AKADEMIE VÝTVARNÝCH UMĚNÍ V PRAZE
Ateliér intermediální tvorby III

DISERTAČNÍ PRÁCE

Pure Data

Rukověť postdigitálního umělce
ver. 0.1a

Květen 2014

Vypracoval: Michal Cáb
Vedoucí práce: Tomáš Vaněk
Konzultant: Miloš Vojtěchovský

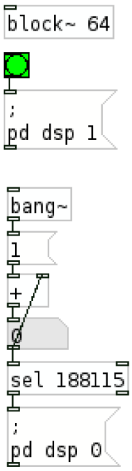
Ádě Taübelové

V inicializaci této práce sehrály bezesporu svou roli Letné dielne, poděkování patří tedy Márii Riškové, Magdaléně Kobzové, Dušanu Barokovi a dalším organizátorům. Výsledná podoba textu předkládané práce byla v dobrém slova smyslu proměněna po kritickém a věcném připomínkování, které mi poskytl Milan Guštar. Díky patří též mému konzultantovi Miloši Vojtěchovskému, zejména za zprostředkování alternativních studijních pramenů. Za naslouchání, diskusi nad tématem a pedagogické vedení děkuji Tomáši Vaňkovi. Za jazykovou korekturu Regině Peškové. Děkuji též všem umělcům, kteří mi poskytli svůj čas, texty a kódy. Jsou jimi Michal Kindernay, Jiří Rouš, Tomáš Šenkyřík, Peter Gonda, Jakub Pišek a Martin Blažíček. Při práci mi byli oporou mí přátelé, se kterými jsem řadu témat diskutoval a zohledňoval jejich připomínky – díky patří Kryštofu Peškovi, Davidu Landovi, Matouši Hejlovi, Ladislavu Čumbovi a řadě členů Pd komunity. V neposlední řadě patří poděkování mým rodičům a všem blízkým, kteří se mnou po čas mé práce na disertaci měli trpělivost.

Prohlašuji, že jsem tuto disertační práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

V _____ dne _____ _____

silence-computated@SR44100.pd
hommage à john cage



Obsah

1 Intro **1**

1.1 O metodě a předmětu . . . 2
1.2 O struktuře . . . 4
1.3 O názvu . . . 5
1.4 Schéma souvislostí . . . 6

2 Předpoklady **7**

2.1 Číslo a písmo . . . 10
2.2 Na cestě k obecnému . . . 12
2.3 Metoda a algoritmus . . . 15
2.4 Mechanizace, neboli přimknutí k hmotě . . . 21
2.5 Elektrifikace . . . 24
2.6 Programování . . . 29

3 Rukověť **35**

3.1 Vývoj Pd . . . 38
3.2 Pd komunita aneb alternativní studijní zdroje . . . 42
3.3 Instalace a první spuštění . . . 43
Před tím, než začneme . . . 47
Základní stavební kameny . . . 49
Píšeme kód / taháme kabely / hallo world . . . 53
Tlačítka, posuvníky, vypínače . . . 55
Help, i need somebody . . . 57
Dataflow . . . 58
Počítadla a čas . . . 63
Zprávy pod drobnohledem . . . 66
Zprávy a proměnné . . . 70
Rozšiřujeme slovní zásobu . . . 73
Stavíme krokový sekvencer . . . 80
Schováváme kód do subpatche . . . 83
Slovní zásoba II. . . . 86
Paměť patche - presety . . . 90
Vstupy a interaktivita . . . 94
Pole . . . 96
Trocha matematiky . . . 97
Abstrakce . . . 102

Interní zprávy . . . 110
Síťování . . . 114
Zvuk . . . 118
Základy . . . 124
Generátory signálů . . . 129
Tvarování vln . . . 132
Obálka . . . 135
Syntetizujeme . . . 139
Vzorky neboli samply . . . 145
MIDI kontrolery . . . 151
Obraz . . . 154
Geometrické objekty a operace s nimi . . . 157
Rastrová grafika . . . 160
Pohyblivý obraz . . . 161
Propojení . . . 165

4 Aplikace **168**

4.1 Peter Ablinger: Deus Cantando . . . 169
4.2 Hans-Christoph Steiner: Solitude . . . 170
4.3 0xA: expr~ . . . 172
4.4 Roman Haefeli: Netpd . . . 173
4.5 Michal Cáb: Ty & Ra, Událost I., II. . . . 175
4.6 Michal Kindernay: Art of pollution . . . 181
4.7 Tomáš Šenkyřík: 001_start.pd . . . 185
4.8 Peter Gonda: Internal Messages VJing . . . 187
4.9 Jiří Rouš: Cesta k Pd 190
4.10 Jakub Pišek: Hallogenerator a Turbosampler . . . 194
4.11 Martin Blažíček: Pd jako „pomůcka k myšlení“ . . . 196

5 Souvislosti **203**

5.1 Postdigitalita . . . 204
5.2 Postdigitální přesahy . . . 208
5.3 Postdigitální „avantgarda“? . . . 212
5.4 DIY a FLOSS . . . 216
5.5 Ohlédnutí . . . 220

6 Outro **223**

Bibliografie **225**

1 | Intro

Jestliže je kladivo jediný nástroj, který máš k dispozici, pak se ti vše jeví jako hřebík.

Zákon nástroje

Pojednávat slovy o nám blízkých skutečnostech je nevděčná činnost, která s sebou nese řadu obtíží. Pojednávající a ona skutečnost se k sobě mohou mít až tak, že zdánlivě splývají, čímž mizí distance a tedy i možnost kritiky. Jiná obtíž také souvisí s odstupem: je takřka nemožné zachytit povahu událostí právě probíhajících – obzvláště pak, když se tyto události jeví jako dříve neviděné a nemají jasný předobraz. Za takovou událost v případě této práce budeme považovat vznik programovacího jazyka Pure Data a jeho užívání ve světě umění.

Teorie o „nových“[1] médiích, filmu, účincích elektromagnetického vlnění na živý organismus nebo o vlivu oranžové barvy na trávení, zdá se také neexistovaly jako věčné předobrazy v hyperuraniu, neobjevily se ani paralelně spolu s těmito jevy. Film přestal být literaturou nebo divadlem a našel svou specifickou tvář až po delší době skrze umělce-teoretiky, kteří díky řadě experimentů a reflexí jeho svébytnost vydobyli.[2] V případě Pure Dat a dalších programovacích jazyků se nyní možná nacházíme v analogické situaci jako tomu bylo kdysi u filmu. Hledání tváře „nových“ médií v postmoderní situaci je ale poněkud náročnějším úkolem, protože díky množství a různosti prostředků produkce (nebo snad můžeme použít odvážné slovo tvorby) jsou tato média obtížněji uchopitelná.[3] Teoretická práce popisující bezprostřední kulturní jevy, o nichž není jisté, zda se v toku událostí potvrdí nebo nikdy neopustí pomyslný inkubátor, je stejně tak křehká jako před-

[1] Slovo klademe do uvozovek kvůli nejasnostem a otázkam, které popisuje např. VOJTĚCHOVSKÝ, Miloš. Co je ještě nového na nových médiích? *A2* [online]. 2008, č. 40 [cit. 2013-01-12]. Dostupné z: http://www.advojka.cz/archiv/2008/40/co-je-jeste-noveho-na-novych-mediich. ISSN 1803-6635.

[2] Např. práce filmového režiséra a teoretika Sergeje Ejzenštejna (1898-1948) a dalších, kteří se zabývali střihovou skladbou.

[3] CSERES, Jozef a MURIN, Michal. *Od analógového k digitálnemu...* Banská Bystrica: Fakulta výtvarných umení, 2010. 219 s. ISBN 978-80-89078-78-3. s. 7-12.

měty jejího zájmu. Navzdory své křehkosti a fragmentárnosti může ale sloužit jako poukaz na význam probádávaného předmětu.

Pojednávat slovy o nám blízkých skutečnostech je nevděčná činnost. Obzvláště pak, když ke slovům zaujímáme apriorně podezřívavé stanovisko. Slovo vytváří hranici tam, kde možná žádná není. Vztah mezi označujícím a označovaným je arbitrární. Slova vytrhávají předměty z původní spojitosti a odhlížejí od těžce postihnutelných jedinečností (kolik nepostihnutelných aspektů nevyslovujeme, když říkáme např. prosté slovo strom?[4]). Navzdory těmto nevýslovným metafyzickým zločinům a omezením se možnost pojednání neztrácí. Otevírá se díky slovotvorbě[5] a gramatice. Lze přeci konstruovat nová slova, nekonečné věty[6] a vetkávat hrubé akty (tj. slova) zpět do jemnější textury souvislostí.

1.1 | O metodě a předmětu

Při četbě tohoto textu ale nebude třeba čelit nebezpečím, jež číhají na návštěvníky babylónské knihovny v podobě výčtu všech kombinací smysluplných i nesmyslných vět.[7] Úběžníkem teoretické práce hodné svého jména[8] je kontemplace vedená rozumem. Jejím výsledkem je obraz (nebo lépe mozaika), v němž pojedanou skutečnost nahlížíme jasně a rozlišeně (nebo lépe roztříštěnou a přesto v celku).

K výše jen lehce nastíněným obtížím přidejme ještě jednu – jde o neostrost pojmu umělecký doktorát.[9] Historie uměleckého doktorátu sahá do poválečných let ve Spojených státech, v našich regionech ale svou identitu toto studium teprve hledá.[10] Z jeho samotné povahy vyplývá interdisciplinarita. Aspirant titulu uměleckého doktorátu stojí jednak před úkolem vykázat relevanci předmětu svého výzkumu obdobnou formou jako je tomu na humanitních vysokých školách a jednak tuto relevanci prověřit uměleckou praxí. Viděno ideálně jsou teorie a praxe dvě spojité nádoby, dva aspekty konání tvůrčího individua, které se v rámci svého pragmatického bláznovství a podivínství pokouší o obojí, s rizikem, že zcela nedosáhne ani jediného. To ale neznamená, že toto obé mizí. Teorie i praxe, vědecké i umělecké může být nerozlišitelně sloučeno – jedno může být inspirováno

[4] Toto není mystická věta. Viz BUBER, Martin. *Já a Ty*. Olomouc: Votobia, 1995. 120 s. ISBN 80-7198-042-1. s. 9-10.

[5] Namátkou: modroha, mrogurt, spahnilý, štěkaná atd.

[6] PINKER, Steven. *Slova a pravidla*. Praha: Academia, 2008. 455 s. ISBN 978-80-200-1641-6. s. 21-22.

[7] BORGES, Jorge L. *Zrcadlo a maska*. Praha: Odeon, 1989. 448 s. ISBN 80-207-0076-5. s. 64-72.

[8] Z řeckého slovesa θεωρεῖν, což znamená nahlížet, kontemplovat.

[9] DOLANOVÁ, Lenka. Umělecký doktorát. *A2* [online]. 2009, č. 16 [cit. 2013-01-08]. Dostupné z: http://www.advojka.cz/archiv/2009/16/umelecky-doktorat. ISSN 1803-6635.

[10] Ibid.

a nahlíženo prizmatem druhého: vidět mechanické kmitání jako estetický akt, nahlédnout hudební kompozici jako algoritmus, nebo použít princip Occamovy břitvy[11] jako tvůrčí metodu, atd.

Zmíněná hybridnost (věda-umění) se jistě odráží i na formě a jazyku dizertace, který ze zažitého rigidního způsobu vědecké publikace bude občas vykračovat směrem k mytopoetickým způsobům promlouvání. Například prostor poznámkového a citačního aparátu si lze představit jako příležitost pro exploitaci – vpád cizorodého, nebo jako podhoubí asociací.

Pokud jsme v souvislosti s uměleckým doktorátem zmínili pojem interdisciplinarity, musíme spolu s ním zmínit i pojem redukce. Z metodologického hlediska se jeví dokonce jako nutnost, jelikož pokud by nedošlo k zúžení, kterým téma nahlížíme, záhy bychom se ztratili v síti odkazů. V případě programovacího jazyka Pure Data by tato síť byla utkána z oborů informatiky (formální jazyky), matematiky (DSP[12]), fyziky (akustika), psychoakustiky, hudební teorie, historie výpočetní techniky a hudebních nástrojů, estetiky, teorie „nových“ médií a jiných. Souvislosti a asociace, které se od Pur Dat odvíjí, jsme se pokusili znázornit pomocí schématu na str. 6.

Je zřejmé, že autor – již jednou podivně rozpolcen tím, že je aspirantem uměleckého doktorátu – pokoušející se pojednat téma vztahu programování a umělecké tvorby je ještě jednou vystaven proplouvání mezi Skyllou vyčerpávající studie (pak ale hrozí nebezpečí ztracení se a neschopnosti vidět celek; autor se k takovému úkolu navíc necítí být kompetentní) a Charybdou ulpění na povrchu či zjednodušení (tomu se autor pokouší vyhnout). Jakou cestou se vydat?

Metodologická a tématická redukce zde předkládané práce je dána autorovou uměleckou a pedagogickou praxí. Během ní sám prozkoumával různé oblasti aplikace programovacího jazyka Pure Data a hledal srozumitelnou úroveň sdělení při výuce.[13] Práce si tedy nedělá nárok na vyčerpávající průzkum, ale prezentuje okruhy, které leží v teoreticko-praktickém ohnisku zájmu autora: programování interaktivních zvukových systémů, syntetizérů, audiovizuálních aplikací, přemítání o historii techniky, postdigitálním a algoritmickém umění, live codingu, „nástrojařině“, „bastlení“, DIY[14] kultuře, open-source komunitě, příčinách, důsledcích a významu toho všeho. Pokouší se o to jazykem, který by měl

[11] Podle františkánského mnicha a filosofa Williama z Occamu (1287-1347). Jde o princip vedení myšlení, který říká, že prostředky nutné k dosažení cíle se nemají zbytečně zmnožovat. Jako takový stojí v základech exaktních věd, ale analogicky si jeho aplikaci lze představit i v umělecké praxi.

[12] Digital Signal Processing, viz s. 119.

[13] Autor přednášel avedl kurzy o Pure Datech na AVU, FAMU, FaVU.

[14] Do It Yourself (DIY) je druh kultury, pro kterou je charakteristický étos soběstačnosti, sdílení informací, dobrovolné skromnosti a trvale udržitelných modelů.

vykazovat srozumitelnost i pro člověka „z vnějšku“, pro kterého jsou zmíněné okruhy neprobádanou krajinou.

Předchozí vymezení předmětu by bylo vágní, pokud bychom k němu nepřipojili následující: domníváme se, že programovací jazyk Pure Data (spolu s dalšími nástroji[15]) je nezanedbatelnou vlnou v moři počítačového umění. Není sice příčinou revoluce, ale spíš součástí změn, resp. změny paradigmatu, která na „novo-mediální“ scéně proběhla a dále probíhá. Miller Puckette (autor Pure Dat) se domnívá, že poslední dvě dekády patřily tvůrcům systémů a budoucnost vidí v algoritmech.[16] Pure Data přinesla nové postupy (zejména v oblasti audiovizuální performance) nebo díky nim na nové úrovni ožívají již objevená témata a umělecké formy (například koncept otevřeného uměleckého díla Umberta Eca nebo přístupy umělců z okruhu hnutí Fluxus).

1.2 | O struktuře

Je lidské myšlení lineární a sukcesivní, nebo spíš připomíná rhizomatickou strukturu? Pohybuje se po liniích, nebo přeskakuje z jednoho uzlu do druhého? Odpověď na otázku leží za hranicemi této práce, ale právě v případě uvažování o rozvržení textu je vhodné ji zmínit. Ačkoliv je text organizován podle určité logiky (od jednoduššího ke složitějšímu, od známého k neznámému) a má lineární charakter, byl by autor potěšen, kdyby jej čtenář nahlížel více jako otevřenou strukturu, v níž se jednotlivé části vzájemně ovlivňují, vysvětlují a mezi jejichž pnutím vyvstávají nevyslovené významy a otázky.

Následující text je rozvržen do čtyř oddílů. Téma vztahu programování a umění se neobjevilo bez příčiny, ale má své dějinné předpoklady. Každé umělecké dílo je dítětem své doby[17] a jako takové v sobě skrytě nese jejího ducha – syntézu otisku světa do autora, jeho étosu, celku podmínek, ve kterých se tvorba odehrává. Do tohoto celku podmínek bezesporu patří i nástrojový repozitář umělce. První část ve zkratce pojednává o vývoji techniky jako o předpokladu, bez kterého by současná situace v počítačovém umění a téma této dizertace nebyly možné. Zmíněny zde jsou elektronické i mechanické výpočetní nástroje. O nástrojovém repozitáři nemusíme uvažovat úzce, ale můžeme rozšířit jeho chápání také na způsoby myšlení, organizace a produkce uměleckého díla – proto následují úvahy o algoritmickém přístupu k tvorbě.

