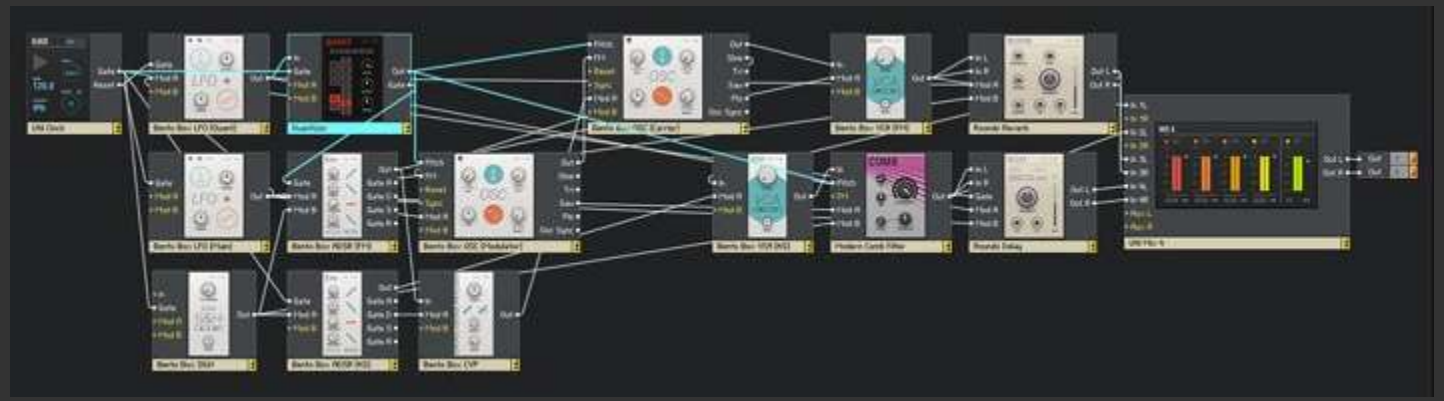
Elektroniczne instrumenty muzyczne

KOMPUTEROWE NARZĘDZIA MUZYCZNE



Opracowanie: Grzegorz Szwoch, Politechnika Gdańska, Katedra Systemów Multimedialnych

Wprowadzenie

- W przeszłości muzycy, którzy chcieli tworzyć elektroniczne brzmienia musieli kupować drogie instrumenty.
- Upowszechnienie się komputerów osobistych i stały wzrost ich mocy obliczeniowej spowodowały, że komputer stanowi obecnie ważne narzędzie dla muzyka i pozwala wykonać wiele rzeczy w domowym studio.
- Najważniejsze metody wykorzystania komputera w muzyce:
  - domowe studio nagrań – programy typu DAW,
  - uruchamianie programowych instrumentów (np. VST),
  - współpraca ze sprzętowymi instrumentami (np. sekwencer MIDI),
  - tworzenie własnych brzmień za pomocą programów komputerowych.

Cyfrowe stacje robocze (DAW)

- Programy DAW – cyfrowe stacje robocze (*digital audio workstation*).
- Oprogramowanie do wielościeżkowej rejestracji i edycji nagrań muzycznych.
- Oprócz rejestracji cyfrowego dźwięku (wokal, instrumenty) pozwalają też na rejestrację i edycję komunikatów MIDI (funkcje sekwencera MIDI).
- Komunikaty MIDI mogą sterować zewnętrznym sprzętowym instrumentem.
- Programy DAW pełnią również rolę *hosta* dla programowych instrumentów w formie wtyczek (*plugin*), w formatach VST, AU. Komunikaty MIDI powodują wytwarzanie dźwięku przez programowy syntezator lub sampler.
- Nagrania muzyki elektronicznej mogą niemal w całości (oprócz wokalu) zostać zrealizowane w domowym studio, z użyciem komputera.

