Realtime Synthesis and Dynamic Music in Games

Sonic College 2023

Jakob Schmid

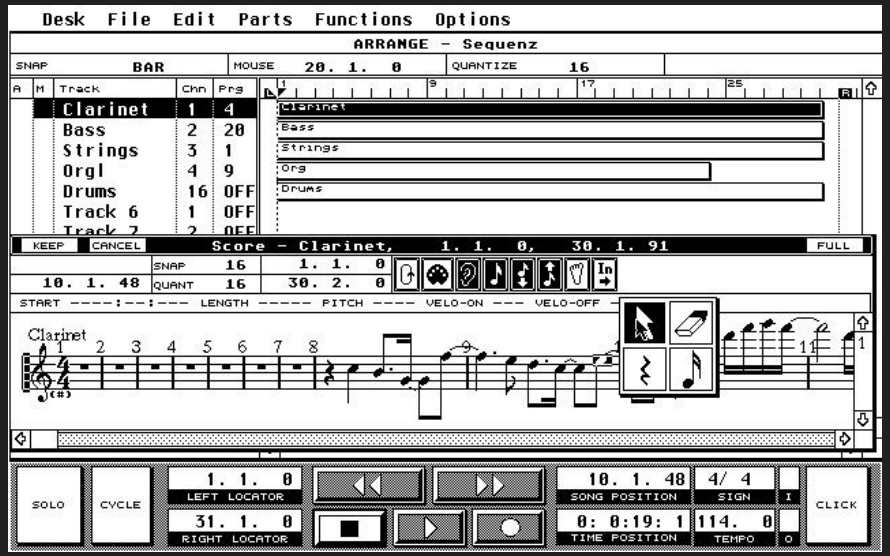
Overview

- Purpose and History of Realtime Synthesis in Games
- Dynamic Music in Games
- Audio Plugins
- Plugin Platforms
- Example Unity Plugins

Purpose of Realtime Synthesis in Games

MIDI-like Sequencing

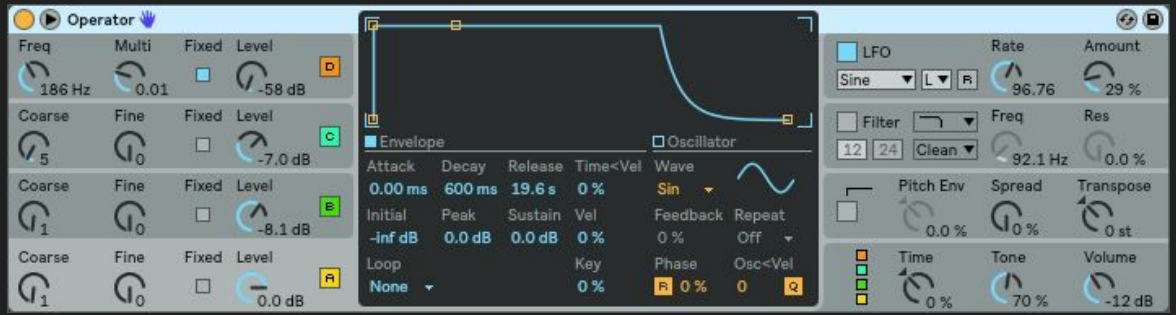
- Sequencing of samples or real-time synthesis
- Key changes
- Removing notes
- Procedural / generative music



Cubase (1989)

Real-time Synthesis

- Parameter changes controlled from game
- Subtle changes in timbre accompany game events
- Variations in timbre retain player interest even though sequence repeats



Ableton Live 10: Operator

History of Realtime Synthesis in Games

Realtime Synthesis was the Norm

- 1970s to mid 1980s: hardware-based realtime synthesis
- Hardware synthesizer-based hardware platforms
  - Arcade machines (1970s and forward)
  - Atari 2600 (1979)
  - ZX Spectrum (1982)
  - Commodore 64 (1982)