[15] Programovací jazyky SuperCollider, Processing, programovací prostředí Fluxus a řada dalších.

[16] KRATOCHVÍL, Matěj. Budoucnost patří algoritmům. *His Voice*. 2006, roč. 6, č. 1, s. 8-9. ISSN 1213-2438.

[17] KANDINSKY, Wassily. *O duchovnosti v umění*. Praha: Triáda, 2009. 134 s. ISBN 978-80-87256-08-4. s. 11.

Druhý oddíl otevírá pojednání o genealogii programovacího jazyka Pure Data, po kterém následuje vlastní rukověť. Jde o kritickou revizi učebních materiálů, které autor během své pedagogické praxe vytvořil. Nejprve jsou zde představeny základy jazyka (slovní zásoba a gramatika), a poté následují ukázky konkrétních aplikací (sestrojení sekvenceru, syntetizéru, sampleru, videomixu, základy interaktivity atd.). Rozsah aplikací je volen tak, aby dokázal pokrýt základní oblasti „novo-mediální" tvorby (zvuk, obraz, pohyblivý obraz) nebo aby poskytnul minimální vodítko pro další samostudium.

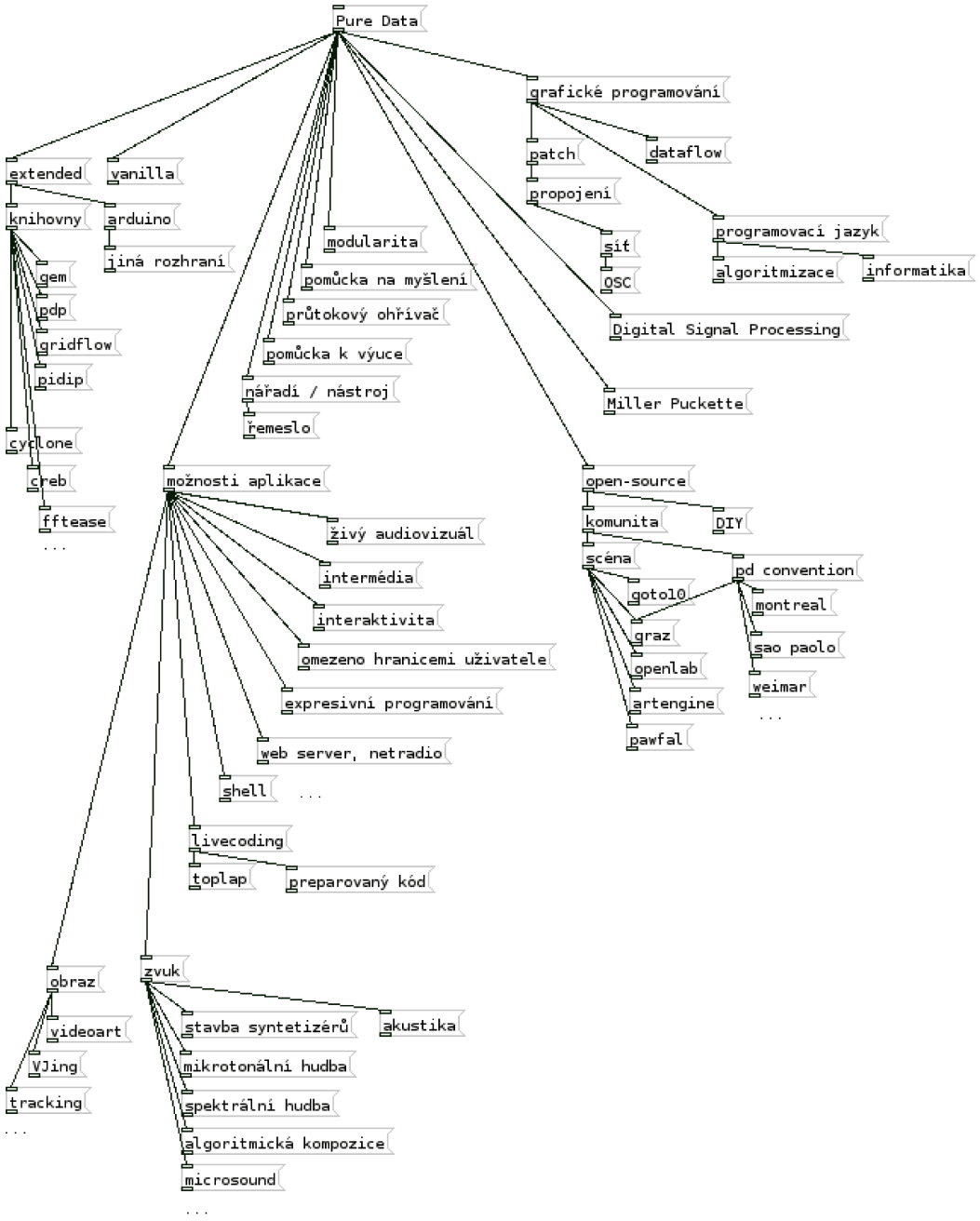
Ve třetím (praktickém) oddílu jsou prezentovány nástroje a kódy, jež vznikly v dílnách umělců, kteří Pure Data využívají jako tvůrčí nástroj. Díky srovnání přístupů různých autorů je lépe nahlédnutelná škála užití tohoto programovacího jazyka. Třetí oddíl byl pro mne zároveň příležitostí kritické sebereflexe a procházení osobního archivu kódů.

Poslední (teoretický) oddíl je věnován pokusům o reflexi a vidění Pure Dat v širších společensko-kulturních souvislostech. Čtenář zde najde kapitoly o postdigitálním umění, DIY a open-source kultuře, live codingu. Obecně jde o témata, jež se Pure Dat nějak dotýkají.

1.3 | O názvu

Hlavní titul a téma dizertace – programovací jazyk Pure Data – bude dostatečně objasněn v následujících kapitolách, nyní se krátce zmiňme pouze o podtitulu. Ten poukazuje na příručnost. Jednou z „ambicí" a cílem následujících stran je být „k ruce", být propedeutikou uvádějící do problematiky aplikace Pure Dat napříč různými oblastmi. Pure Data jsou pomyslným „švýcarákem", multiúčelovým nástrojem v rukou postdigitálního umělce. Pojem „postdigitální" bude podrobněji pojednán v poslední kapitole. Prozatím se spokojme s provizorním chápáním postdigitálního umělce jako umělce-programátora, tedy někoho, kdo raději vytváří vlastní nástroje a aplikace než aby používal nástroje druhých. Není to koncový uživatel aplikace, ale spíš někdo, kdo se chápe jazyka jako základního nástroje k vytváření nových nástrojů a ohýbání těch starých. Není vyloučeno, že pro postdigitálního umělce je již samotný nástroj právě to, o co mu jde.

1.4 | Schéma souvislostí



2 | Předpoklady

Pak jako ve snu začal něco hledat kolem sebe po zemi; nebyl by dovedl vysvětlit co, i kdyby byl nadán darem řeči. Ale měl to dobře rozpoznat v okamžiku, kdy to zahlédne. Byl to těžký špičatý kámen, asi patnáct centimetrů dlouhý, a i když se mu nehodil přesně do ruky, svému účelu vyhovoval. Mávl paží, udivil ho zvýšený důraz švihu a zalil ho příjemný pocit síly a moci.

2001: Vesmírná Odysea

Nic se neděje bez příčiny, přítomnost je spletí kauzálních řad, jejichž počátek nedohlédneme. Pokud bychom chtěli být důslední a začít rozplétat řadu příčin, jež by nás dovedla k základům vztahu techniky a umění, pak by šlo opravdu začít momentem naznačeným v epigrafu – momentem, kdy bytost rodu Homo vztáhla poprvé ruku po jiném a začala toto jiné účelně používat. Tato schopnost není vlastní pouze našemu rodu, ale najdeme ji u řady jiných tvorů z živočišné říše. Korýši, ptáci, opice nám jsou v tomto ohledu bližší, než si myslíme. Dalším důležitým předpokladem pro pochopení stávající situace je např. schopnost symbolizace (pochopení něčeho jako něčeho jiného), schopnost abstraktního a logického myšlení. Bez dispozice odhlédnutí od jednotlivostí a nahlédnutí obecné struktury a bez schopnosti s tímto obecným dále pojmově pracovat podle daných pravidel by vůbec nebylo možné myšlení a poznání.

Naším zájmem zde ale není rozvíjet antropologickou nebo epistemologickou studii, ale ve zkratce prozkoumat historické podmínky, bez kterých by se něco takového jako počítačové umění nebo programovací jazyk Pure Data patrně nikdy neobjevilo. Na obecnější rovině jde o širší vnímání vztahu vývoje vědy, techniky a umění. Každá historická epocha si vytvářela na základě těch či oněch společen-

ských faktorů svůj osobitý vztah ke světu a v souvislosti s ním i svůj umělecký výraz.[1] Věda, technika a umění se k sobě mají jako tři spojité nádoby: vzájemně se podmiňují a vedou spolu „zápas". Naivní dějiny umění by tak mohly být podány jako dějiny způsobů reprezentace, uskutečňované za pomoci určitých technik (pigmenty, grafické techniky, camera obscura, lineární perspektiva, fotografie, film, virtuální realita).[2]

Řecké slovo techné (τέχνη), od kterého je odvozeno slovo technika, v sobě nese řadu poloh, od prostého označení řemeslné dovednosti až po dovednost související s uměním. Technika se neobejde bez určitého poznání – epistémé (ἐπιστήμη).[3] Abychom *věděli* jak co máme udělat a naše jednání nebylo pouhou imitací, musíme porozumět určitým principům. Toto porozumění se ověřuje např. schopností aplikovat danou techniku v různých kontextech. Zvuk lze pochopit jako kmitavý pohyb hmoty v pevném, kapalném a plynném prostředí, který v konečné podobě vyvolává sluchový vjem.[4] Na základě porozumění tomuto principu pak může následovat dlouhosáhlý průzkum v rozkmitávání různých soustav, jak to probíhalo a probíhá v nauce o hudebních nástrojích. Nebo v rámci řešení problému reprezentace prostoru v ploše nám může jako východisko dobře posloužit pochopení základních principů lineární perspektivy.

Příkladem, na němž je patrná vzájemná provázanost ἐπιστήμη a τέχνη, vědeckého a uměleckého diskurzu, je situace na přelomu 19. a 20. století. Matematici jako Henri Poincaré a Bernhard Riemann vytvořili základ pro počet a uvažování ve vícedimenzionálních prostorech. Představa čtvrtého rozměru byla nejprve popularizována v literatuře[5] a posléze ovlivnila i výtvarnou scénu.[6] Pro avantgardu byl tento koncept prostředkem pro opuštění lineární perspektivy a vytyčení nových reprezentačních modelů (kubismus). Na konci dimensionistického manifestu (1936) od Karla Tamko Sirata, který volá po překonání euklidovské geometrie, najdeme jména jako Hans Arp, Francis Picabia a Marcel Duchamp.

---

[1] ŽEGIN, Lev Fjodorovič. *Jazyk malířského díla*. Praha: Odeon, 1980. s. 33

[2] Vilém Flusser - 1988 interview about technical revolution. In: *YouTube* [online]. 13. 01. 2011 [cit. 2014-01-22]. Dostupné z: http://www.youtube.com/watch?v=lyfOcAAcoH8. Kanál uživatele idegeimre0.

[3] Tyto pojmy najdeme již v antické filosofii, kde je jejich úzus nejednoznačný. U Xenofóna ἐπιστήμη poukazuje přímo na dovednost, praxi a τέχνη. V některých z Platónových dialogů je zřejmá jejich souvislost a vzájemná podmíněnost (Plat. Ion 532c). Tuto souvislost ostatně můžeme vnímat i na etymologii slova roz-umím. Aristoteles je rozlišuje přísněji jako řemeslo a vědění. Viz http://plato.stanford.edu/entries/episteme-techne/

[4] SYROVÝ, Václav. *Hudební akustika*. Praha: Akademie múzických umění, 2008. 440 s. ISBN 978-80-7331-127-8. s. 11.

[5] První známější popularizací těchto myšlenek byl román *Plochozemě: Romance o mnoha rozměrech napsaná Čtvercem* od Edwina Abbota.

[6] KAKU, Michio. *Hyperprostor*. Praha: Argo, 2008. 324 s. ISBN 978-80-257-0013-6. s. 69.

Jiným příkladem textu, v němž se autor prorocky, mimo jiné, odvolává též na spojitosti vědecko-technického a uměleckého diskurzu, nám může být vyznání Johna Cage:

> Věřím, že používání hluku v hudební tvorbě bude hrát čím dál větší roli a bude narůstat, dokud nedojdeme k hudbě produkované elektrickými nástroji, které nám umožní používat všechny slyšitelné zvuky. Pro tento účel se budou zkoušet fotoelektrické, filmové a mechanické prostředky.[7]

Cage své Credo přednesl v roce 1937 a k jeho výčtu prostředků by dnes bylo možné přidat i softwarové a digitální nástroje.

Jaques Aumont rozlišuje v pojmu technika tři různé roviny: 1) nástroje k výkonu dané činnosti: například různé druhy barevných pigmentů; 2) technika vlastního provedení dané činnosti: když se například při natáčení filmů začal používat zoom, běžně nahrazoval jízdu kamerou, ale užíval se také k novému efektu (přiblížení, vertigo), který už nic nenahrazoval, ale prosadil se jako nová filmová forma; 3) obecný diskurz o technice a důsledcích, které přinášejí v jednotlivých případech.[8] Je zřejmé, že do prvního bodu spadají v okruhu našich úvah i softwarové nástroje, a pokud pochopíme Aumontův druhý bod obecněji, můžeme o technice jako způsobu provedení uvažovat též např. o algoritmech nebo myšlenkových postupech, které vedou k vytvoření díla. Proto v následujícím textu budou, kromě výběru z kapitol dějin techniky, přítomné také zmínky o formalizaci a algoritmizaci tvůrčích postupů, nebo též o programování.

V prozkoumávání a zpřesňování pojmů by šlo pokračovat dál, pro naši potřebu ale postačí, když budeme mít na paměti diádu ἐπιστήμη – τέχνη (nebo triádu věda – technika – umění) a blíže nespecifikovanou představu o jejich vzájemném vztahu a podmiňování se. Patří se ještě dodat, že inspirací pro organizaci následujícího textu pro mne byla publikace *Algorithmic Composition* od Gerharda Nierhause.[9]

---

[7] CAGE, John. *Silence*. Praha: Tranzit, 2010. 275 s. ISBN 978-80-87259-07-8. s. 3.

[8] AUMONT, Jacques. *Obraz*, Praha: Akademie múzických umění, 2005. 328 s. ISBN 80-7331-045-7. s. 179.

[9] NIERHAUS, Gerhard. *Algorithmic Composition*. Wien New York: Springer, 2009. 287 s. ISBN 978-3-211-77539-6.

2.1 | Číslo a písmo

Počátek digitalizace a výpočetní techniky sahá daleko do historie[10] a abychom uzásadnili myšlenky o algoritmizaci nebo počítačovém umění, je nabíledni začít zmínkou o jejich prastarých předpokladech. Schopnost člověka vytvářet nástroje, pracovní a komunikační prostředky (v to spadá i pojem čísla a písma) je úzce spjata s vývojem myšlení a inteligence. Jednou z kognitivních schopností prvních lidí bylo právě počítání.[11] Jde o schopnost, jíž disponuje každý žák prvního stupně, který provádí primitivní algebraické operace. Při nich předpokládá, že existuje cosi obecného, co spojuje pět stolů s pěti prsty[12] na jeho ruce. S tímto obecným pak může rozehrávat různé matematické hry. Cesta k tomuto „elementárnímu“ poznání, k citu pro číslo, počet, poměr a ke kvantitativnímu vnímání světa byla ale poněkud zdlouhavá.