VST

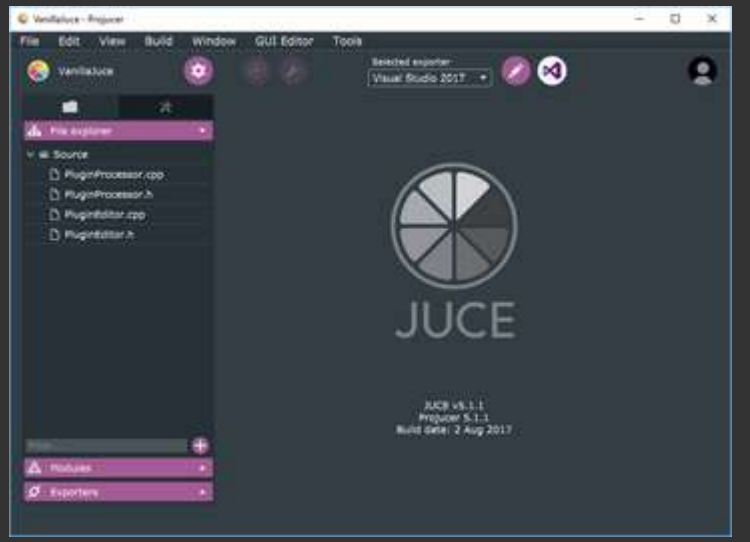
- VST – *Virtual Studio Technology*. Standard opracowany przez firmę Steinberg.
- Programiści mają do dyspozycji SDK – zbiór gotowych procedur.
- Programista tworzy wtyczkę (*plugin*) w formacie VST.
- Efekty VST – przetwarzają dźwięk cyfrowy (np. dodają pogłos).
- Instrumenty VST (VSTi) – odbierają kody MIDI i generują cyfrowy dźwięk, dowolną metodą (wybrana metoda syntezy, sample, itp.).
- Wtyczki są uruchamiane w programie *hosta* – tę rolę pełni program DAW. Przykład darmowego hosta VST: *Cantabile Lite*.
- Aby uruchomić pojedynczą wtyczkę – instrument, można skorzystać z prostego, darmowego programu *SAVIHost*.

Tworzenie instrumentów VST

- Zadaniem programisty instrumentu VST jest:
  - napisanie algorytmu, który generuje (dowolną metodą) dźwięk cyfrowy o parametrach zdefiniowanych w kodach MIDI otrzymanych na wejściu,
  - stworzenie interfejsu użytkownika (GUI),
  - zdefiniowanie i interpretacja parametrów, które wpływają na sposób wytwarzania dźwięku (automatyzacja z poziomu hosta).
- Programista nie musi martwić się o funkcje wejścia/wyjścia, to załatwia za niego host.
- Obecnie dostępna jest wersja VST3 SDK. Upraszcza ona kwestie licencyjne, ale tworzenie instrumentów jest bardziej problematyczne niż w starszym VST2 (obsługa MIDI nie jest priorytetem w VST3).

Nakładki - JUCE

- Ze względu na ograniczenia i trudną obsługę VST SDK, wielu programistów woli wykorzystywać tzw. nakładki (*wrapper*).
- JUCE jest najbardziej popularnym oprogramowaniem do tworzenia programowych instrumentów (i nie tylko).
- Kod pisany w C++ w JUCE może być kompilowany do wielu formatów: samodzielny plik EXE (dla rożnych systemów), wtyczki VST, AU i wiele innych.
- Dostarcza wiele funkcji do tworzenia interfejsów użytkownika, generowania i przetwarzania dźwięków, itp.
- Podwójna licencja: GPLv3 i komercyjna.



Komputerowe języki muzyczne - *CSound*

- Języki programowania ogólnego zastosowania nie umożliwiają wygodnego tworzenia muzyki. Powstały specjalne języki programowania (zwykle skryptowe) służące do tego celu.
- CSound – jeden z najczęściej używanych języków programowania dla muzyki.
- Kod programu składa się z dwóch części:
  - *orchestra* – instrukcje tworzące dźwięki (instrumenty),
  - *score* – instrukcje budujące muzykę z tych dźwięków.
- Duże możliwości, ale dość trudny do nauczenia się.
- Istnieje dużo dokumentacji i przykładów.

CSound

Przykład skryptu w *CSound*:
realizuje prostą, dwuoperatorową syntezę metodą FM.

http://booki.flossmanuals.net/csound/d-frequency-modulation/

CsInstruments:
definicja sposobu generowania dźwięków.