Marble Madness

ZX Spectrum

Commodore 64

Atari 2600

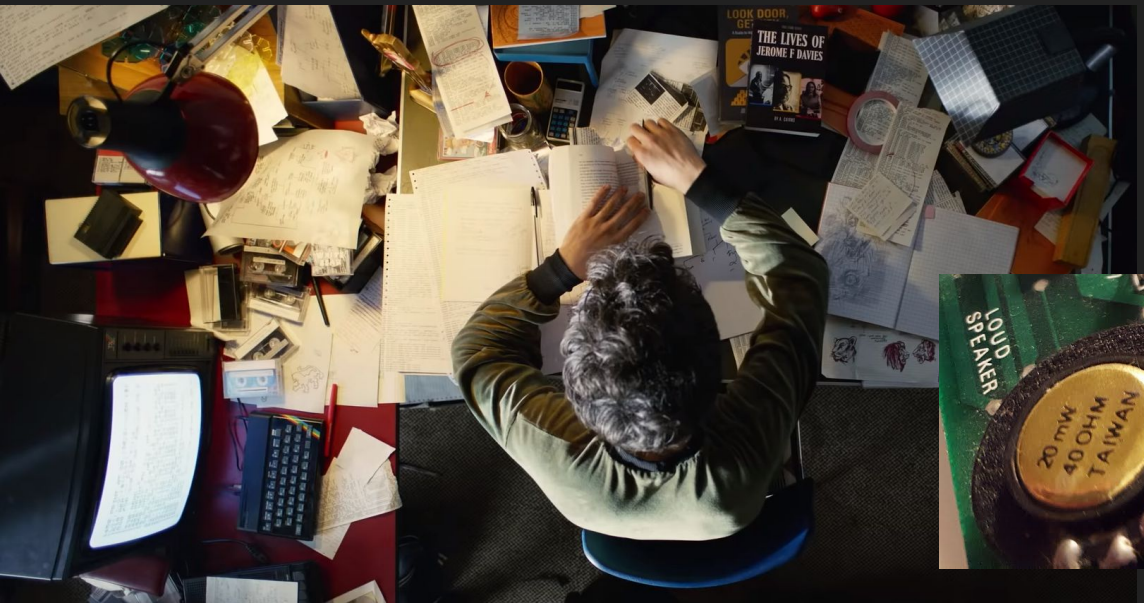
Real-time Synthesis was the Norm

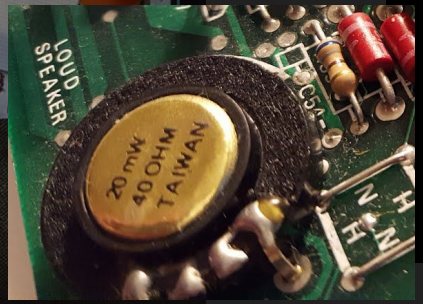
- Sound chips with fixed number of DCOs controlled from CPU
- Possible to play samples using clever tricks
- Sample playback hardware become the norm in 1985 and forward

Amiga 1000 (1985),
sample-based audio hardware

ZX Spectrum Speaker

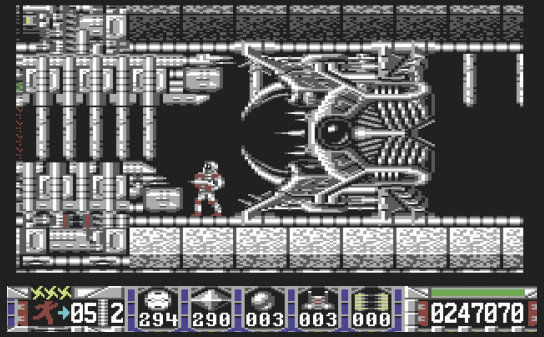
- 1 tone generator
- 1-bit volume, on or off





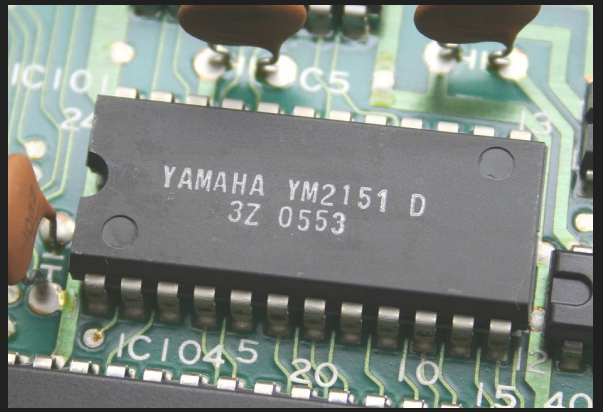
Commodore 64 SID Chip

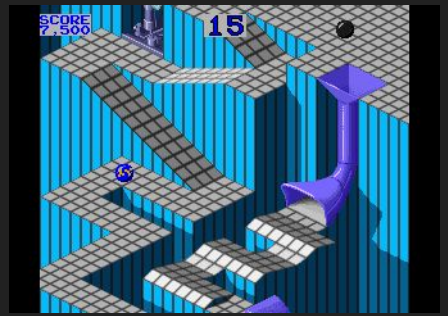
- 3 DCOs
- Waveforms: pulse, triangle, saw, noise
- Ring modulation, oscillator sync
- Multimode filter: low-, high-, bandpass (6dB/12dB rolloff)
- 3 Envelope generators



Yamaha YM2151

- FM synthesis, 4 operators
- 8 channel polyphony
- Used in many arcade games by Atari, SEGA, and Konami
- See also https://vgmrips.net/packs/chip/ym2151





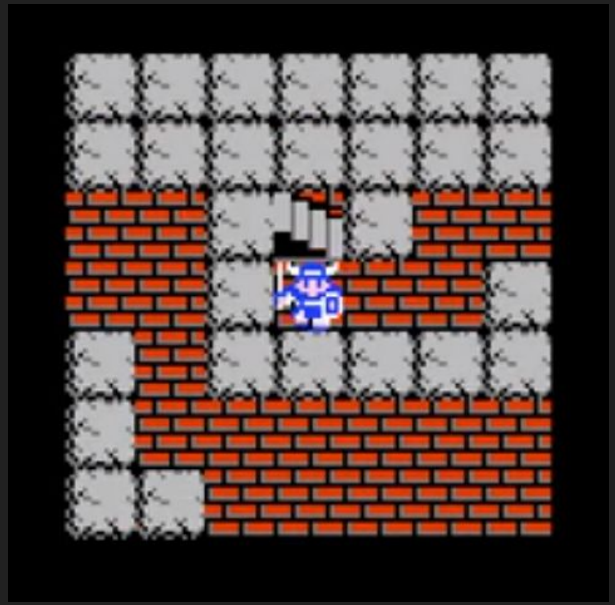
Summary

- MIDI-like sequencing allows generating or modifying notes
- Realtime synthesis allows for game-controlled parameter changes
- Hardware based realtime synthesis was the norm until mid 1980s
- Early audio hardware ranged from 1-bit tone generators to subtractive synthesis and FM synthesis.