Snad nejstarší dochované artefakty související s počítáním jsou archeologické nálezy kostí se zářezy (vrubovky), z nichž některé jsou datovány až do doby 35 000 let před Kristem. Jednou z hypotéz je, že šlo o primitivní lunární kalendáře.[13] Je zřejmé, že zářezy poukazují na kvantitu, nikoliv však na to, co přesně označovala. Jasnější početní artefakty se objevily na území vytyčeném dnešní Sýrií a Íránem (úrodný půlměsíc), kde kolem roku 8000 před Kristem došlo k rozvoji zemědělství a obchodu. Mezi běžnými archeologickými nálezy z té doby byly objeveny také podivné předměty a schránky: hliněné kuličky, válečky, disky, atd. Podle Denis Schmandt-Besseratové[14] bylo funkcí těchto objektů umožnit směnný obchod, přičemž každý z tvarů označoval jinou komoditu: váleček označoval zvíře, kužel proso atd.

Další posun na cestě ke vzniku abstraktního pojmu čísla se odehrál v témže regionu kolem roku 3000 před Kristem. Ve vykopávkách z té doby se našly propracovanější formy hliněných objektů se značkami – ty pravděpodobně sloužily k označení různých počtů nových komodit (látky, kovové výrobky), které se objevily díky rozvoji řemesel. Dřívější podoby směnných objektů začali obchodníci po určitém počtu uzavírat do kulovitých schránek, na jejichž povrchu byl otištěn objekt, který obsahovaly.[15] Když bylo potřeba ověřit počet u kulovité schránky,

[10] PETŘÍČEK, Miroslav. Síť čili tělo bez orgánů, *Filosofický časopis*, roč. 46, 1998, číslo 1, str. 67. ISSN 0015-1831.

[11] NAUMANN, Friedrich. *Dějiny informatiky*. Praha: Academia, 2009. 422 s. ISBN 978-80-200-1730-7. s. 25.

[12] Slovo digitální je odvozeno od latinského slova *digitus*, které znamená prst.

[13] DEVLIN, Keith. *Jazyk matematiky*. Praha: Argo, 2002. 343 s. ISBN 80-7203-470-7. s. 23.

[14] Ibid.

[15] HRUŠKA, Bohuslav. *Pod babylónskou věží*. Praha: Práce, 1987. 374 s. ISBN 24-048-87. s. 71-77.

musela se rozbít a obsah pak mohl být přepočítán. Tato nepraktičnost vedla k tomu, že se do schránek objekty přestaly vůbec vkládat. Zprávou se staly pouze značky na povrchu. Tento moment by se v určitém smyslu dal pokládat za počátek vzniku hliněné tabulky určené k zápisu a též za počátek abstraktního chápání čísla, protože znaky již nekopírovaly informaci obsaženou uvnitř schránky, ale byly zprávou samy o sobě. Potřeba organizace obchodu, směny, majetkových poměrů, dluhů a pohledávek si postupně vyžádala vznik znakových soustav a byla to právě sumerská civilizace, kde došlo k posunu od fyzických početních prostředků ke psaným znakům.

Zápis číselných jednotek je staršího data než první pokusy o záznam mluveného slova. Vývoj písma bývá tradičně členěn na písma logografická (znak reprezentuje jedno slovo), sylabická (znak reprezentuje slabiku) a alfabetická (znak reprezentuje hlásku). Když se vrátíme k sumerské kultuře, tak do kategorie logografického písma by spadalo tzv. protoklínové písmo, kterým se zaznamenávaly zejména obchody. Protoklínové písmo patrně ještě nebylo fonetizováno a obsahovalo znaky pro pole, obilniny, zvířata, míry atd. Vlastní sumerské klínové písmo se objevuje kolem roku 2700 před Kristem. Jeho novum spočívalo v tom, že bylo již sylabické. Základem alfabetických písem, jež dnes známe (latinka, alfabeta), bylo písmo fénické.

Ve zkratkovitých úvahách a spekulacích ohledně počátku pojmu čísla a psaného slova by šlo pokračovat – zejména zmínkami o neokcidentálních kulturách, nebo z úplně jiného pohledu, jaký nabízí např. kognitivní psychologie. Smyslem předchozího ale bylo představení pojmu čísla a písma jako svébytných nástrojů, bez kterých by následující kulturní a intelektuální rozvoj nebyl možný. Počátek těchto nástrojů je zároveň počátkem abstraktního myšlení – objekt je reprezentován pomocí znaku a s tímto znakem je možné dále provádět operace bez přítomnosti objektu samotného. Smysl pro počet a možnost záznamu myšlenkových procesů způsobily epistemickou revoluci, kumulaci informací a odklon od mytopoetického pohledu na svět.

Bez technik, které umožňují kumulaci, přenos a sdílení informací, bychom si asi opravdu jen těžko dokázali představit tak vyspělé civilizace, jako byl Egypt a Babylón nebo antické Řecko.[16] Bez tohoto základního výdobytku lidského ducha by ostatně nebylo možné ani to, co je zde naším předmětem. Technika písma byla v počátku spravována elitami a dalo by se předpokládat, že tyto ji pozitivně doceňovaly. Překvapením je, že již v Platónově dialogu Faidón ale najdeme následující kritiku:

[16]HUSSEY, Edward. *Presokratici*. Praha: Rezek, 1997. 213 s. ISBN 80-86027-07-4. s. 22.

> Když byla řeč o písmu, pravil Theuth: „Tato znalost, králi, učiní Egypťany moudřejšími a dodá jim lepší paměti, je to totiž lék pro posílení jejich paměti a moudrosti.“ Král mu odpověděl: „Theuthe, veliký umělče, někdo je schopen zplodit umělecká díla, jiný však posoudit, jaký podíl škody nebo užitku přinesou těm, kdo jich budou užívat. Tak nyní i ty, kterýž jsi otcem písma, z náklonnosti otcovské jsi o něm řekl pravý opak toho, než jaký má význam. Kdo se mu naučí, přestanou si cvičit paměť a tím budou zapomínat, neboť spoléhajíce na písmo nebudou se rozpomínat sami od sebe zevnitř, nýbrž jen zevně, podle cizích znaků; našel jsi tedy prostředek pro upamatování, ale nikoli pro paměť. Poskytuješ svým žákům jen zdání moudrosti, ale nikoli moudrost pravou. Lidé se dozvědí o mnohých věcech bez učení a budou si myslet, že mnoho znají, ačkoli většinou nebudou znát nic. Bude s nimi těžké pořízení, neboť místo moudrosti budou mít jenom nátěr vzdělanosti.“[17]

2.2 | Na cestě k obecnému

Na procházce Puškinovým muzeem krásných umění v Moskvě můžeme pod inventárním číslem 4676 najít egyptský svitek, na němž v překladu stojí asi toto: „Máš-li dánu pyramidu bez vrcholu vysokou 6 loktů, s dolní hranou 4 lokty a horní 2 lokty; umocni 4 na druhou, dostaneš 16; zdvojnásob 4, dostaneš 8; umocni 2 na druhou, dostaneš 4. Přičti těchto 16 k těmto 8 a 4, dostaneš 28. Vypočti 1/3 ze 6, dostaneš 2. Počítej dvakrát 28. Hle, je to 56. Nalezl jsi správně.“ Podobných „receptů“ je na něm celá řada – mimo jiné jak přepočítávat chleba na pivo. Pokud bychom se vrátili k příkřejšímu rozdělení ἐπιστήμη a τέχνη, můžeme konstatovat, že kolem roku 2000 před Kristem měla egyptská a babylónská matematika blíž k τέχνη. Jsou to postupy, které odpovídají na otázku, jak provést daný výpočet, ale náznak vzorce nebo ono nahlížení obecného principu v ní ještě nenajdeme. Zeměměřiči a kněží sice rozuměli komutativním zákonům (2 + 3 = 5; 3 + 2 = 5), ale ještě je neformalizovali do obecné podoby (m + n = n + m). Nicméně již zde v předobraze vidíme to, co je charakteristické pro algoritmizaci: rozdělení složité operace na elementární kroky.

K zásadnímu posunu na cestě k formalizaci a abstrakci došlo až na poli řecké civilizace. Mezera mezi babylónskou a řeckou matematikou spočívala v rozdílu mezi nahromaděním pravidel pro řešení konkrétních problémů a krokem k abstraktním systémům[18], jaké rozpracovali Euklidés a Aristoteles.[19] Euklidovo dílo *Základy* je systematicky pojednaným matematickým traktátem, v němž se postu-

[17] PLATÓN. *Dialogy o kráse*. Praha: Odeon, 1979. 230 s. ISBN 01-049-79. s. 206-207.
[18] HUSSEY, Edward. *Presokratici*. Praha: Rezek, 1997. 213 s. ISBN 80-86027-07-4. s. 88.
[19] VOPĚNKA, Petr. *Rozpravy s geometrií*. Praha: Panorama, 1989. 519 s. ISBN 505-21-825. s. 181-347.

puje od jednoduchých definic (bod je to, co nemá části; čára je délka bez šířky), přes axiomy[20] (mezi dvěma body lze vést úsečku) až k deduktivně[21] vyvozeným tvrzením. Euklidés vytvořil systém, jenž se stal vzorem pro matematickou metodologii. Jako příklad můžeme uvést elegantní důkaz o nekonečném počtu prvočísel. Ten je konkrétní podobou tzv. důkazu sporem, kdy v počátku zacházíme s negací teze, kterou chceme dokázat, během důkazu ale dojdeme k vnitřnímu sporu, takže musí platit původní tvrzení:

> Nechť existuje jen konečně mnoho prvočísel. Označme je p_1, p_2, ..., p_n. Potom číslo $x = p_1 \cdot p_2 \cdots p_n + 1$ není dělitelné žádným z těchto prvočísel, jelikož při dělení dostaneme vždy zbytek 1. Tím pádem číslo x musí být buď prvočíslo, nebo musí být dělitelné nějakým jiným prvočíslem. To ale znamená, že množina prvočísel z počátku důkazu nebyla úplná, což je spor s předpokladem.

Na Euklidově způsobu myšlení jsou zřetelně patrné prvky algoritmizace: problém je vždy jasně definován (např. rozdělit úsečku na dva stejné díly), a poté následuje konečný výčet kroků, které vedou k jeho řešení. Ranějším a analogickým počinem k Euklidovu geometrickému dílu byla na poli logiky práce Aristotelova, který ve svém díle *Organon*[22] položil základy nauky o formách a zákonech usuzování. *Organon* je systematicky rozčleněn do několika traktátů, v nichž se postupuje od úvah nad základními pojmy a kategoriemi[23] přes nauku o soudu[24], až k sylogismu[25]. Aristoteles definuje sylogismus (úsudek) jako větu, v níž, pokud jisté věci přijmeme jako dané, něco jiného nutně vyplývá.[26] Klasickým příkladem je následující sylogismus:

1. propozice: Všichni lidé jsou smrtelní.
2. propozice: Prezident je člověk.
3. závěr: Prezident je smrtelný.

Aristoteles sám zkoumal obecné formy myšlení a nahrazoval tedy jedinečné pojmy obecnými proměnnými. Formálně zachycuje předchozí sylogismus takto:

[20] Axiom je výchozí tvrzení, které se nedokazuje.

[21] V deduktivním důkazu je tvrzení dokázáno ze stanovených předpokladů pouze na základě logických úvah a toto tvrzení je jisté.

[22] Pro zajímavost uveďme, že řecké slovo οργανον znamená nástroj. V tomto případě jde o nástroj pro správné vedení rozumu na cestě k pravdivému výroku.

[23] ARISTOTELES. *Kategorie*. Praha: Československá akademie věd, 1958. 75 s.

[24] ARISTOTELES. *O vyjadřování*. Praha: Československá akademie věd, 1959. 63 s.

[25] ARISTOTELES. *První analytiky*. Praha: Československá akademie věd, 1961. 220 s.

[26] TUGENDHAT, Ernst a WOLF, Ursula. *Logicko-sémantická propedeutika*. Praha: Rezek, 1997. 237 s. ISBN 80-86027-02-3. s. 56.

Vypovídá-li se A o každém B a B o každém C, musí se A vypovídat o každém C.[27]

Podobně jako písmo umožnilo kumulaci informací, tak logika umožnila jejich třídění – stala se základním epistemickým nástrojem pro zpřehledňování vztahů mezi pojmy – takto ji ostatně používali již první filosofové proti sofistům, kteří se v diskusích za pomoci rétorických kliček a logických obfuskací pokoušeli prosazovat své subjektivní pravdy.[28] Po Aristotelovi následují bohaté a dobrodružné dějiny tohoto oboru, z nichž má cenu zmínit se ještě o megarsko-stoické logické škole, která rozpracovala nauku o implikaci a větných spojkách *jestliže*, *pak*, *avšak* a *tedy*. Tyto spojky jsou dnes standardní součástí programovacích jazyků a slouží v nich k větvení operací.

Dále bychom mohli jmenovat postavy jako Wilhelm Leibnitz nebo George Boole, jež oba, obrazně řečeno, spojovaly úvahy nad jedničkou a nulou. Boole logiku „redukoval" na algebru – nahradil výroky hodnotami 0 nebo 1 a operace „počty", čímž otevřel cestu k počítačovému zvládnutí logiky. Jeho algebra je základem teorie konečných automatů a konstrukce počítačů.[29] Ještě před Boolem se problémem, jak převádět logická tvrzení na čistě matematická, zabýval Leibnitz. Ten vytvořil binární aritmetiku, ve které nám k reprezentaci čísel stačí pouze dva znaky. Inspirací mu v tom byla pravděpodobně čínská kniha I-ťing, v níž binární prvky najdeme v podobě tzv. hexagramů.[30] Při hledání vhodných teoretických modelů pro konstrukci počítače tyto objevy využili na počátku 20. století vědci jako Konrád Zuse nebo Alan Turing[31].

Aristoteles a jeho následovníci se zabývali problematikou třídění věcí a v důsledku toho přistupovali i k úvahám o klasifikaci jim odpovídajících pojmů. Počátky antické „vědy" bychom mohli vidět právě jako období posedlé klasifikací. Třídění živočichů nebo rostlin vedlo k problematice rodu a druhu, k úvahám o obecném a jedinečném i ke zkoumání obecných společných vlastností, podle nichž by bylo možné předměty kategorizovat.[32] Bylo tak založeno něco, co bychom mohli nazvat jako obousečné okcidentální stigma: na jedné straně rozvoj věd, na straně druhé fragmentarizace a mechanizace vědění, ztráta smyslu pro celek a diktát logocentrismu.

---

[27]ARISTOTELES. *První analytiky*. Praha: Československá akademie věd, 1961. 220 s. s. 30.

[28]SOUSEDÍK, Prokop. *Logika pro studenty humanitních oborů*. Praha: Vyšehrad, 2001. 224 s. ISBN 80-7021-509-7. s. 82-87.

[29]MAREŠ, Milan. *Slova, která se hodí*. Praha: Academia, 2006. 348 s. ISBN 80-200-1445-5. s. 69.

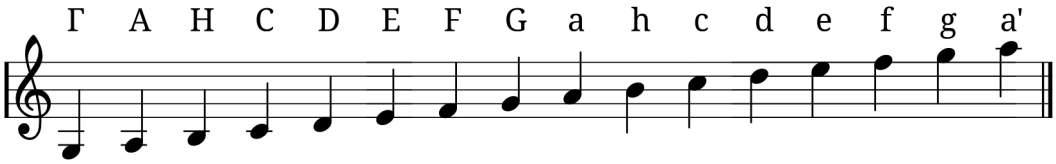
[30]Hexagram je sestava šesti plných nebo přerušených čar, které dohromady dávají 64 možných kombinací. Viz http://en.wikipedia.org/wiki/I_Ching.

[31]LEAVITT, David. *Muž, který věděl příliš mnoho*. Praha: Argo, 2007. 270 s. ISBN 978-80-7363-086-7. s. 63.

[32]ARISTOTELES. *Kategorie*. Praha: Československá akademie věd, 1958. 75 s. s.7.