CsScore:

dane do generowania dźwięków
(wysokość, czas trwania, głośność, itp.)

```
<CsoundSynthesizer>
<CsInstruments>
sr = 48000
ksmps = 32
nchnls = 2
0dbfs = 1
instr 1

kCarFreq = 440      ; carrier frequency
kModFreq = 440      ; modulation frequency
kIndex = 10         ; modulation index

kIndexM = 0
kMaxDev = kIndex*kModFreq
kMinDev = kIndexM*kModFreq
kVarDev = kMaxDev-kMinDev
kModAmp = kMinDev+kVarDev

; oscillators
aModulator poscil kModAmp, kModFreq, 1
aCarrier poscil 0.3, kCarFreq+aModulator, 1

outs aCarrier, aCarrier
endin
</CsInstruments>

<CsScore>
f 1 0 1024 10 1     ; Sine wave for table 1
i 1 0 15
</CsScore>

</CsoundSynthesizer>
; written by Alex Hofmann (Mar. 2011)
```

SuperCollider

- Nowszy język skryptowy do programowania muzyki, podobny do *CSound*.
- Przeznaczony do syntezy dźwięku oraz komponowania algorytmicznego.
- Działa w architekturze „klient-serwer”. Składa się z silnika dźwięku (*scsynth*) i interpretera kodu (*sclang*).
- Umożliwia tworzenie interfejsów użytkownika.
- Opracowany z myślą o wykorzystaniu do sterowania procesem generowania muzyki w czasie rzeczywistym, „na żywo” (*live coding*).
- Dostępny dla wielu systemów operacyjnych.
- Mniejsza liczba przykładów niż dla *CSound*.

SuperCollider - przykład

Przykład skryptu *SuperCollider*: synteza FM

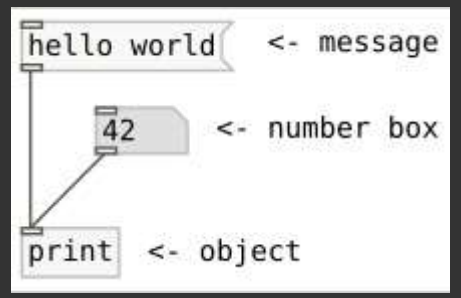
http://danielnouri.org/docs/SuperColliderHelp/Tutorials/Mark_Polishook_tutorial/Synthesis/14_Frequency_modulation.html

```
(
SynthDef("fm1", { arg bus = 0, freq = 440, carPartial = 1, modPartial = 1, index = 3, mul = 0.05;
            // carPartial :: modPartial => car/mod ratio

            var mod;
            var car;

            mod = SinOsc.ar(freq * modPartial, 0, freq * index * LFNoise1.kr(5.reciprocal).abs);

            car = SinOsc.ar( (freq * carPartial) + mod, 0, mul);

            Out.ar(bus, car)
}).load(s);
)

(
Synth("fm1", [\bus, 0, \freq, 440, \carPartial, 1, \modPartial, 1, \index, 10]);
)

(
s.queryAllNodes;
)
```

Max i Pure Data (pd)

- Tworzenie kodu programu nie jest intuicyjne dla muzyków.
- Alternatywne podejście: graficzne programowanie za pomocą schematu.
- System *Max* opracowany przez Millera Puckette, implementacje:
  - *Max* – komercyjna,
  - *Pure Data* (pd) – *open source*.
- Przetwarzanie dźwięku, synteza, sampling, obsługa MIDI i wiele innych.
- Łączenie bloków za pomocą linii. Dość ascetyczny interfejs użytkownika w *pd*.
- Bardziej „poglądowy” sposób programowania,
  ale skomplikowane układy mogą być „poplątane”.
- Dużo dostępnych przykładów.