Dynamic Music in Games

Dragon Warrior

- NES 1986
- Dungeon music changes key with dungeon level
- Helps player finding their way around?



Otocky

- NES 1987
- Music generated by gameplay elements

iMUSE

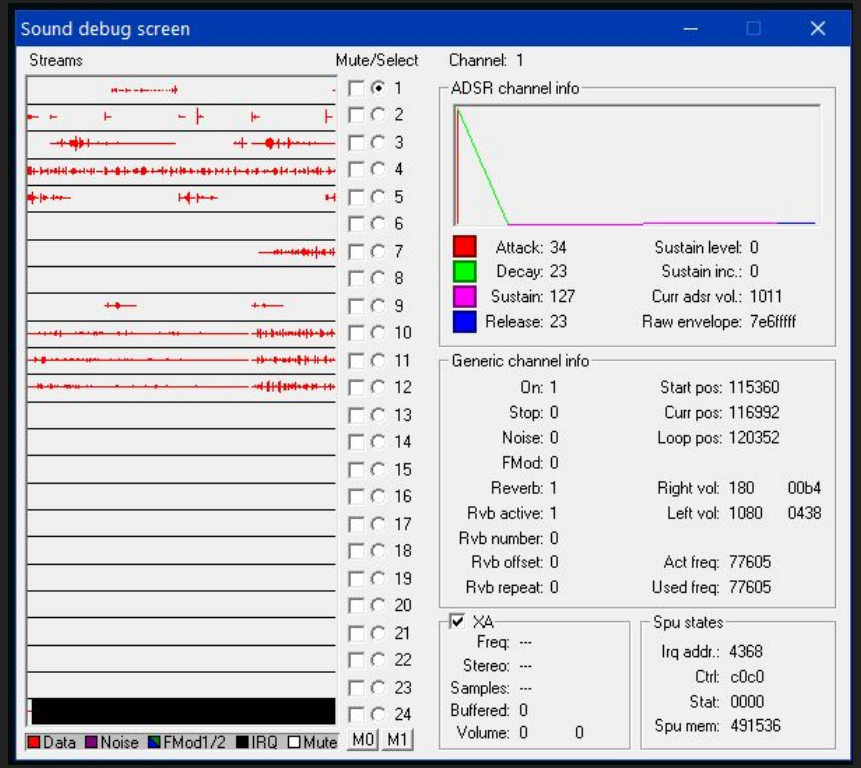
- Interactive Music Streaming Engine
- LucasArts
- First used in 'Monkey Island 2: LeChuck's Revenge', DOS 1991
- Adaptive MIDI music with seamless transitions and feedback to gameplay

Mid-late 1990s

- Sample-based audio hardware
- Cheaper memory makes sampled sound more useful

PlayStation Audio Hardware

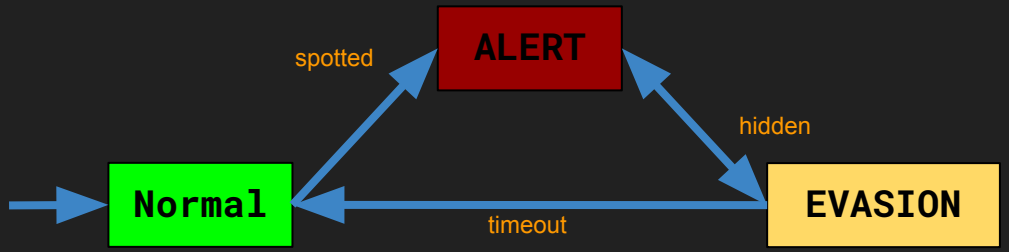
- 24 channels, 16-bit, 44.1 KHz
- Each channel:
  - looping
  - pitch
  - amplitude envelope
  - panning
  - effect send on/off
- 1 stereo streaming CD track
- 1 configurable delay / reverb effect
- 512 KB audio memory



Metal Gear Solid

- PlayStation 1998
- Stealth action game
- Directed by Hideo Kojima

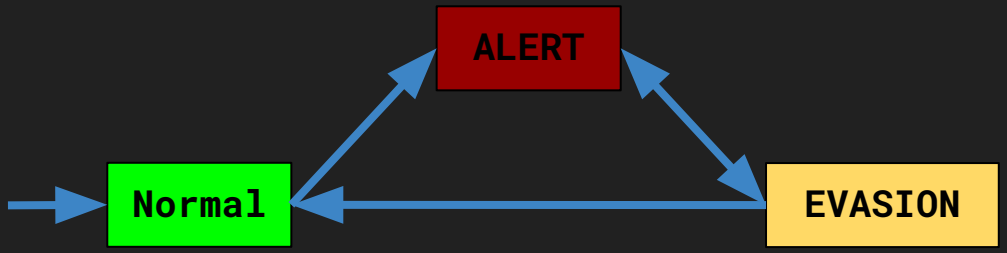
Metal Gear Solid: Game States

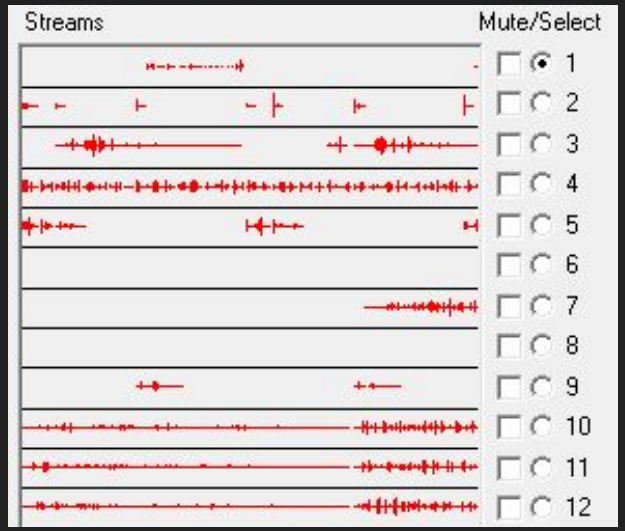


- Normal >> ALERT: immediately when spotted
- ALERT >> EVASION: automatic when hidden for a while
- EVASION >> Normal: automatic when hidden for a while