2.3 | Metoda a algoritmus

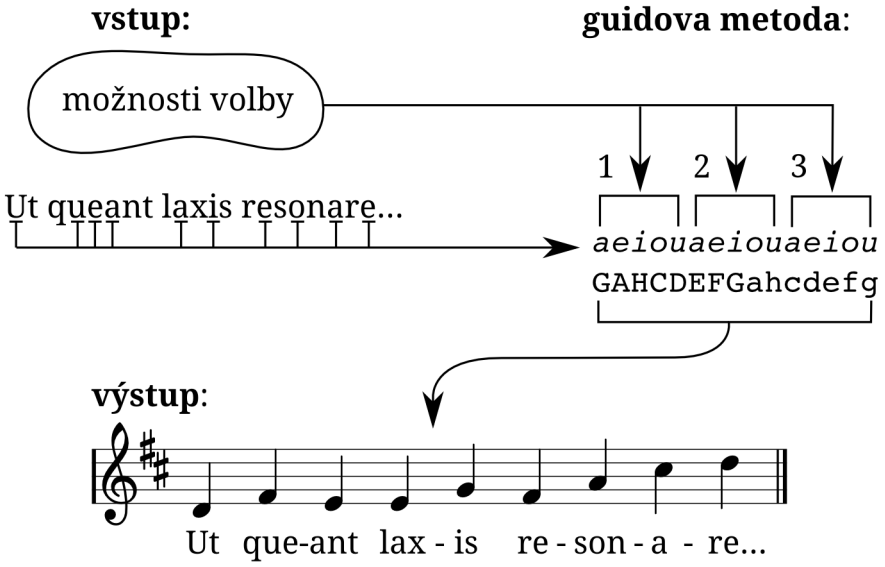
Někdy kolem roku 1026 publikoval benediktýnský mnich a hudební teoretik Guido z Arezza traktát s názvem *Micrologus*, který bývá považován za první pokus o popis kompoziční metody za pomoci konečné množiny pravidel.[33] Podívejme se na tuto metodu blíže. Guido nejprve sestrojil korespondenční tabulku mezi notami a samohláskami:



Každé tónové výšce v rozsahu dvou oktáv pak postupně přiřadil opakující se samohlásky, takže dostal následující řady:

noty:	Γ	A	H	C	D	E	F	G	a	h	c	d	e	f	g	a'
samohlásky:	*a*	*e*	*i*	*o*	*u*	*a*	*e*	*i*	*o*	*u*	*a*	*e*	*i*	*o*	*u*	*a*

Dalším krokem byl výběr latinského textu, vyabstrahování samohlásek z každého slova a jejich asociace s notami. Jelikož pro každou samohlásku jsou možné tři asociace (čtyři asociace v případě samohlásky *a*), existuje i více možných melodií pro jeden text. Pokud by obsahoval pouze jednu samohlásku, pak by výstupem byly tři možné „melodie“ o jednom tónu; v případě dvou samohlásek je to 3^2 melodií o dvou notách; v případě n samohlásek by to bylo 3^n melodií. Vidíme tedy, že počet možných kompozicí roste s délkou textu. Schematicky by Guidovu metodu bylo možné zachytit asi takto:



[33] LOY, Gareth. *Musimathics*. Cambridge: MIT, 2006. 482 s. ISBN 978-0-262-12282-5. s. 285-288.

Vstupy jsou zde dva: jednak vyabstrahované samohlásky a jednak subjektivní asociace not na samohlásky. I když jsou možnosti výběru a asociace usměrněny metodou, není tato sama zárukou vzniku použitelné kompozice. Guido sám k tomu poznamenává, že pouze výběrem těch nejlepších možností z několika pokusů povede k uspokojivému výsledku, v němž se text a melodie setká. Z pozdějších předkomputačních příkladů metod vytváření kompozice bychom měli jmenovat ještě tzv. *Würfelspiel*[34], kde byla subjektivní stránka výběru nahrazena hodem kostky. Hudební díla vzniklá hrou v kostky najdeme u Wolfganga A. Mozarta, Josepha Haydna, nebo C. P. E. Bacha. Variací na Guida i *Würfelspiel* je ve 20. století tzv. aleatorická hudba, kde je pomyslný hod kostkou místem v kompozici, které do ní vnáší prvek náhody a poskytuje interpretovi možnost dotvoření partitury.

Pokud pomineme příklady z dějin výtvarné kultury, jako je problém vydláždění daného prostoru geometrickým vzorem[35] nebo aplikace zlatého řezu ve výtvarném umění[36], pak se nabízí otázka, proč k užití formálních postupů docházelo převážně v souvislosti s hudební kompozicí. Spojitost hudby a matematiky sahá až k pythagorejské tradici a úvahám o harmonických poměrech čísel vyjádřených tóny. Pythagorejci si vystačili s číselnou řadou 1, 2, 3, 4 – resp. s poměry 2:1, 3:2, 4:3, které vyjadřují základní konsonantní souzvuky: oktávu, kvintu a kvartu. Zdá se, že hudba a její podstatné složky (tóny a rytmus) má k formalizaci blíž než výtvarné artefakty. Jedním z argumentů pro předchozí tvrzení snad může být fakt, že tón (a jeho atributy jako frekvence a délka; témbr zde pomíjíme) a melodie jsou snadněji formalizovatelné než např. křivka a obraz. Jiný důvod bychom mohli vidět v „dualistickém dědictví“ antického Řecka, kde byla múzická umění (hudba, tanec a zpívaná poezie) tradičně nadřazována konstruktivním uměním (architektura, sochařství, malířství) pro jedinečnost a neopakovatelnost performativního aktu.[37]

Dnes je situace samozřejmě jiná a od 60. let 20. století je vypozorovatelný vzrůstající zájem o formalizaci a algoritmizaci díla i na výtvarné scéně – jako příklady lze jmenovat Zdeňka Sýkoru nebo Miroslava Klivara, kteří patřili mezi první průkopníky počítačově generovaného výtvarného díla.[38] Jiným příkladem užívání

[34]On-line verzi je možné vyzkoušet na *Sunsite.univie.ac.at* [online]. [cit. 2014-01-12]. Dostupné z: http://sunsite.univie.ac.at/Mozart/dice/.

[35]AS-SAID, Issam a PARMAN, Ayşe. *Geometrická koncepce v islámském umění*. Praha: Argo, 2008. 166 s. ISBN 978-80-7203-911-1.

[36]LIVIO, Mario. *Zlatý řez*. Praha: Argo, 2006. 255 s. ISBN 80-7203-808-7.

[37]CSERES, Jozef a MURIN, Michal. *Od analógového k digitálnemu...* Banská Bystrica: Fakulta výtvarných umení, 2010. 219 s. ISBN 978-80-89078-78-3. s. 8.

[38]ŠPERKA, Martin. The Origins of Computer Graphics in the Czech and Slovak Republics. *Leonardo*. 1994. č. 27 [cit. 2013-08-08]. Dostupné z: http://www.jstor.org/stable/1575949. ISSN 0024-094X.

specifických metod a algoritmů ze současnosti nám může být literární skupina Oulipo, nebo výslovně zachycené návody na performance, jak je najdeme v jedné z „kuchařek“ hnutí Fluxus:[39]

> Láhev s vodou
> Naplň sklenici vodou z láhve
> Nalij vodu ze sklenice zpět do láhve
> Naplň sklenici a znovu opakuj předchozí proceduru
> Opakuj dokud voda není úplně rozlitá

Metoda je to, co nám umožňuje řešit výpočty, pěstovat plodiny, ale i napsat hudební kompozici – zkrátka nevylučuje žádnou z oblastí lidského konání, ať už jde o řemeslo, umění nebo vědu. Zabývat se metodami vzniku hudební kompozice nebo výtvarného díla nám může poskytnout aparát ke komparaci témat, motivů a způsobů řešení napříč různými oblastmi i časy v dějinách umění.[40] Metoda je důležitý mem lidské kultury, který nese informaci umožňující společnostem uchovávat, zdokonalovat a předávat funkční modely.[41] Abychom ale docenili význam metody a zároveň se přiblížili našemu tématu, bude zapotřebí pojednat přesněji o pojmu algoritmu, jemuž lze rozumět jako vysoce kvalifikovanému typu metodologie.[42]

Slovo algoritmus je odvozeno od perského spisovatele a matematika Abū ʿAbd Allāha Muḥammada ibn Mūsā al-Chwārizmīho, jehož jméno bylo ve 12. století polatinštěno jako Algorismus. Předběžné vymezení pojmu algoritmu najdeme u Donalda Knutha v této podobě:[43]

> Moderní význam slova algoritmus je hodně podobný výrazům jako *recept*, *proces*, *metoda*, *technika*, *procedura*, *postup*, *bláboly*, jen slovo „algoritmus“ znamená přece jen trošku něco jiného. Algoritmus je nejen konečnou množinou pravidel, která popisují posloupnost operací pro řešení jistého typu problému, ale zároveň musí splňovat pět důležitých vlastností.

[39] *Deluxxe.com* [online]. [cit. 2014-03-02]. Dostupné z: http://www.deluxxe.com/beat/fluxusworkbook.pdf.

[40] LOY, Gareth. *Musimathics*. Cambridge: MIT, 2006. 482 s. ISBN 978-0-262-12282-5. s. 285.

[41] PEREGRIN, Jaroslav. *Člověk a pravidla*. Praha: Dokořán, 2011. 166 s. ISBN 978-80-7363-347-9. s. 125-132.

[42] LOY, Gareth. *Musimathics*. Cambridge: MIT, 2006. 482 s. ISBN 978-0-262-12282-5. s. 288.

[43] KNUTH, Donald. *Umění programování, 1. díl*. Brno: Computer Press, 2008. 648 s. ISBN 978-80-251-2025-5. s. 4-6

Mezi tyto vlastnosti dále řadí: 1) *konečnost* – algoritmus musí vždy po konečném počtu kroků skončit; 2) *určitost* – každý krok algoritmu musí být přesně definován a pro každý případ v něm musí být s určitostí a jednoznačností popsány prováděné operace; 3) *vstup* – každý algoritmus má vstupy, které do něj zadáváme před jeho zahájením, nebo se dynamicky načítají za jeho běhu; 4) *výstup* – algoritmus má jeden nebo více výstupů, které mají zadaný vztah ke vstupům; 5) *efektivita* – všechny operace algoritmu musí být v rozumné míře jednoduché.

Programy, a to i ty, které generují algoritmické kompozice, mohou být vytvářeny celou řadou způsobů, tradičně se ale dělí na dva. Tím prvním je „top-down" metoda, kdy postup řešení „rozdrolujeme" na základní kroky algoritmu. Opačný přístup „bottom-up" je zase případem, kdy program vzniká „odspoda". Nejprve se formulují dílčí kroky algoritmu, které pak vedou k řešení daného problému. Určitou analogii bychom mohli vidět v tom, jak vzniká hudební kompozice.[44] Přístup „odspoda" např. může připomínat pokusy nebo volné improvizace a následné zobecnění nasbíraného materiálu v kompozici.

Mezi konkrétní přístupy užívané nejčastěji v algoritmické kompozici lze jmenovat následující: Markovovy řetězce, generativní gramatiky, neuronové sítě, teorie chaosu a soběpodobnost, genetické algoritmy, celulární automaty atd. Pro detailnější studium této problematiky lze doporučit již zmíněnou knihu *Algorithmic Composition*[45].

Pokud bychom se nyní vrátili ke Guidovi z Arezza a poměřili jeho metodu Knuthovým vymezením algoritmu, pak uvidíme, že v případě *určitosti* a *výstupu* (pokud pochopíme vztah mezi vstupem a výstupem deterministicky) neobstojí. U Guidovy metody není pro každý krok jednoznačně popsána prováděná operace, ale je zde ponechána jistá míra svobody a nedeterminovanosti. Gareth Loy ji proto vymezuje jako nedeterministickou metodologii, nebo stručněji – jako umění. Je to poněkud vágní a zároveň provokativní tvrzení. Při troše odvahy snad ale tímto prizmatem můžeme nahlédnout některé z kapitol dějin umění. Například impresionisti, autoři dodekafonické hudby, surrealističtí básníci a malíři atd. byli „vázáni" určitou metodou, jejímž výsledkem byla různá stejnost. Na druhou stranu Gareth Loy nemá docela pravdu, protože v rámci algoritmického umění najdeme i tvrdě deterministické kompozice, které Knuthovy požadavky bezezbytku splňují – jde např. o kompozici *Abakus* a některé z kompozic cyklu *Flex* Milana Guštara, nebo díla jako *Walz of Determinism* od Kryštofa Peška.

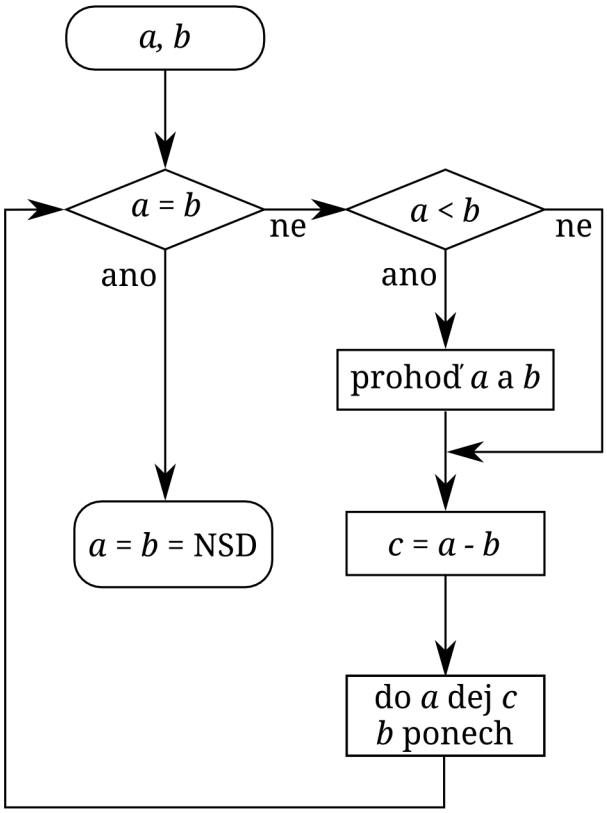
V závěru této kapitoly se musíme zmínit ještě o způsobu vizualizace algoritmů

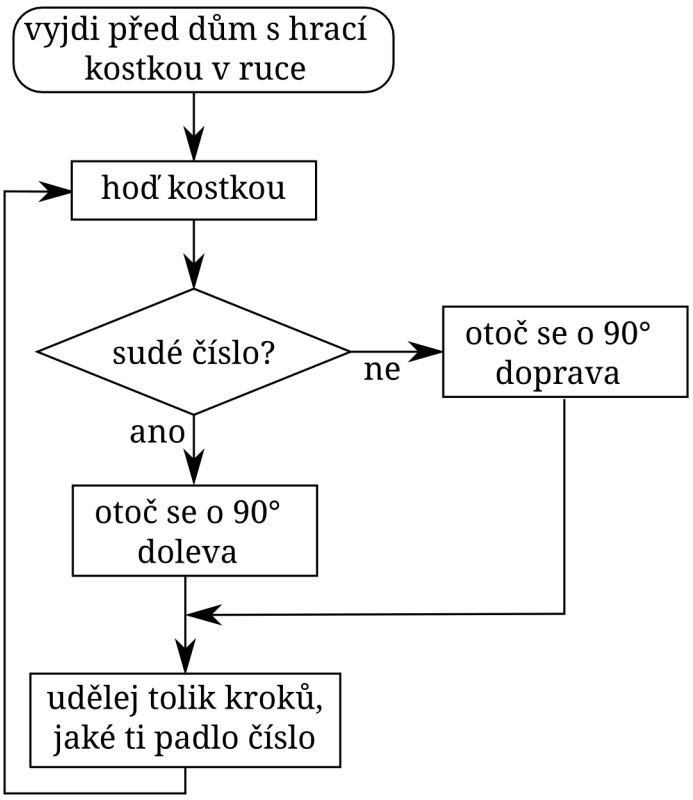
[44]COPE, David. *Techniques of the Contemporary Composer*. New York: Schrimer Books, 1997. 250s. ISBN 0-02-864737-8. s. 192.