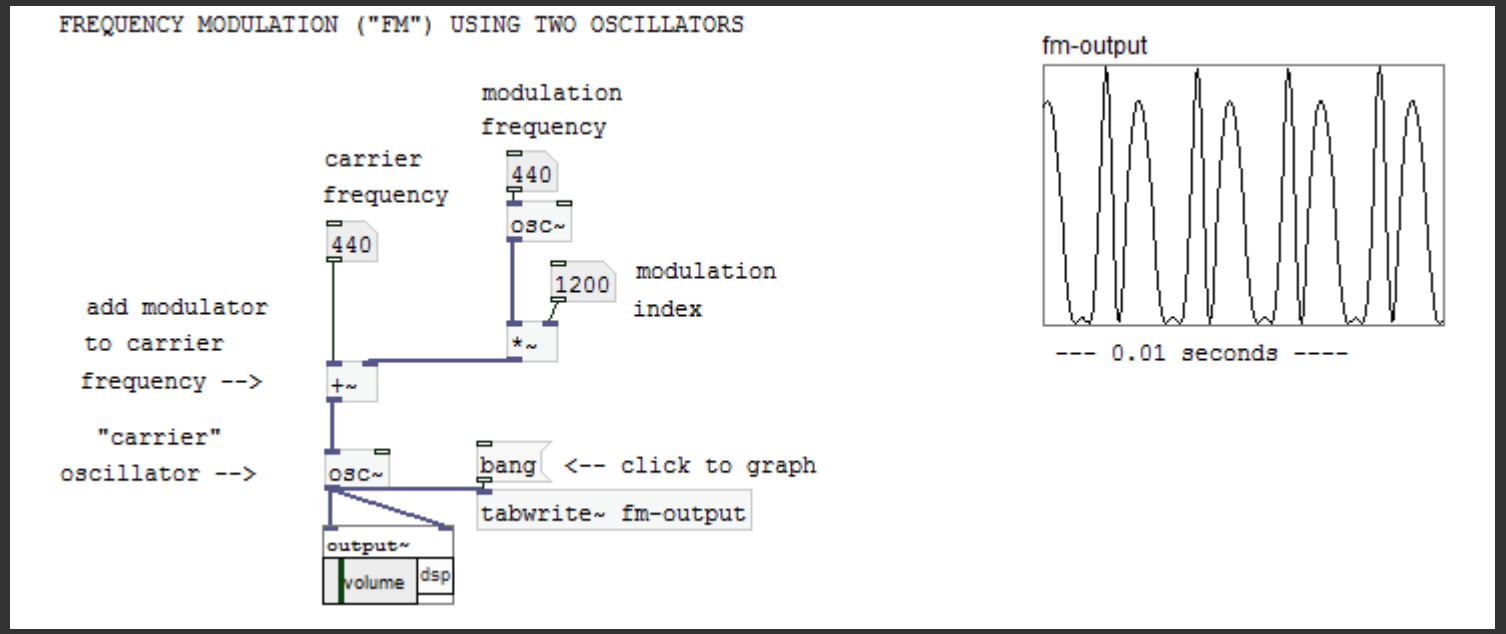
pd~



Pure Data - przykład

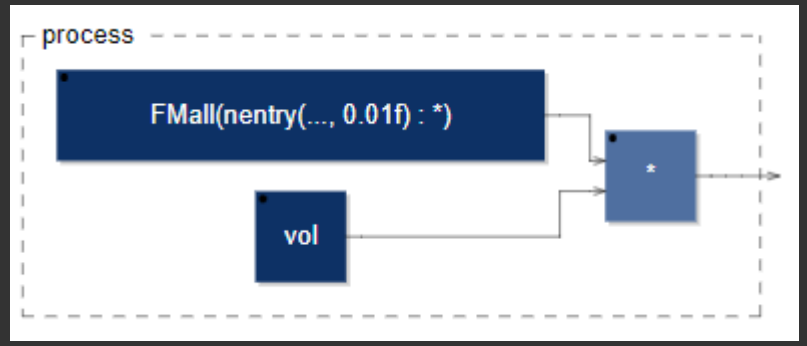
Układ syntezy *Simple FM* w programie *pd*

z przykładu dostarczonego z programem (*A09.frequency.mod.pd*)



Faust

- Faust (*Functional audio stream*) jest językiem programowania ukierunkowanym na syntezę dźwięku, tworzenie programowych instrumentów i efektów, itp.
- Podstawą jest schemat blokowy, opisujący algorytm przetwarzania sygnału.
- Schemat jest zapisywany z użyciem wewnętrznego języka programowania.
- Kod jest kompilowany, powstaje program – samodzielny lub wtyczka.
- Przykłady: https://faustcloud.grame.fr/doc/examples/index.html