Metal Gear Solid: Music System

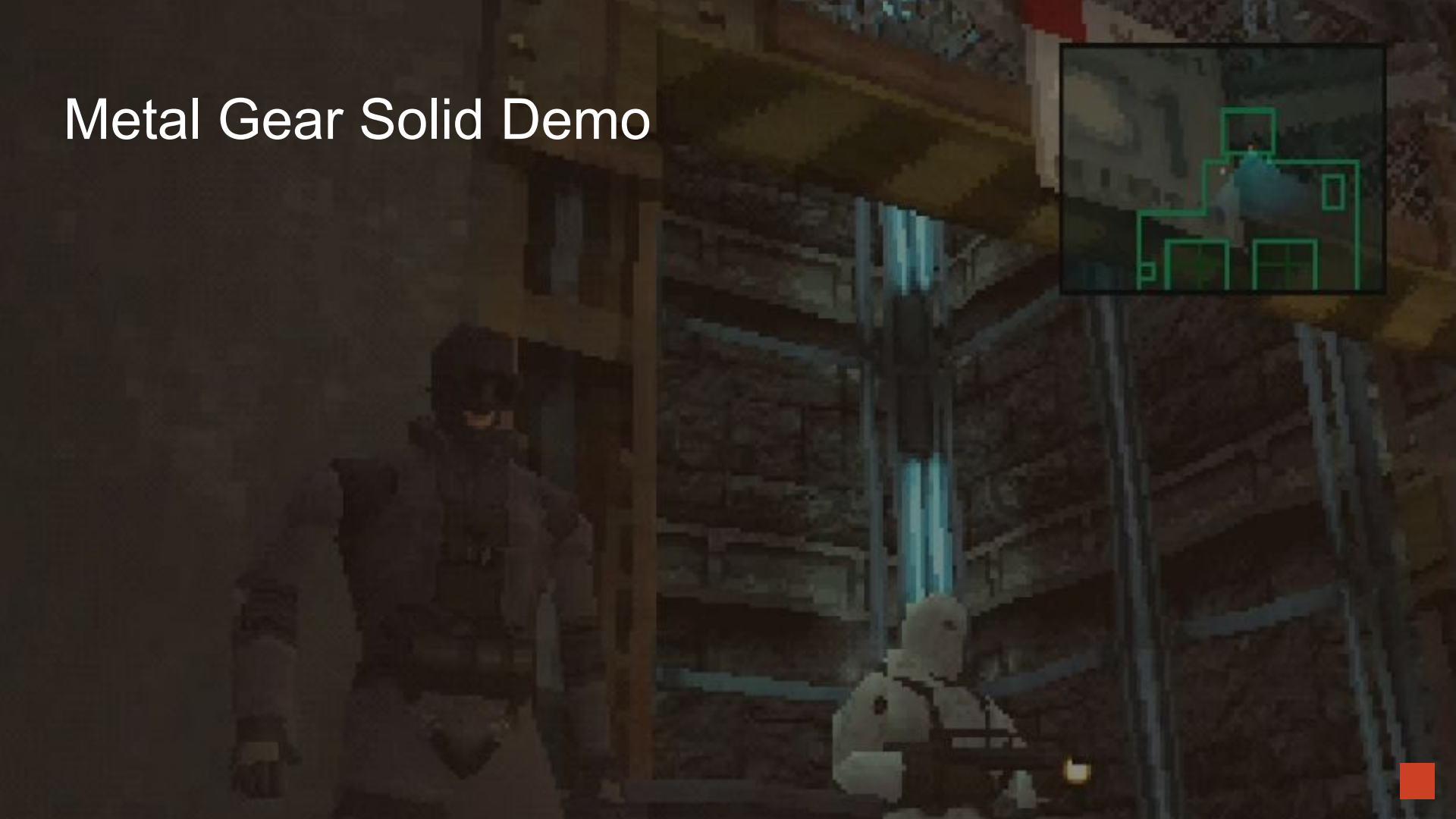
- 12-track sample-based sequencer
- Follows game states:





- Normal >> ALERT: immediately play **!** sound (diminished chord), key and tempo change, exciting music
- EVASION: some tracks fade out and tempo slows down
- EVASION >> Normal: Switch to calm background music