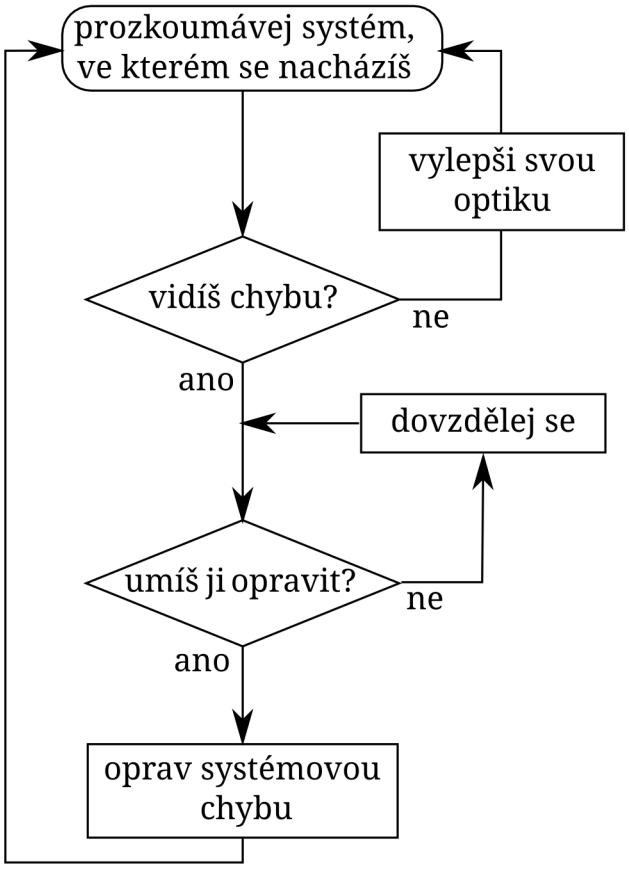
[45]NIERHAUS, Gerhard. *Algorithmic Composition*. Wien New York: Springer, 2009. 287 s. ISBN 978-3-211-77539-6.

za pomoci vývojových diagramů, jak je představil ve dvacátých letech 20. století Frank Gilberth. Vývojový diagram je totiž srozumitelnou reprezentací jednotlivých kroků a podmínek algoritmu a jako takový tvoří základní stavební kámen vizuálních programovacích prostředí, mezi která spadají i Pure Data. Tělo diagramu se sestává z následujících částí: spojnice (úsečka zakončená šipkou), která naznačuje posloupnost operací v algoritmu; obdélník s popisem, jenž definuje dílčí krok ve zpracování algoritmu; kosočtverec naznačuje větvení v algoritmu na základě splnění nebo nesplnění dané podmínky; obdélník se zaoblenými rohy označuje začátek a konec algoritmu.

Prakticky si představíme tři ukázky: první je příklad z úsvitu dějin algoritmů – jmenovitě jde o Euklidův algoritmus pro hledání největšího společného dělitele dvou přirozených čísel. Na něm je, kromě Knuthem jmenovaných vlastností, dobře vidět i jiné: elegantní jednoduchost a určitý šarm v provedení. Tyto atributy nás mohou vést při úvahách nad vyhodnocováním kvality jednotlivých algoritmů. Druhá a třetí ukázka jsou„měkčí“ vývojové diagramy popisující performance bezúčelné procházky a hacktivistický postoj. O jejich funkčnosti a platnosti nechť se čtenáři přesvědčí na základě své vlastní praxe. Též je možné vstoupit do hry a pokusit se doplnit nebo rozšířit algoritmy podle návrhů vedle nich uvedených. A nebo ještě lépe: zkusit si vytvořit své vlastní a lepší algoritmy.



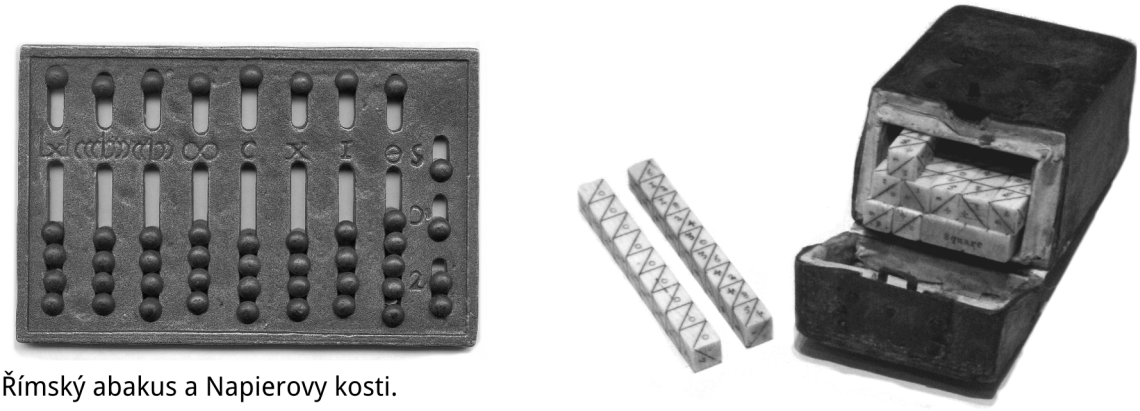

vyjdi před dům s hrací kostkou v ruce
hoď kostkou
sudé číslo?
ne
otoč se o 90° doprava
ano
otoč se o 90° doleva
udělej tolik kroků, jaké ti padlo číslo


prozkoumávej systém, ve kterém se nacházíš
vylepši svou optiku
vidíš chybu?
ne
ano
dovzdělej se
umíš ji opravit?
ne
ano
oprav systémovou chybu

2.4 | Mechanizace, neboli přimknutí k hmotě

V předchozích kapitolách jsme se zabývali myšlenkovými koncepty, jež jsou základem pro výpočetní techniku a programování. V této a následující kapitole se seznámíme s tím, jak tyto neviditelné koncepty postupně dostaly tělo. Algoritmy v sobě mají určitou performativnost[46] – jsou to mechanické návody vybízející k provádění a to bylo uskutečňováno již od dávných dob s pomocí nástrojů, které od přirozenosti líným a chybujícím lidem pomáhaly. Vývoj a zdokonalování těchto nástrojů vedlo ke vzniku osobního počítače, tak jak ho známe dnes.

Hovořili jsme již o vrubovkách, ale pokud bychom měli začít poněkud sofistikovanějšími nástroji, pak se musíme zmínit o abaku – „počtářské pomůcce“, jejíž kořeny sahají až do roku 2500 před Kristem do Akkadské říše. Odtud se abakus rozšířil po předním východě, do Řecka a Říma a do středověké Číny, kde se nazýval suan-pan.[47] Jeho varianty najdeme i v Japonsku (soroban) a Rusku (sčot). Elementární početní myšlenkové procesy jsou na nich nahrazeny mechanickým posunem daného počtu žetonů[48] a výsledek je na nich v podstatě bezprostředně viditelný. Jiná mechanická pomůcka pro počítání byla představena roku 1617 ve spisu *Rabdologia* Johnem Napierem – šlo o tzv. Napierovy kosti, se kterými bylo možné, kromě základních aritmetických operací, provádět také odmocňování.[49]



Římský abakus a Napierovy kosti.

Díky rozvoji jemné mechaniky a hodinářství přišly ke slovu také jemnější počítací mechanismy. Návrhy počítacího stroje najdeme již mezi řadou dalších vyná-

[46] V minulosti se dokonce konaly veřejné počtářské soutěže, na kterých jednotliví účastníci poměřovali své schopnosti a techniky. Dnešním slovníkem řečeno šlo vlastně o počtářskou performanci.

[47] MAREŠ, Milan. *Slova, která se hodí*. Praha: Academia, 2006. 348 s. ISBN 80-200-1445-5. s. 244.

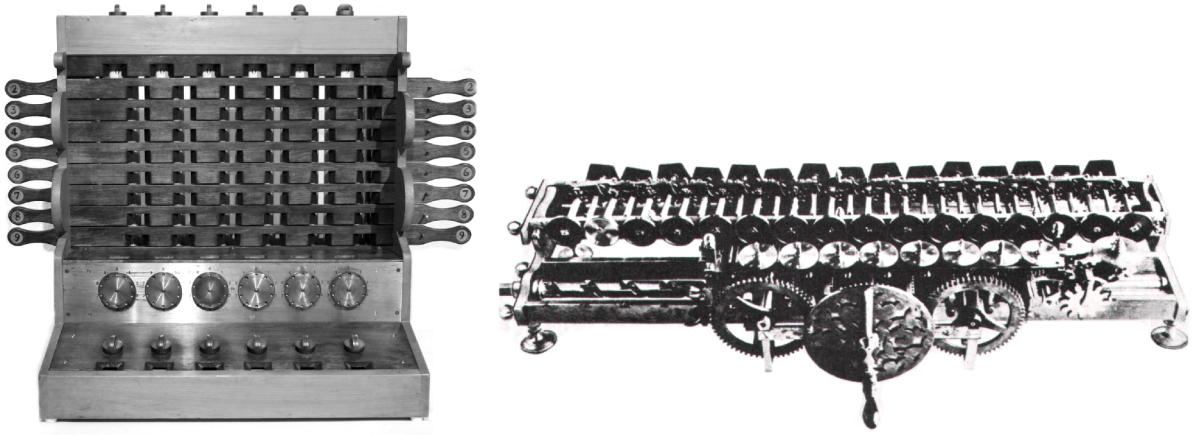
[48] Jak se na abaku počítalo můžeme najít na *Wikihow.com* [online]. [cit. 2014-01-14]. Dostupné z: http://www.wikihow.com/Use-an-Abacus.

[49] Opět se zde nebudeme detailně věnovat popisu funkčnosti. Pro zvědavého čtenáře dostupné například na *Wikipedia.org* [online]. [cit. 2014-01-14]. Dostupné z: http://en.wikipedia.org/wiki/Napier's_bones.

lezů u Leonarda da Vinci.[50] První navržený a též zkonstruovaný počítací stroj ale sestavil až Wilhelm Schickard v roce 1623. V dopise Johannu Keplerovi se o něm zmiňuje takto:[51]

> Totéž, co jsi provedl počtářskou cestou, jsem se nedávno pokusil provést mechanicky a postavil jsem z jedenácti celých a šesti neúplných koleček přístroj, který automaticky počítá zadaná čísla: sčítá, odečítá, násobí i dělí. Pořádně by ses zasmál, kdybys tu byl a viděl, jak se sama zvyšují místa po levici, kdykoli se přechází desítka nebo stovka, nebo když se jim něco při odečítání ubírá.

Poté následovaly mechanické početní stroje od Blaise Pascala (tzv. Pascalina) a Gottfrieda W. Leibnizte, který byl s Pascalovou prací obeznámen. Navíc ale přišel s inovací v podobě kola se zuby proměnné délky. Jeho řešení se pak udrželo až do dneška a použil je i autor jedné z posledních mechanických kalkulaček Curt Herzstark (Curta).



Schickardova a Leibnitzova mechanická „kalkulačka".

Předchozí zmíněné strojky byly v podstatě jednoúčelové a nenajdeme v nich představu programovatelnosti a oddělení kódu od mechanismu (software / hardware). Naopak kód je zde „vtělen" do nastavení a propojení jednotlivých ozubených kol. Mechanický předobraz programovatelného počítače najdeme až u Charlese Babbage (1791–1871) a jeho analytického stroje, který operoval v desítkové soustavě. Ideu programovatelnosti za pomoci děrných štítků si Babbage vypůjčil z Jacquardova tkalcovského stavu.

Na rozdíl od spřádání lístků a květů ale analytický stroj měl spřádat vzory algebraické. Jeho architektura v sobě zahrnovala představu procesoru, paměti

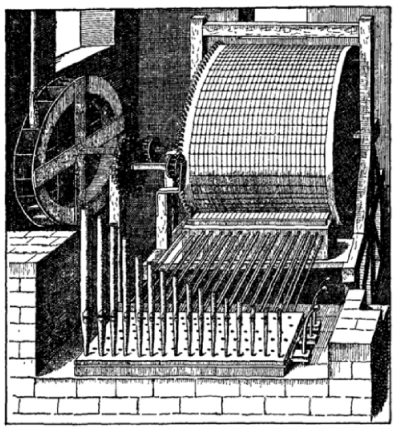
[50]NAUMANN, Friedrich. *Dějiny informatiky*. Praha: Academia, 2009. 422 s. ISBN 978-80-200-1730-7. s. 63.
[51]Ibid. str. 65.

k uchování mezivýsledků i vstupní a výstupní zařízení. Dále měl umožňovat větvení a skoky v prováděném programu.[52]

I když původní motivy k vybudování analytického stroje (a též jeho předchůdce, tzv. diferenčního stroje) byly pragmatické – měl řešit především výpočty logaritmických tabulek a jiné úmorné početní úkoly – je zřejmé, že si Babbage a jeho blízká spolupracovnice Ada Lovelace uvědomovali univerzálnější možnosti jeho použití:[53]

> Analytický stroj může provádět operace také s jinými entitami, než jsou čísla. Je to možné v případech, kdy se mezi entitami nacházejí vzájemně zaměnitelné základní vztahy, které jsou převeditelné do operační notace (algoritmus), s nimiž umí analytický stroj pracovat. Představme si tak např. základní vztahy tónů v nauce o harmonii a hudební kompozici – ty je možné vnímat přesně ve výše zmíněném smyslu. Analytický stroj by tak mohl skládat hudební kompozice libovolného stupně složitosti a rozsahu.

Od této citace vede cesta k úvahám o vývoji samočinných a programovatelných hudebních nástrojů. Již v 9. století perští bratři Banū Mūsā v knize o důmyslných vynálezech publikovali návrh na vodou poháněné varhany (původně mnohem starší nástroj, jehož historie sahá až do 3. století před Kristem), které disponovaly vyměnitelnými kotouči s různými melodiemi. Dalším z jejich vynálezů byla automatická a programovatelná flétna. Jejich vynálezy jsou předobrazem pozdějších samohrajek a dnešních programovatelných sekvencerů[54] – tj. přístrojů, jež umí v čase provádět určité události (např. tón). Jejich konstrukcí se budeme podrobněji zabývat později.



„Notový zápis“ na otočném bubínku – předobraz dnešních sekvencerů.

[52] Ibid. str. 101.

[53] LOY, Gareth. *Musimathics*. Cambridge: MIT, 2006. 482 s. ISBN 978-0-262-12282-5. s. 285.

[54] Vyčerpávající studií na toto téma je publikace GUŠTAR, Milan. *Elektrofony I*. Praha: Uvnitř, 2007. 397 s. ISBN 978-80-239-8446-0.

Charlesi Babbageovi se jeho velkorysé plány kvůli finančním a technickým komplikacím nikdy nepodařilo realizovat. Druhá verze diferenčního stroje byla sestavena až v roce 1991 při příležitosti dvoustého výročí narození jeho vynálezce.[55] Z analytického stroje existují pouze některé součásti (aritmetická jednotka, tiskárna), ale jako celek nebyl do dnešního dne sestrojen. Vedou se spory o to, zda byl Alan Turing, autor teoretického modelu obecného výpočetního stroje, obeznámen s Babbageovou prací. Ať už je to s odpovědí jakkoliv, je zřejmé, že Babbageův analytický stroj předjímá Turingův univerzální stroj a podobné konstrukční prvky pak najdeme v návrzích konstruktérů prvních počítačů, kterými byli Konrád Zuse a Howard H. Aiken.[56]

2.5 | Elektrifikace

Vize světa, jakou ve svých dílech nabízí autoři steampunkového žánru a ve kterém hlavní energii pro pohon strojů zajišťuje pára, by byla jistě možná, pokud by ale neprobíhal výzkum nad oním zvláštním jevem, vyskytujícím se po tření ebonitové tyče liščím ohonem. Vpád elektřiny a následná elektrifikace struktur a systémů vytvořených lidmi znamenala nejenom významný technický zlom, ale i změnu našich kognitivních struktur. Vynález baterie (přenosného zdroje energie), žárovky, elektromotoru, telegrafu a řady dalších zařízení ovlivnil naše biorytmy, chápání prostoru i politické stability.

> Co může více posílit mír, než stálý a neomezený diskurz mezi všemi národy a lidmi světa? První olivovou ratolestí, kterou nás věda obdařila, byla parní síla. Pak přišla ještě slibnější ratolest – tento úžasný elektrický telegraf, díky kterému může každý, kdo se nachází v dosahu kabelu, ihned kontaktovat své bližní, ať se nacházejí kdekoli na světě.[57]

Je asi přirozené, že na úsvitu té které techniky se o ní lidé vyjadřují s utopickým zápalem, jako je tomu ve výše uvedené citaci. Procesu elektrifikace se nevyhnula lidská komunikace ani způsob zaznamenání vizuálních a zvukových dat. Podle Geoffreye Batchena toto vše probíhalo zcela v duchu modernity a tedy i v duchu kapitalismu, industrializace a kolonialismu. Technické vymoženosti, dle Batchena, spojuje touha po automatizaci aktu zobrazování a následné odsunutí lidského

[55] Babbage's Difference Engine No. 2. In: *YouTube*[online]. 2. 05. 2008. [cit. 2014-01-22]. Dostupné z: http://www.youtube.com/watch?v=0anlyVGeWOI. Kanál uživatele WIRED.

[56] NIERHAUS, Gerhard. *Algorithmic Composition*. Wien New York: Springer, 2009. 287 s. ISBN 978-3-211-77539-6. str. 43.