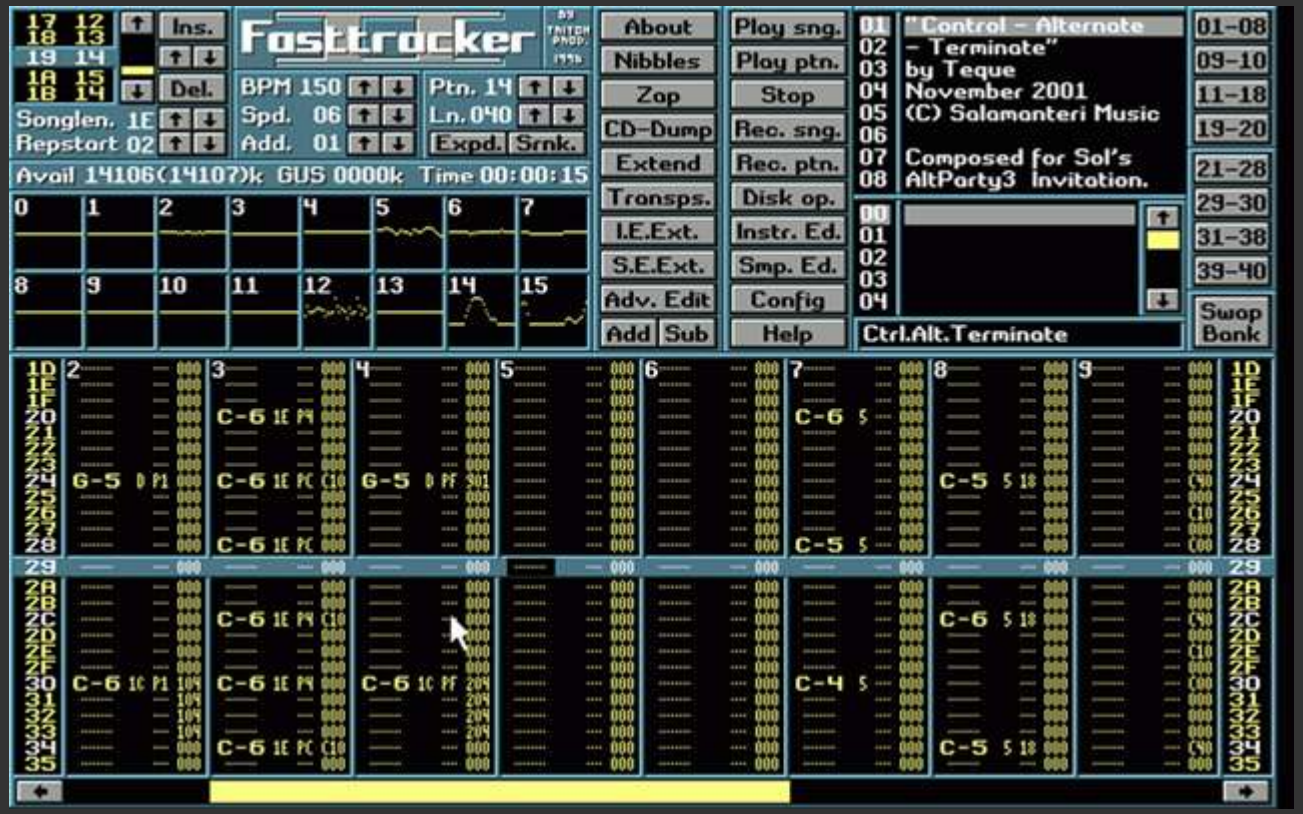
FAUST

Trackery

- Trackery powstały w latach 80. Były wykorzystywane przez *demoscenę* do tworzenia demów – wizualizacji z podkładem muzycznym.
- Tracker łączy prosty polifoniczny instrument samplingowy z sekwencerem.
- Krótkie próbki (sample) z nałożoną obwiednią tworzą instrumenty.
- Wzorzec (*pattern*) to tablica reprezentująca oś czasu (*timeline*) oraz ścieżki (*track*) zawierające osobne głosy.
- Wzorzec zawiera nuty (*note*) – numer instrumentu, wysokość, głośność, efekty brzmieniowe.
- Z wzorców układany jest utwór muzyczny – sekwencja (*order*).
- Bardzo zwięzły opis – trzyminutowy utwór zapisany jest w pliku – module, o rozmiarze kilkudziesięciu kilobajtów.