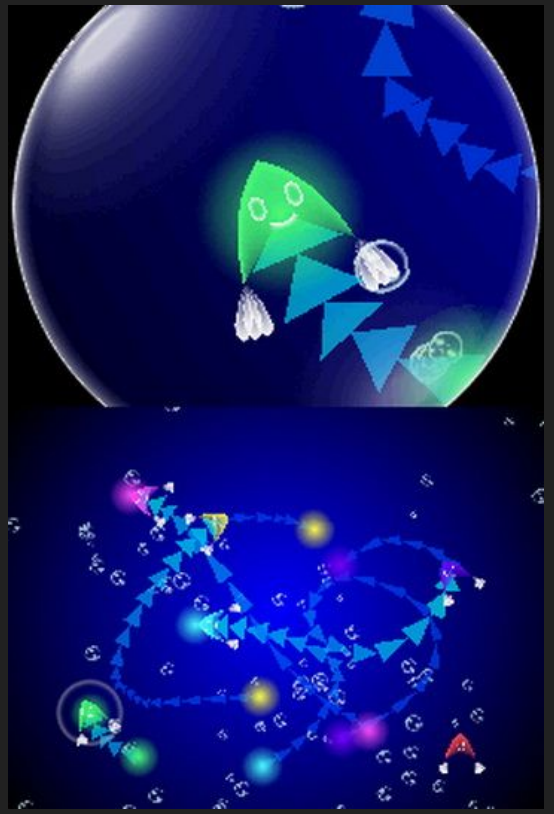
Metal Gear Solid Demo



Dynamic Music in Games: Newer examples

Electroplankton

- NDS 2005
- Generative music toy



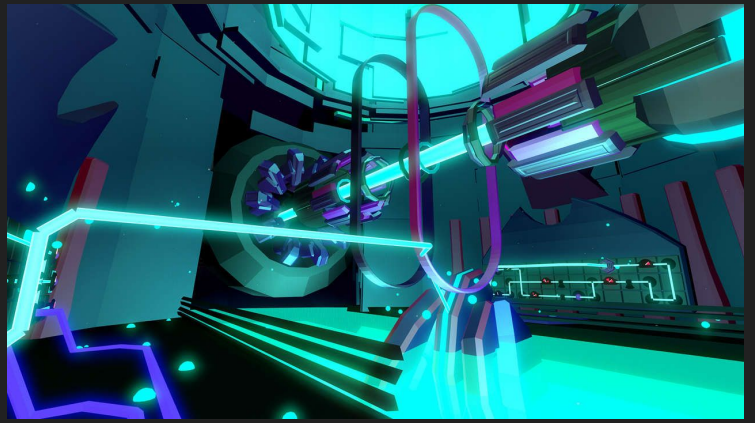
Pure Data in Games

Spore (PC, 2008)

Life simulation game with generative PD score by Brian Eno

FRACT OSC (PC, 2014)

First-person puzzle game where you construct a realtime-synthesized piece of music



Rise of the Tomb Raider

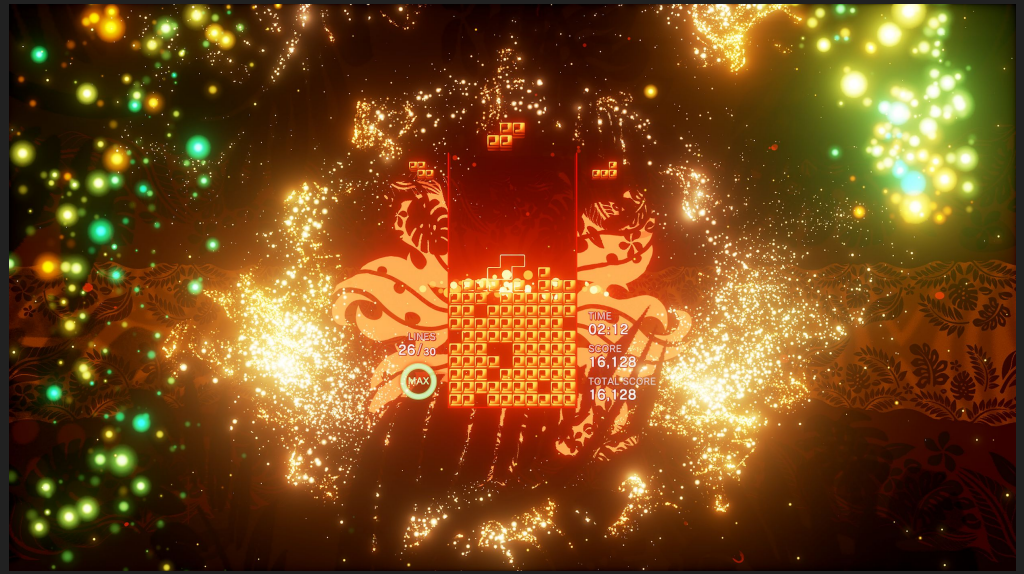
- PS4, Xbox One 2015
- Dynamic Percussion System for battle sequences
- Generated drum sequence that reacts to battle intensity level

Control

- PS4, Xbox One 2019
- Martin Stig Andersen
- Micro-sequencing

Tetris Effect

- PS4, Xbox One 2018
- Quantizes player input to beats and triggers samples in time with music
- Samples are pitched to reflect key changes in music
- Each level is an music toy that the player can play with



Tetris Effect Video

Rytmos

- Switch, Windows 2023
- Music by Niels Böttcher
- Dynamic sequencing based on puzzle state
- Features many non-well tempered tunings
- A few realtime Unity Native Audio Plugin effects by me

Rytmos Video

Summary

- Dynamic music in games since 1980s
- Dynamic music create variation and can express game state
- A few games generate music via Pure Data
- Quantized player input can be directly used for sequencing music