[57] NAUMANN, Friedrich. *Dějiny informatiky*. Praha: Academia, 2009. 422 s. ISBN 978-80-200-1730-7. s. 167.

těla z aktivní do pasivní role.[58] Jen na okraj zde zmiňme, že právě kvůli tomuto „původu“ Batchen považuje většinu „nových médií“ za stejně stará, jako je stará modernita.[59]

Martin Heidegger proti oslavě techniky staví toto stále aktuální a střízlivější zhodnocení:

> Vzdálenosti v čase a prostoru se neustále zmenšují. Kam se dříve cestovalo týdny a měsíce, dostaneme se nyní letadlem přes noc. Co jsme se dříve dozvěděli až po letech, nebo vůbec ne, dovídáme se nyní v rozhlase hodinu od hodiny – ihned. … Toto kvapné odstraňování veškerých vzdáleností však nevytváří blízkost; neboť nepatrná míra vzdálenosti není ještě blízkostí. Co nám film a rozhlas přiblížily na dosah ruky, může nám přesto zůstat vzdálené. Co leží v nedohlednu, může nám být blízké. Malá vzdálenost není ještě blízkost. Velká vzdálenost není ještě dálka.[60]

K Heideggerově velmi obecnému popisu problému vztahu vzdálenosti a blízkosti se ještě vrátíme v následujících kapitolách jako k jednomu z důležitých antropologických momentů a motivů pro postdigitální umělce. Prozatím ale zůstaneme u námi sledované linie myšlení, tj. u zjednodušeného popisu vývoje techniky. Pokud bychom měli proces elektrifikace vztáhnout k předchozím kapitolám a pokračovat v budování mostu, jenž vede k vynálezu osobního počítače a programovacím jazykům, nemůžeme nezmínit žongléra, konstruktéra nejneužitečnějšího stroje (podle původní ideje Marvina Minského), autora pojmu *bit* a prvního šachového algoritmu, hackera rulety a zásadní postavu v dějinách teorie informace, jakou byl Claude Shannon.

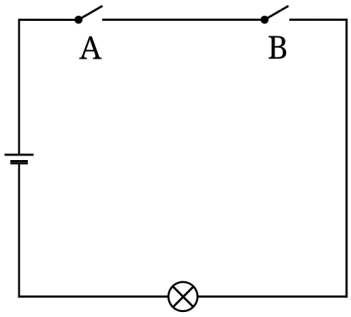
Jeden ze základů výpočetní techniky položil Shannon již ve své magisterské práci z roku 1937 s názvem *A Symbolic Analysis of Relay and Switching Circuits*. V ní se, mimo jiné, zabýval aplikací booleovské algebry na elektrické obvody. Booleova algebra používá logické operátory jako AND (konjunkce, symbolický zápis: $\wedge$), OR (disjunkce, symbolický zápis: $\vee$) a NOT (negace, symbolický zápis: $\neg$) a dvě binární hodnoty 1 a 0. Shannon poukázal na souvztažnost mezi elektrickými obvody a logickými operacemi – výsledek logické operace nebo výpočtu může být reprezentován elektrickým okruhem a vice versa. Pro ilustraci se podívejme na následující schémata a formule.

---

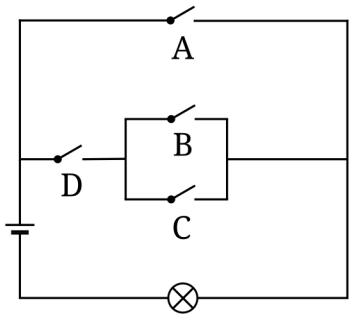
[58] DVOŘÁK, Tomáš (ed.). *Kapitoly z dějin a teorie médií*. Praha: Akademie výtvarných umění v Praze, 2010. 349 s. ISBN 978-80-87108-16-1. s. 231.

[59] Ibid. s. 232.

[60] HEIDEGGER, Martin. *Básnicky bydlí člověk*. Praha: OIKOYMENH, 1993. 187 s. ISBN 80-85241-40-4. s. 7.



A ∧ B



A ∨ (D ∧ (B ∨ C))

První schéma se dá „číst“ takto: žárovka se rozsvítí pouze tehdy, pokud je vypínač A *a současně* vypínač B sepnutý. Druhé pak takto: žárovka se rozsvítí, pokud je sepnutý vypínač A nebo vypínač D *a současně* vypínač B nebo C. První schéma je příkladem logické konjunkce (výrok je pravdivý, pokud jsou věty A i B pravdivé), druhé pak kombinací konjunkce a disjunkce (výrok je pravdivý, pokud je pravdivá věta A, nebo věta D a současně věta B nebo C).

Z prostého Shannonova postřehu, že dva stavy relé mohou kódovat jedničku a nulu, pravdu a nepravdu, se následně odvíjela problematika logických sítí, teorie konečných automatů a digitálních počítačů.[61] Shannon by patrně nesouhlasil s tím, když bychom o jím objevené souvztažnosti hovořili jako o elektrifikaci myšlení, jeho pohled na věc by byl patrně opačný – je to přece myšlení, které vstupuje do prostoru elektrických obvodů. V rámci „měkkého“ myšlení si ale můžeme dovolit tuto perspektivu obrátit a pochopit elektřinu jako druh energie, jež vstupuje do prostoru myšlení a přivádí je k novému pohledu na sebe sama. Před elektřinou to bylo světlo.

Během 30. a 40. let 20. století, kdy synchronně na jedné straně v USA pracoval Howard Aiken a na druhé Konrad Zuse v Německu (a do určité míry bychom sem mohli zařadit i dění okolo kryptoanalytického centra v anglickém Bletchley[62]), vznikly první prototypy počítačů. Zuseho Z1 z roku 1938 bývá považován za první plně automatický, programově řízený a programovatelný počítač.[63] Právě na vývoji Zuseho počítačů (Z1 až Z4) je dobře viditelný proces elektrifikace. První model Z1 byl sestaven jen z mechanických částí, Z2 a další modely pak již obsahovaly elektrická relé. Povaha konstrukčních prvků má přitom zásadní vliv na rychlost výpočtu – mechanické relé je pomalejší než elektromechanické.

[61]MAREŠ, Milan. *Slova, která se hodí*. Praha: Academia, 2006. 348 s. ISBN 80-200-1445-5. s. 291.

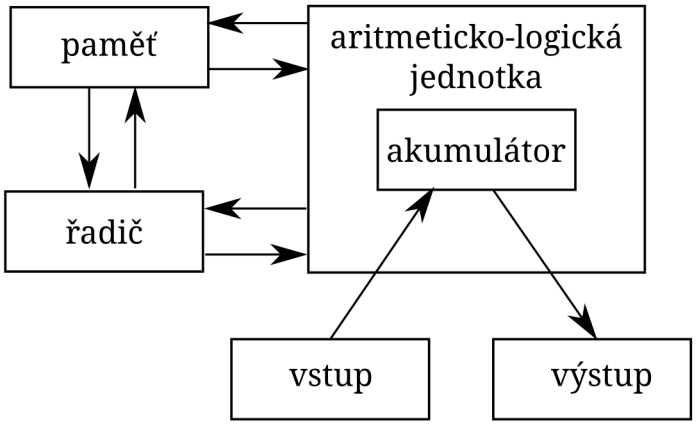
[62]LEAVITT, David. *Muž, který věděl příliš mnoho*. Praha: Argo, 2007. 270 s. ISBN 978-80-7363-086-7. s. 152-160.

[63]NAUMANN, Friedrich. *Dějiny informatiky*. Praha: Academia, 2009. 422 s. ISBN 978-80-200-1730-7. s. 199-212.

S aplikací elektronek (trioda 1907) a později tranzistorů (1947) a mikroprocesorů (1971) výpočetní potenciál mnohonásobně vzrostl.

Kromě praktických konstruktérů počítačů, kteří vytvořili první funkční počítače a jejich architekturu budovali s odkazem na Babbage a na svých vlastních inovacích, sehráli v dějinách vývoje počítačů roli autoři teoretických modelů jako byl Alan Turing a John von Neumann. Turing, v návaznosti na matematiky Gödela a Hilberta a v hledání odpovědi na otázku rozhodnutelnosti (tzv. Entscheidungsproblem), přistoupil k problematice tak, že ji převedl na otázku vypočitatelnosti.[64] Nezabýval se výslovně tím, zda jsou všechny logické výroky nějaké teorie dokazatelné, ale tím, zda se všechny výpočetní postupy dají zapsat jako algoritmy.[65] Jako pomůcku v hledání odpovědi na tuto otázku vytvořil koncept jednoduchého stroje (tzv. Turingův stroj), jenž sestává ze tří částí: zjednodušeně jde o pásku, čtecí a zapisovací hlavu a instrukční tabulku. Jednorozměrná páska odpovídá paměti počítače, do které hlava zapisuje (nebo čte) znaky na základě definovaných pravidel.[66] Pro Turinga byl jeho stroj „pouze“ matematickým objektem, tím, jak by ho šlo fakticky sestrojit, se příliš neobtěžoval.[67] Přesto jsou ale počítače, jež dnes používáme, jen variací na jeho původní koncepci.

Po předobrazech, jakými byly práce Charlese Babbage, či Turingův stroj, přišel John von Neumann s konkrétním konstrukčním plánem pro výpočetní systém, který je po něm pojmenovaný jako Von Neumannova architektura:[68]



[64]Viz GLEICK, Jaems. *Informace*. Praha: Argo, 2013. 397 s. ISBN 978-80-257-0901-6. s. 166-172.

[65]MAREŠ, Milan. *Slova, která se hodí*. Praha: Academia, 2006. 348 s. ISBN 80-200-1445-5. s. 306.

[66]Detailní popis spolu s řadou příkladů čtenář nalezne v LEAVITT, David. *Muž, který věděl příliš mnoho*. Praha: Argo, 2007. 270 s. ISBN 978-80-7363-086-7. s. 45-86.

[67]Hezkou realizací Turingova stroje je například konstrukce Mike Daveyho. *Aturingmachine.com*. [online]. [cit. 2014-05-02]. Dostupné z: http://aturingmachine.com/.

[68]MAREK, Rudolf. *Assembler pro PC*. Brno: Computer Press, 2007. 228 s. ISBN 80-7226-843-0. s. 21-24.

Počítač tvoří řadič, aritmeticko-logická jednotka, paměť, vstupní a výstupní jednotky. Důležitou vlastností Von Neumannovy architektury je, že struktura počítače je nezávislá na typu řešené úlohy.[69] K její charakteristice dále patří to, že v paměti je uložen program i data. Program v paměti lze, stejně jako data, modifikovat, což souvisí s otevřeností a flexibilitou takové architektury (program zde není „natvrdo" vtělen do hardware). Program obsahuje řídící struktury (tj. možnost „odskočit si" na jiné místo v kódu) a paměť je analogií jednorozměrné pásky z Turingova stroje.

Viděno úhlem pohledu vědeckých revolucí od doby Alana Turinga a Johna von Neumanna ve výpočetní technice nedošlo k zásadnějším koncepčním inovacím. Nové paradigma snad představuje výzkum v oblasti kvantových počítačů.

A znovu se musíme ve výkladu umírnit, ač jsme zde více témat nezmínili, než zmínili. Naznačenou polyfonii souvislostí necháme ev. dohrát čtenáře. Šlo by sledovat další generační vývoj počítačů až k masovému rozšíření 8 bitových a následně 16 a 32 bitových architektur mikroprocesorů v 80. a 90. letech 20. století; rozehrát zde formování a vznik posthistorické informační společnosti a to, jak byla výpočetní technikou podmiňována.[70] Jen letmo uveďme, že Vilém Flusser ji charakterizuje nástupem technických, digitálních obrazů, jejichž základem je binární kód a které smazávají rozdíl mezi realitou a fikcí.[71]

Jiným potenciálně sledovatelným hlasem pokračující melodie by byly úvahy Lva Manoviche nad „novými" médii. Manovich zdůrazňuje moment setkání výpočetních a mediálních technik, jehož výsledkem bylo převedení všech klasických médií (obraz, text atd.) do dat přístupných počítačům.[72]Vlastnosti „nových" médií pak shrnuje kategoriemi modularita, číselná reprezentace, automatizace a variabilita a překódování.[73]

Nezazní zde ani úvahy o rozdílu analogového a digitálního signálu, zpětné vazbě, nebo antropomorfní srovnání činností naší neuronové sítě s architekturou počítače. Teoretických textů, věnujících se počítačovému umění a charakteristice „nových" médií, nalezne informací chtivý čtenář dostatečné množství při prozkoumání citačního aparátu nebo bibliografické sekce.

---

[69]Např. americký počítač ENIAC sloužil k řešení diferenciálních rovnic a programován byl ještě za pomoci drátěných propojek a přepínačů. Následující typ EDVAC, na jehož návrhu se Neumann podílel, byl již řízen podobně jako dnešní počítač, tj. programem uloženým v paměti.

[70]Zajímavou úvahu na téma rozdílů mezi determinací a podmiňováním kultury technikou, částečně spřízněnou s pozicí prezentovanou v této práci, čtenář najde v LÉVY, Pierre. *Kyberkultra*. Praha: Karolinum, 2000. 229 s. ISBN 80-246-0109-5. s. 23.

[71]RUSNÁKOVÁ, Katarína. *V toku pohyblivých obrazov*. Bratislava: Vysoká škola výtvarných umení v Bratislave, 2005. 193 s. ISBN 80-88675-97-9. s. 45.

[72]DVOŘÁK, Tomáš (ed.). *Kapitoly z dějin a teorie médií*. Praha: Akademie výtvarných umění v Praze, 2010. 349 s. ISBN 978-80-87108-16-1. s. 33.

[73]Ibid. s. 34-49.

Smysl předchozích nástinů spočívá v pokusu o vybudování předporozumění Pure Datům jako nástroji, který vyrůstá ze specifického vědecko-technického podhoubí, přičemž sledování dějin tohoto podhoubí lze současně nahlédnout jako schematické dějiny evropského rozumění světu. Vytváření nástrojů (a Pure Data do této kategorie částečně spadají) se do velké míry odvíjí od stavu a výdobytků exaktních věd. Tímto tvrzením se dotýkáme diskuse z první Pure Data Convention[74], kde se prodiskutovávala problematika tzv. *white canvas ideology* – tedy otázka, zda před sebou při zacházení s Pure Daty máme „bílé plátno". Thomas Musil k tomu říká následující:

> Pure Data nejsou sama o sobě prázdným plátnem. Stojí za nimi staletí vědeckých idejí. Pure Data se mohou stát prázdným plátnem ve stejném smyslu jako kladiva, dláta, štětce, housle nebo sbíječky. Pure Data nejsou umění. Pure Data mohou být použita k vytvoření umění stejně jako kladiva, dláta, štětce, housle nebo sbíječky. [75]

Kromě těchto důvodů, jež nás vedly k průzkumu dějinného podhoubí, ale musíme přidat ještě jeden, řekněme postdigitální, důvod. Zatím opět pouze ve zkratce předešleme, že pro postdigitální postoj je charakteristická jednak snaha porozumět nástrojům, se kterými umělec zachází, a jednak kritický postoj k nim. Tomuto postoji jsme se pokusili dostát.

2.6 | Programování

S vynálezem programovatelných strojů se vynořila též otázka, jak těmto strojům zadávat úkoly, nebo jinými slovy: jak je programovat. Zabývali se jí již i Charles Babbage a Ada Lovelace, která bývá považována vůbec za první programátorku.[76] Přirozený jazyk, kterým každodenně komunikujeme naše postřehy, myšlenky a pocity, pro tento účel nelze použít. Aby nám počítač „rozuměl", je třeba na přirozeném jazyku provést redukci: vyloučit nejednoznačnosti a atomizovat mluvení na jednoduché instrukce.[77] Programovací jazyky jsou vlastně naše kon-

[74] Jde o konferenci, na které se setkává Pure Data komunita a diskutuje aktuální témata. Ta první proběhla v roce 2004 v Grazu; zatím poslední v roce 2011 ve Výmaru.

[75] MUSIL, Thomas, WILTSCHE, Harald. I'm the Operator with the Pocket Calculator. *Bang Pure Data* [online]. Hofheim: Wolke Verlag, 2006 [cit. 2014-05-10]. Dostupné z: http://pd-graz.mur.at/label/book. s. 46.

[76] Ada pracovala na problému výpočtu tzv. Bernoulliho čísel. Viz GLEICK, Jaems. *Informace*. Praha: Argo, 2013. 397 s. ISBN 978-80-257-0901-6. s. 98-101. V dějinách bychom samozřejmě našli starší předobrazy: např. Al-Džazárího automatického bubeníka nebo již zmíněného Jacquarda a jeho programovatelný tkalcovský stav.