Trackery

Fasttracker (komputery Amiga)



Wzorzec w *OpenMPT*

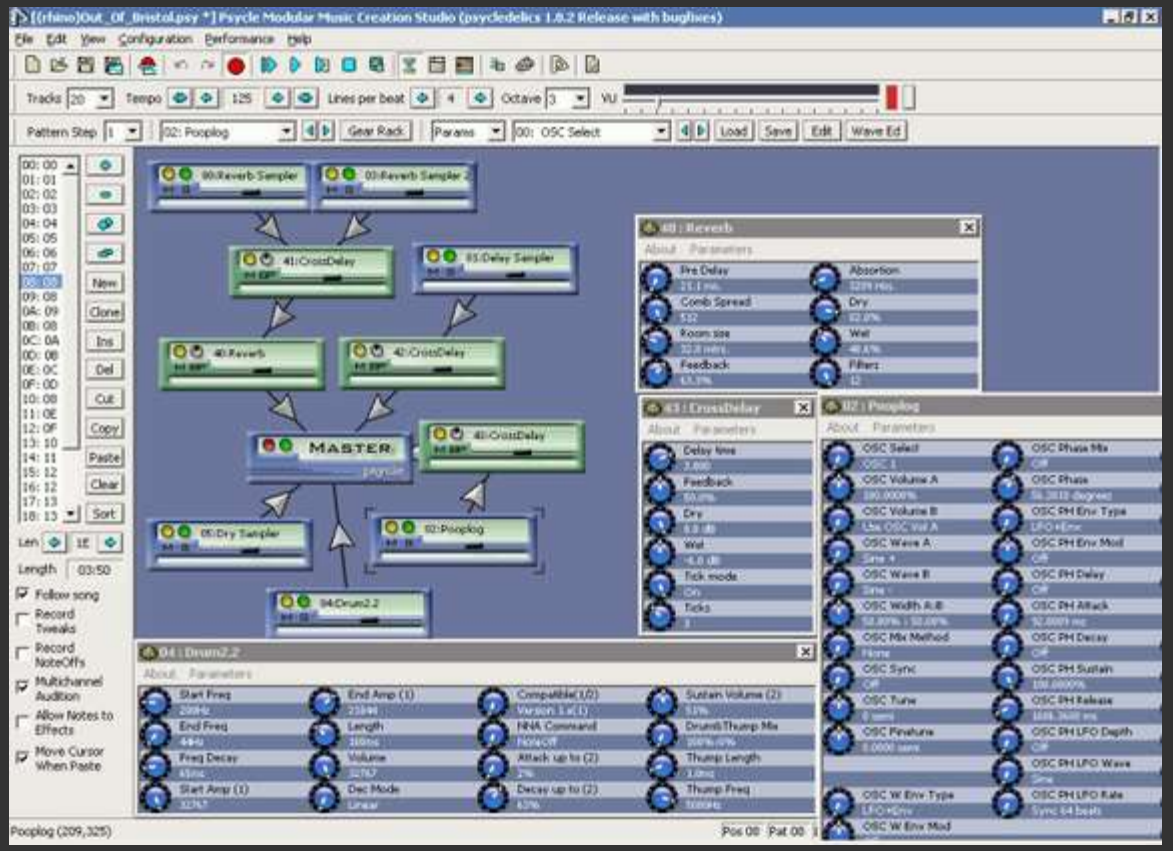
0	E-4 26 v64 V80	E-4 26 v64 004	... .	... .
1	... .	... .	A-4 26 v64 ...	A-4 26 v64 004
2	... .	... .	... .	... .
3	... . SEE	... . S66	... . S66	... . S66
4	G-5 26 v64 ...	G-5 26 v64 004	... .	... .
5	... .	... .	D-5 26 v64 ...	D-5 26 v64 004
6	... .	... .	... .	... .
7	... . SEE	... . S66	... . S66	... . S66
8	A-4 26 v64 ...	A-4 26 v64 004	... .	... .
9	... .	... .	D-5 26 v64 ...	D-5 26 v64 004
10	... .	... .	... .	... .
11	... . SEE	... . S66	... . S66	... . S66
12	G-5 26 v64 ...	G-5 26 v64 004	... .	... .
13	... .	... .	E-5 26 v64 ...	E-5 26 v64 004
14	... .	... .	... .	... .
15	... . SE7	... . S66	... . S66	... . S66
16	D-4 26 v64 ...	D-4 26 v64 004	... .	... .
17	... .	... .	F-4 26 v64 ...	F-4 26 v64 004
18	... .	... .	... .	... .
19	... . SE7	... . S66	... . S66	... . S66
0	E-4 26 v64 ...	E-4 26 v64 004	... .	... .
1	... .	... .	A-4 26 v64 ...	A-4 26 v64 004
2	... .	... .	... .	... .

