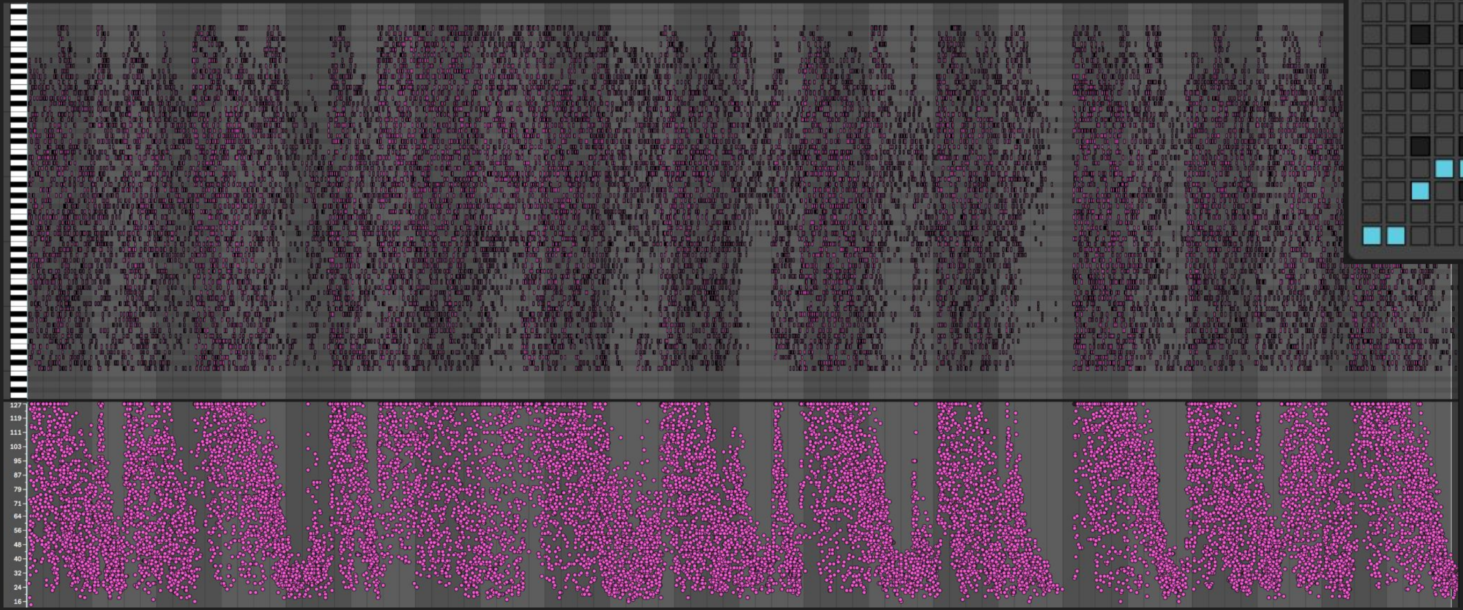
Disable Feedback-Detection in loopMIDI to avoid auto-muting.

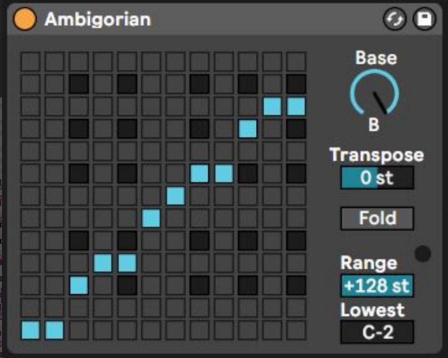
To extend polyphony of any Ableton instrument, use an Instrument Rack with Key splits.

Puzzle Feedback Music

▸ cocoon-puzzfeed

MIDI Vocoder: Puzzle Feedback




Ambigorian

Base: B

Transpose: 0 st

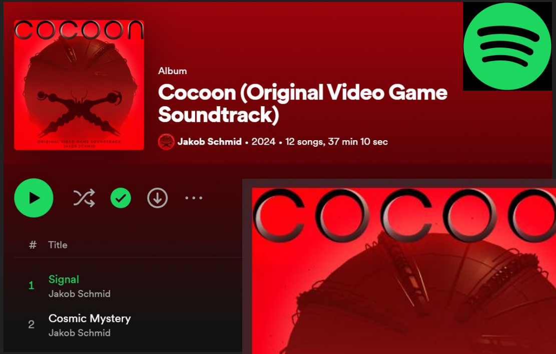
Fold

Range: +128 st

Lowest: C-2

▸ midi_vocoder-puzzfeed

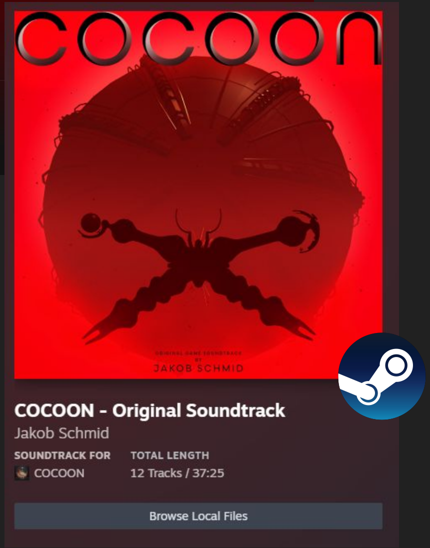
Questions?


Album

Cocoon (Original Video Game Soundtrack)

Jakob Schmid • 2024 • 12 songs, 37 min 10 sec

#	Title
1	Signal Jakob Schmid
2	Cosmic Mystery Jakob Schmid


COCOON - Original Soundtrack

Jakob Schmid

SOUNDTRACK FOR	TOTAL LENGTH
COCOON	12 Tracks / 37:25

Browse Local Files

AVAILABLE NOW ON

STEAM XBOX NINTENDO SWITCH PS5 | PS4