Audio Plugins

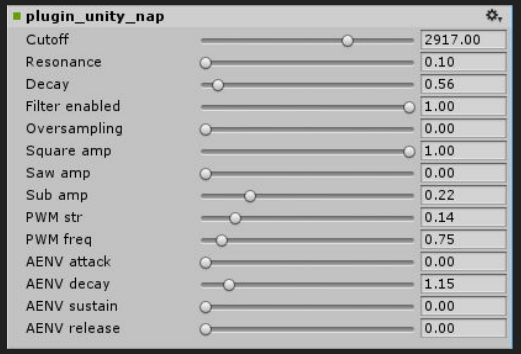
Modern Realtime Synthesis

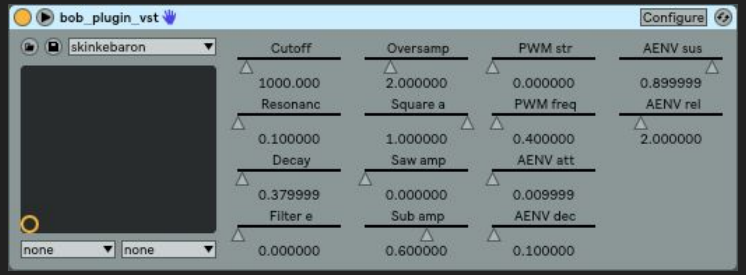
- Implemented as audio plugins in sound engines
- Normally rendered on CPU, not in dedicated hardware

FMOD Studio plugin

Audio Plugin Types

- FMOD Studio Plugin
- Wwise Sound Engine Effect Plugin
- Unity Native Audio Plugin
- VST 2.4
- Audio Units (Core Audio)





What is an Audio Plugin?

- A piece of code that outputs samples to an audio buffer
- Some wrapping that enables parameters and stuff

Audio Buffers

An audio buffer is a block of memory containing samples:

```
S0 S1 S2 S3 S4 S5 S6 S7
```

Rendering to Audio Buffer

An audio buffer is a block of memory containing samples:

```
buffer -> S0 S1 S2 S3 S4 S5 S6 S7
float [] buffer = new float[SAMPLE_COUNT];
```

Rendering code fills buffer with samples:

```
void process(float [] output, int length)
{
    for(int s = 0; s < length; ++s)
        output[s] = COMPUTE SAMPLE;
}
```

Stereo Audio Buffer

An interleaved stereo audio buffer:

```
L0 R0 L1 R1 L2 R2 L3 R3
```

Rendering to Stereo Audio Buffer

An interleaved stereo audio buffer:

```
L0 R0 L1 R1 L2 R2 L3 R3
```

Rendering code:

```
float [] buf = new float[SAMPLE_COUNT * 2];
void process(float [] output, int length) {
     int idx = 0;
     for(int s = 0; s < length; ++s) {
         output[idx++] = COMPUTE LEFT SAMPLE;
         output[idx++] = COMPUTE RIGHT SAMPLE;
     }
}
```

Synths vs. Effects

Implemented exactly the same way, except:

- Effects receive audio input
- Synths receive note and parameter input

Effect Rendering

Example code for a mono effect:

```
float [] input  = new float[SAMPLE_COUNT];
float [] output = new float[SAMPLE_COUNT];

void process(float [] input, float [] output, int length)
{
    for(int s = 0; s < length; ++s)
        output[s] = COMPUTE SAMPLE FROM input[s];
}
```

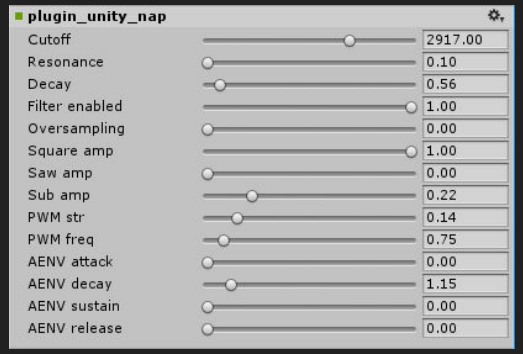
Summary

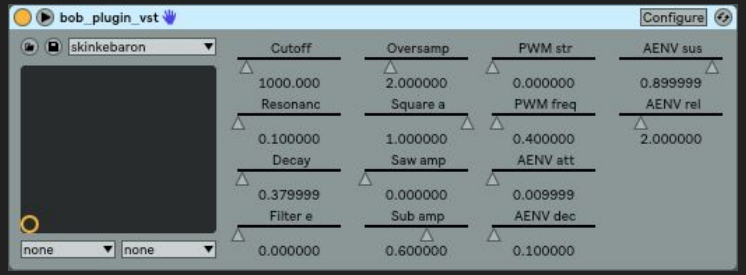
- Realtime synthesis is done using software audio plugins
- Different audio software have different plugin types
- Audio plugins output samples to audio buffer
- Synths and effects are very similar, except for their input