Trackery modularne

Współczesne programy typu *tracker* często zamiast prostych instrumentów z próbek wykorzystują złożone układy modułów:

- generatory – syntezatory, samplery,
- efekty – przetwarzanie brzmienia,
- wzmacniacze, miksery i inne.

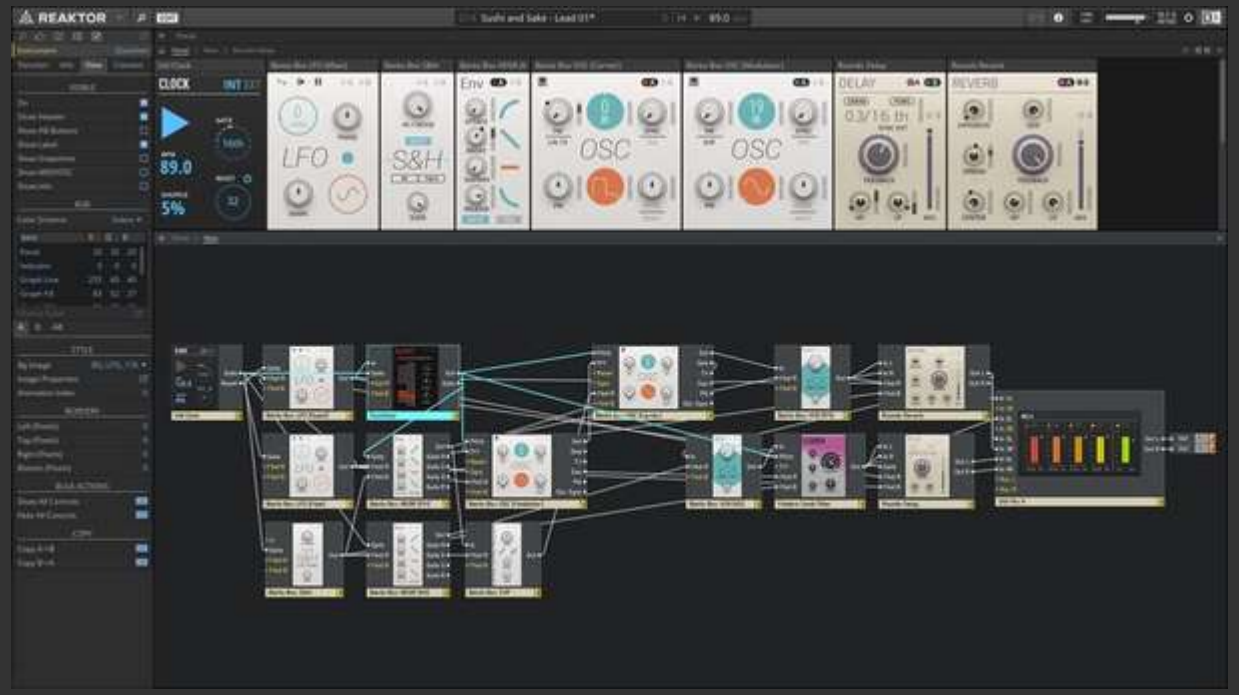
Przykład: *Psycle*



Modularne układy tworzenia dźwięku

Oprogramowanie modularne – dostarcza gotowych modułów (często z możliwością dodawania nowych) oraz środowisko do ich łączenia w celu zbudowania własnego instrumentu generującego dźwięki muzyczne.