[77] Viz kapitola o metodě a algoritmu, s. 15-20.

strukce, které umožňují formulovat algoritmy a vznikly jako pomůcka pro zjednodušení přístupu k počítačům.[78]

První počítače a jejich programové vybavení řešily úlohy týkající se výpočtu balistických drah, dnes existují programy obchodující místo lidí na burze nebo schopné napodobit Bachův kompoziční styl a případně jej prolnout s Mozartem.[79] Snaze o formalizování a převádění na algoritmus jakoby dnes snad podléhalo zcela vše.

S programováním a záznamem instrukcí s pomocí tzv. pseudokódu[80] jsme se již do určité míry seznámili v podobě vývojových diagramů. Primitivní algoritmy popisující např. vaření kávy nebo přechod přes silnici by jistě každý za pomoci pár instrukcí a podmínek (řídících struktur) dokázal sestavit. Někteří autoři publikací o programování se dokonce domnívají, že dítě sbírající kostky do krabice je svým způsobem programátor, uplatňující v řešení problému tak pokročilý koncept, jakým je rekurze.[81]

Pseudokód je dobrým pomocníkem v navrhování programů, ale v samotném programování počítače není k ničemu. Chceme-li, aby počítač něco udělal, musí být instrukce pro něj zapsány ve strojovém kódu. Se strojovým kódem se ale zase obtížně pracuje člověku. Proto již v počátcích počítačové éry vznikl jazyk symbolických instrukcí a jeho překladač, neboli assembler. Ten má na starosti překlad námi čitelné a mnemotechnicky zapamatovatelné instrukce (např. NOP, což je zkratka No Operation) do strojového kódu (v případě dnes standardního mikroprocesoru typu x86 je to sekvence 10010000). Přestože je jazyk symbolických instrukcí méně záhadný než strojový kód, má k intuitivnímu programovacímu jazyku pořád dost daleko.

Výsada jazyka symbolických instrukcí spočívá v tom, že programátor může komunikovat s hardware opravdu na nízké úrovni, nicméně vývoj programů je obvykle zdlouhavý. Najít chybu v rozsáhlejším projektu se snadno stane noční můrou. V typologii programovacích jazyků je označován jako nízkoúrovňový, jelikož je svázaný s architekturou mikroprocesoru – program napsaný pro archi-

[78] NAUMANN, Friedrich. *Dějiny informatiky*. Praha: Academia, 2009. 422 s. ISBN 978-80-200-1730-7. s. 331.

[79] Viz David Cope a jeho program Emily Howell. *Artsites.ucsc.edu*. [online]. [cit. 2014-01-04]. Dostupné z: http://artsites.ucsc.edu/faculty/cope/Emily-howell.htm.

[80] Alternativní, krátký a výstižný úvod do pseudokódu a pseudoprogramování představuje např. ERICKSON, Jon. *Hacking – umění exploitace*. Brno: Zoner Press, 2009. 544 s. ISBN 978-80-7413-022-9. s. 19-32.

[81] WRÓBLEWSKI, Piotr. *Algoritmy*. Brno: Computer Press, 2004. 351 s. ISBN 80-251-0343-9. s. 27-28. Je otázkou, zda je toto přirovnání přiměřené. Problém sbírání kostek je sice řešitelný s pomocí rekurze, kdy funkce volá sebe samu, ale jednodušeji to lze provést s pomocí iterace. Více k problematice viz poznámka č. 81.

tekturu jednoho procesoru nebude fungovat na architektuře jiného procesoru.[82]

Tyto potíže vedly k vývoji tzv. vysokoúrovňových programovacích jazyků. Od 50. let 20. století jich vzniklo několik desítek, jmenujme aspoň ty nejznámější: Fortran, BASIC, Lisp, Smalltalk, Pascal, C, Java, Python atd. Jejich charakteristikou je to, že jdou dál od jazyka stroje a přibližují se k pseudokódu – tedy k způsobu formulování a řešení problémů, který je blízký lidem. Programy zapsané ve vysokoúrovňových jazycích jsou srozumitelnější a lépe čitelné než zápis v jazyku symbolických instrukcí, proto se také snadněji odlaďují. Výhodou těchto jazyků je též nezávislost na hardwarové platformě. Jednou napsaný kód lze totiž s pomocí tzv. kompilátoru přeložit do strojového kódu pro tu kterou platformu.

Dodejme ještě, že programovací jazyky se dělí na kompilované (C, Pascal) a interpretované (Python, Java). V kompilovaných jazycích se čitený program převadí do „nečitelného“ strojového kódu – výstupem kompilace je rovnou spustitelný a na původním zdrojovém programu nezávislý binární soubor. Interpretované jazyky žádný spustitelný soubor nevytváří – kromě původního programu k jeho rozběhnutí potřebujeme ještě interpret, který zdrojový kód provádí.

Programátor, poté co navrhne a napíše kód, jej – podle toho v jakém jazyku tvořil – buď zkompiluje, nebo nechá interpretovat. Pokud se během kompilace či za běhu programu objeví nějaké chyby, následuje fáze hledání a opravování chyb, optimalizace a ladění. Fáze psaní, kompilování/interpretování a ladění je zjednodušeně viděno vlastně celý cyklus vývoje software.[83] Zvláštností Pure Dat je to, že patří k jazykům, které kód interpretují bezprostředně po napsání, tzv. v reálném čase. Co to ve svém důsledku znamená si záhy ukážeme.

Hranice jazyka jsou hranicemi světa.[84] Například jazyk jihoamerických indiánů z kmene Pirahã nemá žádná slova označující barvy nebo vzdálenou minulost a budoucnost. Některé jazyky zdůrazňují víc procesy, jiné věci. Podobně je tomu i u programovacích jazyků: v některých se dobře zachází s textem, v jiných se zvukem. To, co s v některých dá říct jednoduše, je v jiných na hranici možnosti uskutečnění, což je ostatně důvod existence takového množství programovacích jazyků. Volba programovacího jazyka je současně volbou stylu, jakým budeme o pojednávaném tématu mluvit, nebo jinak: s jakou mírou složitosti budeme řešit zadaný problém. Pokusit se v Pure Datech naprogramovat rozsáhlou textovou databázi by nebylo zrovna nejmoudřejší rozhodnutí.

---

[82] ERICKSON, Jon. *Hacking – umění exploitace*. Brno: Zoner Press, 2009. 544 s. ISBN 978-80-7413-022-9. s. 18.

[83] Někteří autoři chápou dokonce celý proces programování jen jako hledání a odlaďování chyb.

[84] WITTGENSTEIN, Ludwig. *Tractatus logico-philosophicus*. Praha: OIKOYMENH, 2007. 87 s. ISBN 978-80-7298-284-4. s. 65. Tvrzení 5.6.

Během vývoje řady programovacích jazyků se na základě jejich charakteru, rozdílů a aplikovatelnosti konstituovala základní programátorská paradigmata.[85] Pro výčet a detailní popis zde není prostor, proto si ve zkratce představíme jen objektově orientované a *dataflow* paradigma.

Jazyky jako Java nebo Smalltalk patří do objektově orientovaného paradigmatu, ve kterých jsou problémy a svět popisovány s pomocí objektů. Noty, barvy, události, dokumenty – jakákoliv skutečnost je v těchto jazycích chápána jako objekt s určitými vlastnostmi. Každý objekt je nositelem informací o sobě samém – o tom, co se sebou může sám dělat a o svém stavu. Objekt má také schopnost na požádání svůj stav měnit. Například objekt nota má vlastnosti výška a délka; tyto parametry lze měnit podle daných kompozičních pravidel; stavem pak může být to, jestli zní, nebo ne.

Dataflow paradigma, do něhož spadají i Pure Data, je charakteristické tím, že program chápe jako orientovaný graf. Stěžejním pojmem zde není objekt, ale data a to, jak „tečou“. Změna nějaké proměnné v programu vede automaticky k přepočítání hodnot s touto proměnou svázaných. Proto je toto paradigma vhodné pro zpracování signálů. Obrazně si programování v dataflow paradigmatu lze představit jako operace probíhající na vodním toku – stavěním (programováním) hrází, čističek, zavlažovacích kanálů atd. proud (signál) nabývá nových kvalit, ohýbá se a směřuje tam, kam ho chceme dovést. Je nabíledni, že pro reprezentaci programu v dataflow paradigmatu bude nejvhodnější použít schéma nebo diagram. Krok od jazyka symbolických instrukcí směrem k vyšším programovacím jazykům by se dal interpretovat jako emancipační gesto, jímž se svět programování stává srozumitelnější a přístupnější širšímu publiku. Druhou vlnou emancipace je pak přechod od „jednorozměrných“ programovacích jazyků, ve kterých se program zapisuje do řádku, k „dvourozměrným“, kde je zobrazen jako diagram v ploše.[86] Programy jsou díky vizuální reprezentaci intuitivně pochopitelné. Pure Data, stejně jako řada jiných programovacích jazyků „pro umělce“, tak sledují koncepci snadného přístupu k programování a rychlého prototypování.[87]

Dionýsky naladěný a programátorsky intaktní čtenář by mohl mít námitky, týkající se neslučitelnosti světa umění a programování. Jeho argumenty by se patrně týkaly přílišné intelektualizace, desubjektivizace, odcizení a rigidnosti. Apol-

[85]Dobrým úvodem je např. skriptum Davida Skoupila. Viz *Phoenix.inf.upol.cz* [online]. [cit. 2014-02-13]. Dostupné z: http://phoenix.inf.upol.cz/esf/ucebni/uvod_para.pdf.

[86]Analogii nacházíme také v přechodu od ovládání počítačem jen s pomocí terminálu ke grafickému uživatelskému rozhraní.

[87]RITSCH, Winfried. Does Pure Data Dream of Electric Violins? *Bang Pure Data* [online]. Hofheim: Wolke Verlag, 2006 [cit. 2014-05-10]. Dostupné z: http://pd-graz.mur.at/label/book. s. 14.

linské protiargumenty by zase hájily programování jako druh kontemplativní činnosti a optimalizaci kódu by interpretovaly v duchu přímé úměrnosti mezi jednoduchostí a elegancí. Dionýský postoj by nad programováním mohl též „zlomit hůl“ kvůli předurčenosti: jako by se zdálo, že v programování vždy musíme jen naplnit nějaký předem jasně definovaný cíl – jako by se někam vytrácela náhoda, kouzlo nechtěného a improvizace. Zde bych rád vstoupil do jemné polemiky s Kryštofem Peškem, který píše:[88]

> Programovací jazyk je přednostně zkonstruován pro definici známého a pochopeného. V případě neznámých nebo nepoznaných veličin je programovací jazyk víceméně k ničemu.

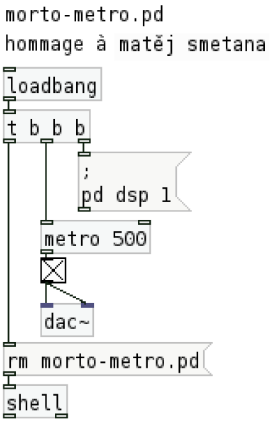
V Pure Datech samozřejmě jde naprogramovat již známé, pochopené a předem definované koncepty. Díky tomu, že je kód ale interpretován bezprostředně po zapsání[89] a díky vizuálnímu rozhraní, se programování přibližuje performativnímu aktu nebo jam-session. Touto cestou je možné v Pure Datech (a v podobně koncipovaných jazycích) dospět prostřednictvím programátorské improvizace[90] k neznámému a neočekávanému. Postava programátora nevylučuje Dionýsia ani Apollóna. Pokračovat v argumentech sledujících vzájemnost mezi programováním a uměním by se podobalo apologii samozřejmého.[91] Zakončíme tuto kapitolu raději prostým tvrzením: zdá se, že od doby, co jsou zde počítače a programovací jazyky umožňující různé řečové akty, je zde i počítačové a softwarové umění.

[88]PEŠEK, Kryštof. *Processing Beta*. Praha: Akademie múzických umění, 2013. 152 s. ISBN 978-80-7331-224-4. s. 29.

[89]Jako by se „přeskakoval“ tradiční cyklus psaní a interpretování/kompilace kódu.

[90]Viz kapitola o live codingu.

[91]Tato apologie proběhla v řadě textů. Viz bibliografie.


morto-metro.pd
hommage à matěj smetana
loadbang
t b b b
;
pd dsp 1
metro 500
dac~
rm morto-metro.pd
shell

3 | Rukověť

Běda těm, kdo lehají na ložích ze slonoviny, povalují se na pohovkách, jídají jehňata ze stáda a telata z chléva, blábolí za zvuku harfy, vymýšlejí si hudební nástroje jako David…

Am 6, 4-5

Pure Data (dále jen Pd) je open-source grafický programovací jazyk vyvíjený od druhé poloviny 90. let 20. století Millerem Puckettem[1] a širokou komunitou programátorů[2], jehož možnosti překračují výrazovou škálu tradičně dostupných programů sloužících ke zpracování videa a zvuku. Spolu s dalšími programovacími jazyky, jakými jsou Processing, SuperCollider, VVVV, Fluxus a jiné, se Pd staly výrazným prvkem formujícím podobu audiovizuální scény. Jednoduše řečeno: v širším měřítku došlo k posunu od pouhého užívání software k jeho vytváření.

Při prvním hledání odpovědi na otázku čím Pd jsou použijeme jednoduchý příměr: Pd jsou nástrojem k vytváření nástrojů. Jako úvodní a velmi obecná definice by mohla dobře posloužit, ale přesto není zcela přiléhavá. Detailnější pohled nám odhalí pojmovou distinkci mezi nástrojem a nářadím[3], proto by bylo vhodnější Pd vidět jako základní nářadí, s jehož pomocí lze vytvářet nástroje. To je ale pořád nepřesná definice. Problém spočívá v tom, že do pokusů definovat jazyk se projektují způsoby jeho užívání.

Malíř, hledaje specifičnost výrazu, se nemusí zastavit u konečného výsledku reprezentovaného dílem, může vytvářet své vlastní štětce, barvy i plátno[4], a po-

[1] Miller Puckette je programátor, umělec a matematik, který působil na IRCAMu. V současnosti vyučuje na University of California v San Diegu. Viz *Msp.ucsd.edu* [online]. [cit. 2014-01-04]. Dostupné z: http://msp.ucsd.edu/. Je součástí improvizačního tria The Convolution Brothers.

[2] *Puredata.info* [online]. [cit. 2014-03-20]. Dostupné z: http://puredata.info/.

[3] O kladivu spíš řekneme, že je nářadím než nástrojem. O klarinetu zas hovoříme jako o nástroji a ne jako o nářadí.

[4] Jaký to má důvod a význam ve vztahu k dění na „novo-mediální" scéně se budeme věnovat v předposlední kapitole nazvané *Souvislosti*.

dobně se to má i s hudebníkem, video-artistou nebo umělcem pohybujícím se v oblasti intermédií. V Pd lze uskutečnit syntézu – jak mezi jednotlivými žánry (hudební kompozice, videoart), tak mezi slovem, obrazem a zvukem. Aplikovatelnost Pd vede napříč videoartem, VJingem, scénografií, interaktivními instalacemi, až k algoritmickým kompozicím, procedurální hudbě, grafickým partiturám[5], konstrukci syntetizérů a komplexních systémů. Jeho možnosti však nekončí na úrovni software – v kombinaci s hardwarovým rozhraním Arduino[6] lze v Pd na vstupu zpracovávat data z analogových senzorů a na výstupu je možné si představit libovolné elektromechanické zařízení. Krásným příkladem je „mluvící piano" od umělce Petera Ablingera.[7] Další charakteristikou tohoto jazyka spočívají v tom, že umí komunikovat po síti a jeho kód je možné modifikovat přímo za běhu programu – program tak již není „mrtvým" kódem, který je nutné před spuštěním zkompilovat, ale je bezprostředně interpretován. Spolu s těmito vlastnostmi se objevily i nové performativní žánry.[8] Výběr z prací realizovaných s pomocí Pd čtenář najde v kapitole s názvem *Aplikace*.