Plugin Platforms

Plugin Platforms

Same code,
different
platforms



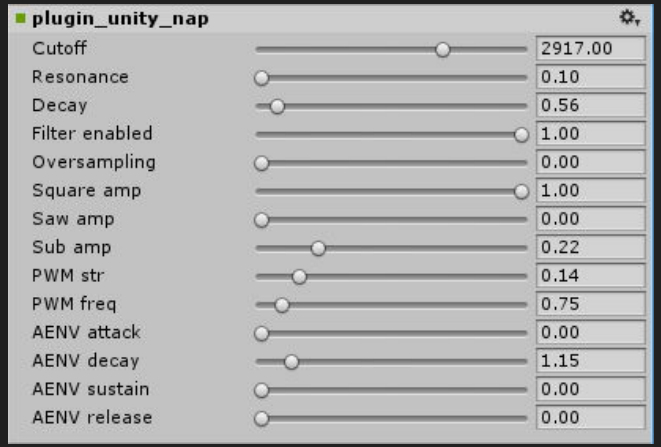


FMOD Studio Plugin

```
FMOD_RESULT F_CALLBACK Plugin_FMOD_dspprocess(
    FMOD_DSP_STATE *dsp,
    unsigned int length,
    const FMOD_DSP_BUFFER_ARRAY * inbufferarray,
          FMOD_DSP_BUFFER_ARRAY *outbufferarray,
           [..] )
    {
        RENDER length SAMPLES TO outbufferarray->buffers[0]
        return FMOD_OK;
    }
```

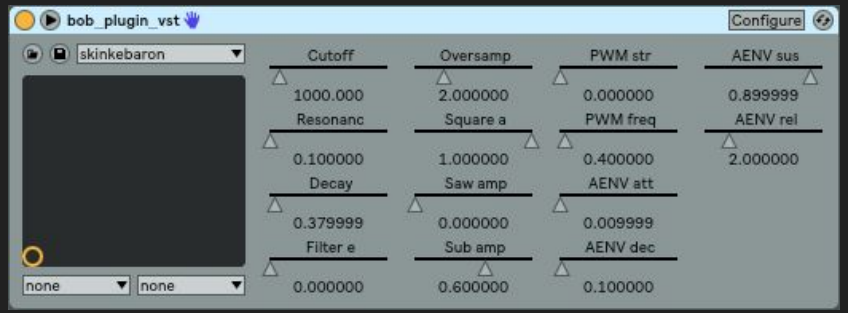
Unity Native Audio Plugin

```
[..] ProcessCallback([..],
     float* inbuffer, float* outbuffer, unsigned int length,
     int inchannels, int outchannels)
     {
         RENDER length SAMPLES TO outbuffer
     }
```



VST 2.4

```
void VstXSynth::processReplacing(
     float** inputs, float** outputs,  // input / output - buffers
     VstInt32 sample_frames )          // buffer size
{
     // not interleaved, left and right are separate
     float* buf_left  = outputs[0];
     float* buf_right = outputs[1];
     RENDER sample_frames SAMPLES TO buf_left AND buf_right
}
```



Summary

- Same code can easily be adapted for different plugin platforms
- FMOD Studio, Unity Native Audio Plugins, and VST 2.4 have similar interfaces

Example Unity Plugins

Unity C# Plugin Structure

```
class MySynthBehaviour : MonoBehaviour
{
    [...]
     void OnAudioFilterRead(float[] data, int channels)
     {
          int length = data.Length / channels;
          int idx = 0;
          for (int s = 0; s < length; ++s)
          {
               data[idx++] = COMPUTE LEFT SAMPLE
               data[idx++] = COMPUTE RIGHT SAMPLE
          }
     }
}
```

Sine Synth

```
float phase = 0.0f;
float freq = 200.0f;
const float secondsPerSample = 1.0f / 48000.0f;
void OnAudioFilterRead(float[] data, int channels)
{
    int length = data.Length / channels;
    int idx = 0;
    for (int s = 0; s < length; ++s)
    {
        float out = Mathf.Sin(phase * Mathf.PI * 2.0f);
        data[idx++] = out; // left channel
        data[idx++] = out; // right channel
        phase += freq * secondsPerSample;
        if(phase > 1.0f) phase = 0.0f;
    }
}
```

Distortion Effect (from 140)

```
int D = 0; // downsample factor
void OnAudioFilterRead(float[] data, int channels)
{
    if(D > 1)
    {
        for (int s = 0; s < data.Length; s+=2)
        {
            data[s]   = data[s / D * D]; // left channel
            data[s+1] = data[s / D * D + 1]; // right channel
        }
    }
}
```

Music Code Example

```
class SpookyBeat : MonoBehaviour
{
    float s = 0;

    void OnAudioFilterRead(float[] data, int channels)
    {
        int smp = 0, length = data.Length;

        while (smp < length)
        {
            s = ++s % 288000;
            float p = (s / 288000) * 0.5f;
            float pBar = (p * 8) % 1;
            float hhAmp = (0.13f + ((pBar * 4) % 1) * -0.09f);

            // mixer
            float output = BD(pBar * 8 / 3) * 0.8f
                + HH((pBar * 8) % 1) * hhAmp
                + bass(p) * 0.2f + bass(p - 0.024f) * 0.1f;

            for (int c = 0; c < channels; ++c)
                data[smp++] = output;
        }
    }

    // Bassdrum: sine with pitch and amplitude envelope
    float BD(float p)
    {
        float env = Mathf.Clamp01(0.1f - (p % 1f)) * 10f;
        float fr = 30f + env * 100f;
        float ph = (p % 1f) * fr;
        return Mathf.Sin((ph % 1f) * 6.28f) * env;
    }

    // Hihat: noise with amplitude envelope
    float HH(float p)
    {
        return Mathf.PerlinNoise(p * 2000, 0f) * (1f - p);
    }

    // Spooky bass: FM synth
    float bass(float p)
    {
        return Mathf.Sin(p * 4000 + Mathf.Sin( p * 4000
             + Mathf.Sin(p * 3.28f) * 1111))
             * Mathf.Sin(((p * 64 / 3f) % 1) * 3.141f);
    }
}
```