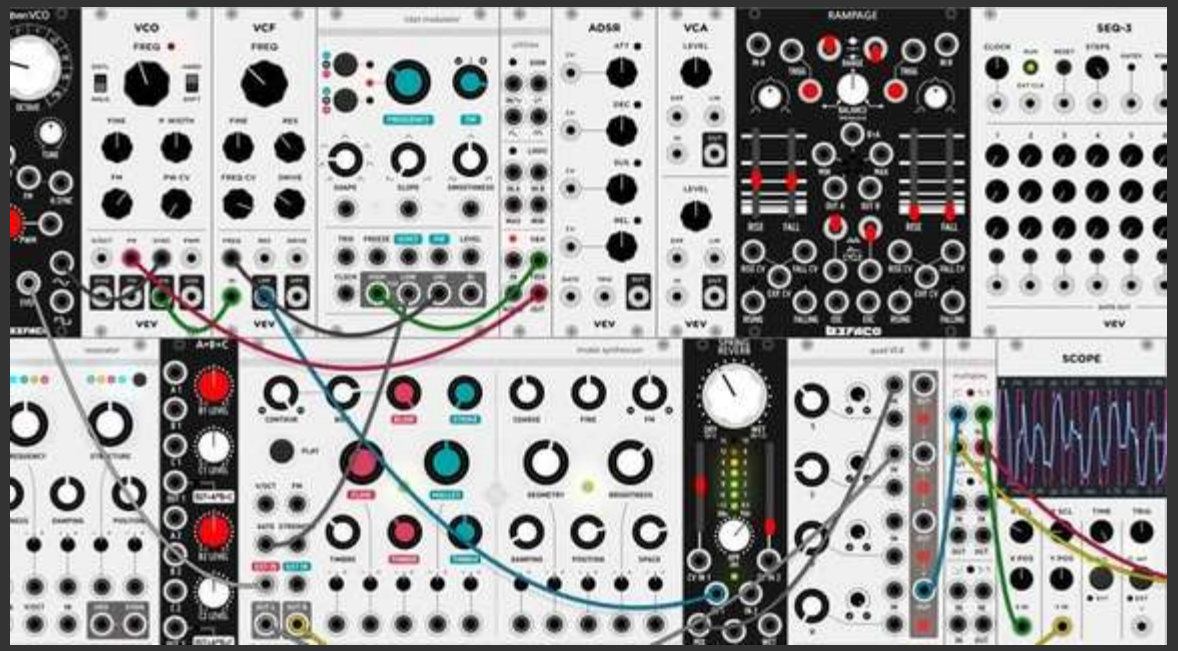
Przykład: NI *Reaktor*



Modularne układy tworzenia dźwięku

VCV Rack – modularne środowisko pomyślane jako cyfrowa emulacja systemu *Eurorack* do realizacji syntezy subtraktywnej.

Można zbudować własny syntezator subtraktywny łącząc moduły – dostarczone z programem (darmowe i komercyjne) oraz stworzone samodzielnie.



Programy do kompozycji algorytmicznej

- Kompozycja algorytmiczna (*algorithmic composition*) polega na tworzeniu muzyki, a nie pojedynczych dźwięków, przez komputerowy algorytm.
- Program komputerowy sam komponuje utwory muzyczne.
- Podejścia algorytmiczne (często się je łączy):
  - oparte na teorii muzyki (gramatyczne),
  - oparte na modelach statystycznych,
  - oparte na modelach stochastycznych (losowe),
  - fraktalne (metody chaosu deterministycznego),
  - oparte na sztucznej inteligencji, w tym na głębokim uczeniu.
- Przykład: *cgMusic* (autor: Maciej Biedrzycki)
  (http://codeminion.com/blogs/maciek/2008/05/cgmusic-computers-create-music/)

Literatura

- VST3 SDK (Steinberg), dla twórców wtyczek VST: https://steinbergmedia.github.io/vst3_dev_portal/pages/
- SAVIHost (host wtyczek VST): http://www.hermannseib.com/english/savihost.htm
- CSound: http://www.csounds.com/
- SuperCollider: http://supercollider.github.io/
- Pure Data (pd): https://puredata.info/
- *Psycle*: http://psycle.pastnotecut.org
- Wikipedia – kompozycja algorytmiczna: https://en.wikipedia.org/wiki/Algorithmic_composition