Summary

- Unity audio plugins can be written in C#
- Unity audio plugins have the same structure as other audio plugins
- Example synth and distortion effect
- Example music code

References

Karen Collins: "An Introduction to Procedural Music in Video Games" (2009)
https://bit.ly/2FfuN6E

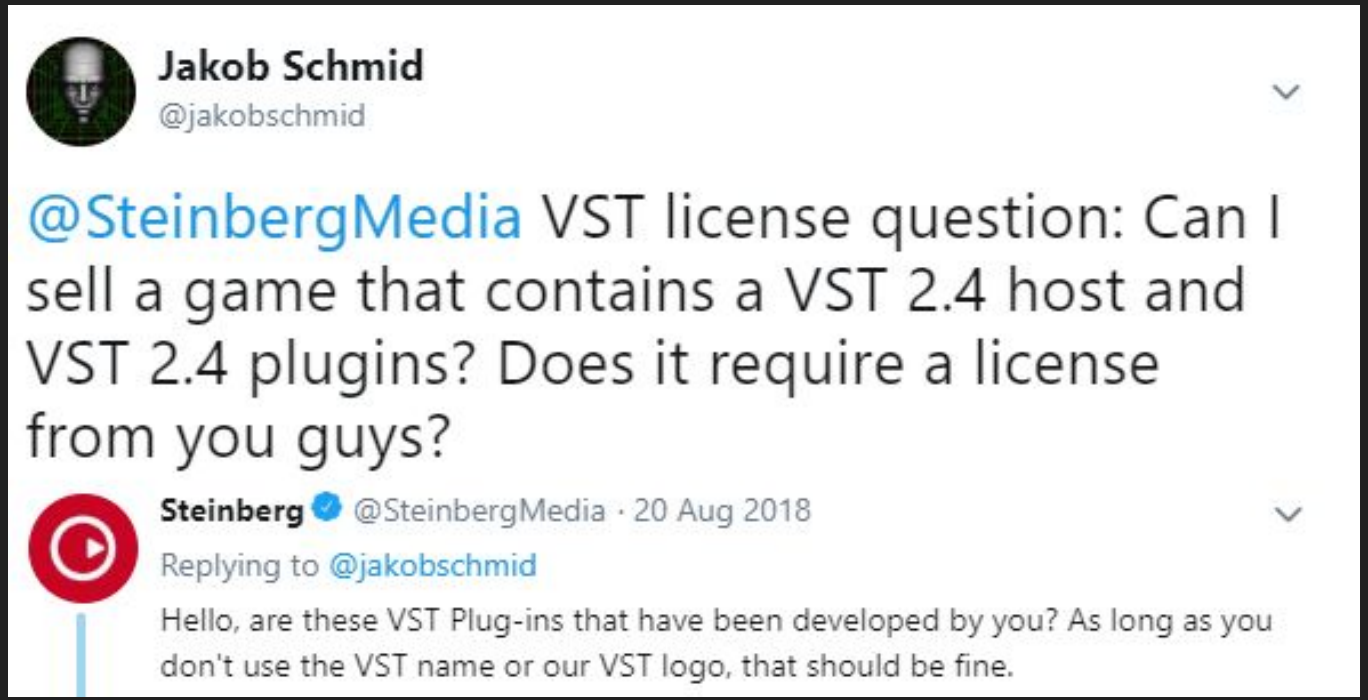
Igor Dall'Avanzi: "Procedural Music in AAA: Rise of the Tomb Raider and the Dynamic Percussion System" (2016)
https://bit.ly/2HMEvjJ

Leonard J. Paul lectures about Pure Data for games
https://bit.ly/2FnIGjo

Jakob Schmid: "Metal Gear Solid (PSX) Dynamic Music"
https://youtu.be/naTy9fhrjCo

Questions?

VST Plugins or Audio Units in Games?



VST Plugins or Audio Units in Games?

If plugin is open source or homemade:

- Relatively easy to adapt to game audio plugin

VST Plugins or Audio Units in Games?

Most interesting VST/AU plugins are *not* open source.

Technically they could still work in a game, however:

- Illegal distribution: Most VST/AU plugins licensing models do not allow for redistributing to potentially millions of users in a game.
- Limited platforms: Most VST/AU plugins are available in binary form for Windows and Mac OS X, but not for Android, iOS, PS4, Xbox One, etc. so would only work on computers.

VST Plugins or Audio Units in Games?

- Possible.
- Not practical!

Audio Plugin Interface

- Audio system calls our code with buffer
- Our code writes samples to buffer
- Audio hardware outputs buffer to speaker

Wwise Sound Engine Effect Plugin

```
void IAkOutOfPlaceEffectPlugin::Execute(
     AkAudioBuffer * io_pInBuffer,   // input buffer
     AkUInt32        in_uInOffset,   // offset
     AkAudioBuffer * io_pOutBuffer ) // output buffer
{
     float *buf = io_pOutBuffer->GetChannel(0);
     RENDER [FIXME - how many samples?] TO buf
}
```

Dead Space

- Xbox 360, PS3 2008
- Uses traditional dynamic orchestral music
- Atonal orchestral stings are triggered by the player seeing a mutant for the first time

DEMO: Example Plugins in Action

- Standalone
- Unity Native Audio Plugin
- FMOD Studio
- VST 2.4

Atari 2600 TIA Chip

- Integrated graphics and sound
- 2 DCOs pulse waveform
- 32 pitch values (not enough)
- 4 bit volume