Domnívám se, že díky Pd a dalším jazykům zažila „novo-mediální" a audiovizuální scéna období euforie, jako je tomu vždy, když objevíme nový prostor pro myšlení a konání. Ona euforie ale souvisí i s frustrací. Puckette k tomu říká:

> Má oblíbená metafora je o srovnání dvou místností, jedné malé a druhé velké. Když žijete v té malé, možná se budete chtít přemístit do té větší, abyste měli méně ohraničení. Větší místnost má ale pochopitelně větší povrch a tím i větší hranice. Podobně se může stát, že čím více průhledné, pružné a mocné nářadí používáte, tím více způsoby můžete narazit na zeď. Takže, ačkoliv se vám Pd pokouší zprostředkovat více svobody, nakonec vám dává ty nejbohatší příležitosti k frustraci.[9]

Frustrace, jak ji myslí Puckette, je patrně mnohovrstevná. Účelem zde předkládané rukověti je minimalizovat frustraci související s neschopností mluvit, takže na následujících stranách čtenář najde příručku uvádějící do programování v Pd. Zaměřuje se přitom na úplné základy jazyka a nepředpokládá nic kromě minima z počítačové gramotnosti. Může dobře posloužit hudebníkům, video a sound

[5] Viz s. 170-171.

[6] *Arduino.cc* [online]. [cit. 2014-03-20]. Dostupné z: http://www.arduino.cc/.

[7] Speaking Piano. In: *YouTube* [online]. 5. 10. 2009. [cit. 2014-03-21]. Dostupné z: https://www.youtube.com/watch?v=muCPjK4nGY4. Kanál uživatele TheMcphearson. Viz též: *Ablinger.mur.at* [online]. [cit. 2014-03-21]. Dostupné z: http://ablinger.mur.at/speaking_piano.html.

[8] Viz *livecoding*, s. 208-210.

[9] SCHEIB, Christian. Two Rooms. *Bang Pure Data* [online]. Hofheim: Wolke Verlag, 2006 [cit. 2014-04-01]. Dostupné z: http://pd-graz.mur.at/label/book. s. 168.

artistům, divadelníkům, filmařům, intermediálním umělcům a kutilům, podivínům, studentům či vůbec komukoliv, kdo hledá takovou „větší místnost“ – tj. nový a relativně jednoduchý programovací jazyk a pro koho je samotné setkání s novým jazykem dostatečně dobrodružnou výzvou. Po nastudování rukověti by čtenář měl být dostatečně vybaven k tomu, aby dokázal realizovat vlastní projekty.

Nejprve se budeme v Pd učit „žvatlat“ a osahávat hranice jeho možností. Mají-li z tohoto setkání ale vytanout významy a pragmatické přesahy, nelze zcela predikovat, jelikož toto se do určité míry odvíjí od učedníka. Nicméně bych čtenáře na této cestě rád povzbudil: díky uživatelské přívětivosti v podobě grafického rozhraní a díky špetce čtenářova úsilí smíchaného s chutí experimentovat se Pd mohou v jeho rukou stát takřka univerzálním nářadím.

Zároveň bych ale čtenáře chtěl varovat před úskalím: následující stránky jsou totiž vedeny perspektivou a užíváním Pd, které je vlastní autorovi. Proto bych též poprosil zkušenější programátory o shovívavost či věcnou kritiku týkající ev. vylepšení těchto studijních materiálů. Ačkoliv jsem se v rukověti pokusil zachovat srozumitelnost a pojednat úvod do jazyka i základní aplikace v patřičné obecnosti, nelze si odmyslet, že se do celkového vyznění promítá to, jak Pd sám používám. Je to pouze jeden z možných přístupů – pokud by čtenář měl potřebu jej na základě vlastních přání ohnout, nebo opustit, ať tak směle učiní.

Vzpomínám si, jak náročné pro mě, jako pro neprogramátora, byly první kroky a pokusy s Pd. Dostupný byl pouze oficiální manuál a dokumentace od Millera – studijní materiály bezesporu na vysoké úrovni, ale pro začátečníka poněkud nevhodné. Významná pro mě byla návštěva Letných dielní, kde jsem díky tutorství Petera Gondy[10] a Daniela Tótha[11] začal s intenzivnějším samostudiem a psaním prvních kódů. Vedlejším „produktem“ mého „úsilí“ byla série workshopů, které se konaly v roce 2008 na Akademii výtvarných umění v Praze. Tutoriály jsem připravil převážně na základě vlastních zkušeností a také zkušeností, jež jsem nabyl studiem kódů druhých – zejména šlo o práce Andyho Farnella publikované na jeho webu a později vydané jako kniha *Designing Sound*. Tehdy ještě nebyl k dispozici výborný úvod do Pd od Johannese Kreidlera nebo série neméně kvalitních FLOSS tutoriálů. Východiska, postupy a strukturování témat jiných autorů příruček nebyly až tolik odlišné. V mnoha aspektech byly jejich příručky mnohem lepší – teprve díky nim jsem byl s to objevit „hrubky“ ve vlasních kódech. Proto jsem se ostatně také rozhodl se ke starým tutoriálům vrátit a na základě výše zmíněných je opravit a doplnit. Věřím, že budou čtenáři k užitku.

[10] *Gondapeter.sk* [online]. [cit. 2014-03-20]. Dostupné z: http://gondapeter.sk/.
[11] *Poo.sk* [online]. [cit. 2014-03-20]. Dostupné z: http://poo.sk/.

3.1 | Vývoj Pd

Před tím, než se pustíme do instalace Pd na váš počítač, si dovolím ještě jeden krátký úkrok stranou. Bude to takový dozvuk první kapitoly. V ní jsme prozkoumali v obecné šíři dějiny předpokladů počítačového a softwarového umění, ale nedotkli jsme se konkrétních okolností vzniku Pd. Čtenář, který je již historickými exkurzy unaven, může tuto část přeskočit. Sám nicméně považuji za důležité zasadit Pd do užšího kontextu vývoje hudebního software a do kontextu „revolucionizace" hudby počítačem.

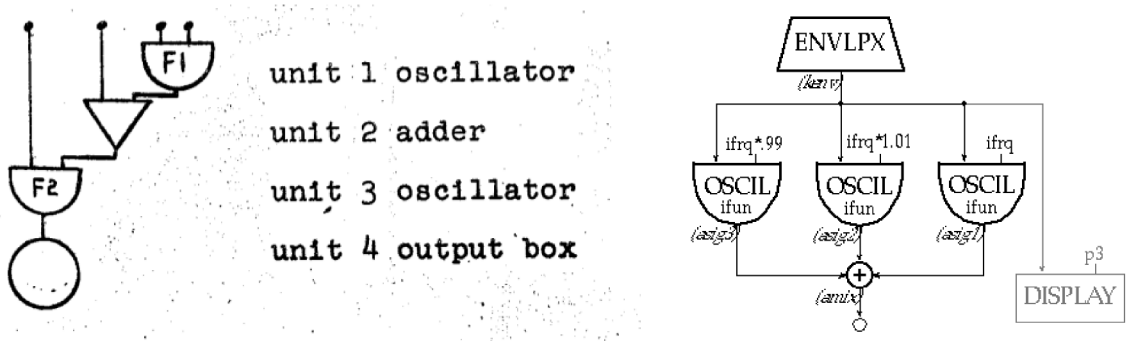
Za obecně uznávané milníky v dějinách počítačové hudby bývají považovány jednak skladba *Illiac Suite* z roku 1956, kterou zkomponoval počítač, a jednak vznik první verze programovacího jazyka z rodiny Music-N od Maxe Mathewse. Mathews pomocí Music I v roce 1957 syntetizoval asi 17 vteřin dlouhou kompozici. Obojím byly položeny základy pro to, co se dnes označuje zkratkami CAC a CGM – tedy komponování s pomocí počítače (Computer Aided Composition) a počítačem generovaná hudba (Computer Generated Music).[12]

V tom, co je důležité, vzhledem k Pd, se budeme držet právě Mathewse. Ten ve verzi Music III přišel s konceptem tzv. jednotkových generátorů (*unit generators*, ve zkratce také *ugens*), které se staly univerzálně používaným modelem. Pomocí *ugens* lze snadno popsat zapojení jednotlivých částí syntetizéru, ať už jde o oscilátory, filtry nebo obálky.[13] Tento model najdeme v řadě následujících jazyků a aplikací, např. v programovacím prostředí Csound od Barryho Vercoea, komerční aplikaci Reaktor, programovacích jazycích ChucK a SuperCollider a konečně také u Pure Dat. Puckette se domnívá, že to byl Mathews, kdo konceptem jednotkových generátorů předjímal analogové modulární syntetizéry typu Moog Modular. Právě k zacházení s modulárním syntetizérem, resp. k tomu, jak se v něm zapojují jednotlivé moduly s pomocí kabelů, je možné programování v Pd dobře přirovnat. Podívejme se pro srovnání, jak vypadá schéma s generátory v jazyce Music IV a v Csoundu.[14]

[12]PUCKETTE, Miller. Computing While Composing. In: *Msp.ucsd.edu* [online]. [cit. 2014-01-04]. Dostupné z: http://msp.ucsd.edu/Publications/om-reprint.pdf.

[13]Pro zběžný úvod do problematiky a dějin syntetizérů viz GUŠTAR, Milan. *Elektrofony II*. Praha: Uvnitř, 2008. 518 s. ISBN 978-80-239-8447-7. s. 23-43.

[14]Převzato z *Ccrma-ftp.stanford.edu* [online]. [cit. 2014-01-04]. Dostupné z: ftp://ccrma-ftp.stanford.edu/pub/Lisp/music-iv-programmers-manual.pdf a *Csounds.com* [online]. [cit. 2014-01-04]. Dostupné z: http://www.csounds.com.



Různé zobrazení *ugens* v Music IV a Csound.

Kromě Mathewsova přínosu by zde Pd nebyla bez konceptu vizuálního programování, které jako první prakticky představil v roce 1966 William R. Shutherland[15], a bez stále vzrůstajícího výpočetního potenciálu počítačů. Mathews na to vzpomíná takto:

> Rané počítače nemohly zahrát skladbu v reálném čase – spočítání byť jen jedné vteřiny zvuku trvalo velmi dlouho. Dnes je ale hraní hudby v reálném čase možné, a tak se laptopy připojují ke sborům a orchestrům, aby novými a bohatými témbry doplnily krásu témbrů akustických nástrojů.

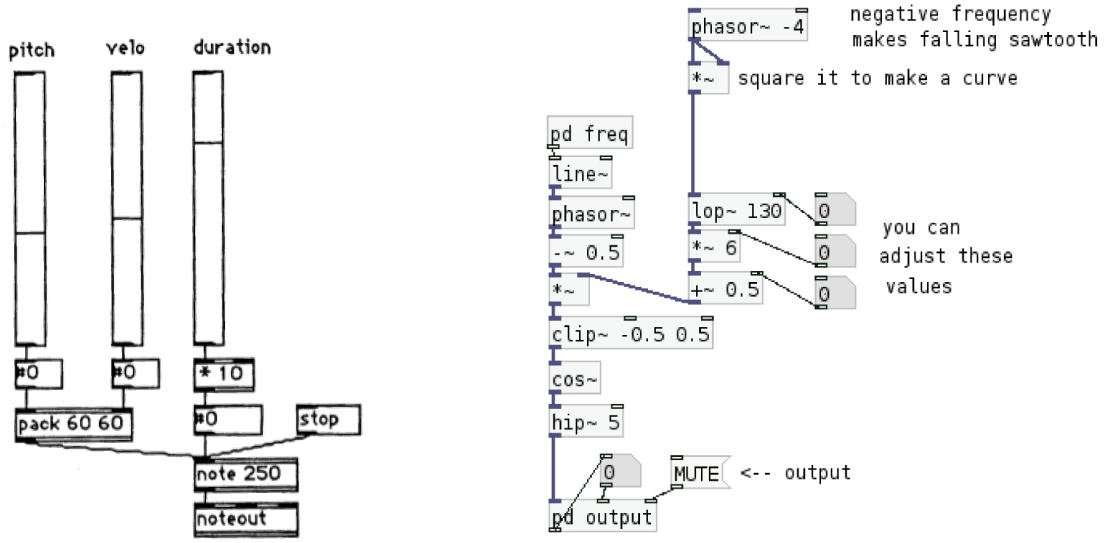
Mathews i Vercoe se problematikou zpracování zvuku v reálném čase zabývali a Miller Puckett je oba zmiňuje jako dva velké inspirátory.[16] Již na počátku 80. let Puckette pod vedením Barryho Vercoea na MIT prováděl první experimenty na počítači PDP 11, poté ho Vercoe v roce 1985 přizval k výzkumu na IRCAM, kde vytvořil Music 500, Max a Patcher. První verze Maxu neměla ještě grafické uživatelské rozhraní – tím byl právě až Patcher (1988), který v podstatě obsahoval již všechny zásadní prvky, jaké má dnes Max/MSP a Pd: objekty se vstupy a výstupy, jež lze spojovat kabely; rozlišení mezi editačním a performačním módem; základní aritmetiku; implemetnaci MIDI atd.[17] Patcher už samotným názvem odkazuje na prostředí modulárních syntetizérů, které se „programují propojováním“[18] – zapojení se říká *patch* a stejně se označují i kódy napsané v Maxu nebo Pd.

[15]Programming with On-Line Graphics. In: *YouTube* [online]. 9. 08. 2011 [cit. 2014-01-21]. Dostupné z: https://www.youtube.com/watch?v=NLyIYmPfCps. Kanál uživatele Bill Buxton.

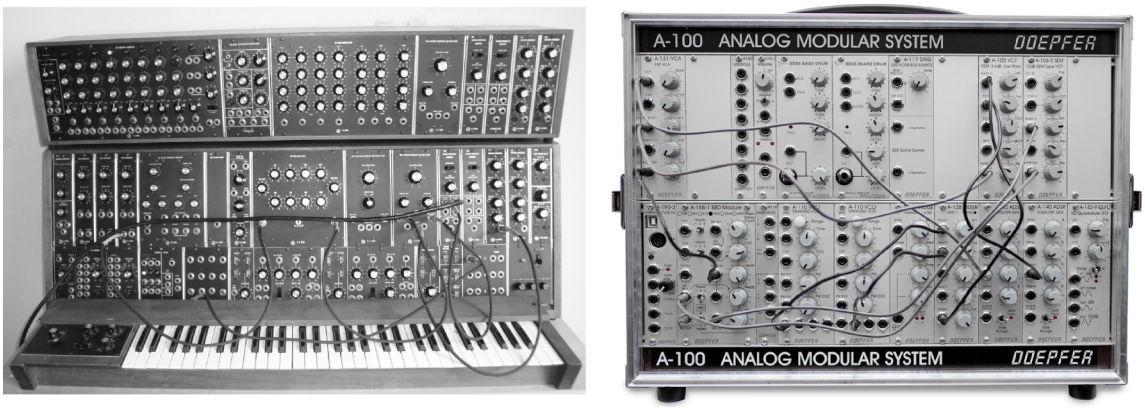
[16]PUCKETTE, Miller. Max at Seventeen. In: *Msp.ucsd.edu* [online]. [cit. 2014-02-21]. Dostupné z: http://msp.ucsd.edu/Publications/dartmouth-reprint.dir/.

[17]PUCKETTE, Miller. The Patcher. In: *Msp.ucsd.edu* [online]. [cit. 2014-01-04]. Dostupné z: msp.ucsd.edu/Publications/icmc88.pdf.

[18]5 Minute Romp thru the IP. In: *YouTube* [online]. 13. 12. 2007. [cit. 2014-04-01] Dostupné z: https://www.youtube.com/watch?v=8qh6jRzjmcY. Kanál uživatele evitube. Nebo na patchování Moog Modularu: moog modular tutorial clip 17 fun1. In: *YouTube* [online]. 26. 09. 2009. [cit. 2014-04-01]. Dostupné z: https://www.youtube.com/watch?v=zarAMHlbxZQ. Kanál uživatele ketchupok.



Vlevo Patcher z roku 1988 a vpravo Pure Data.



Zapatchovaný Moog (vlevo) a Doepfer (vpravo).

Po delším vývoji Maxu, který byl s úspěchem nasazován v hudebních vystoupeních, se Miller v roce 1994 přemístil z IRCAMu do San Diega, kde začal pracovat na Pd. Základy obou jazyků jsou obdobné, nicméně na Pd chtěl Puckette pokročit ve vývoji dál a překonat to, co na Maxu v té době vnímal jako omezení: jednou z klíčových inovací bylo uvedení grafických datových struktur, které mohou reprezentovat např. hudební partitury. Dále Miller vytvořil sérii základních DSP objektů, které umožnily syntézu zvuku přímo v počítači, na kterém Pd běží (ranější verze Maxu jménem Max/FSP pouze posílala instrukce k provedení výpočtů DSP jednotce ISPW). Tyto objekty pak byly importovány do Maxu, který se na Millerovu počest přejmenoval na Max/MSP.

S přesunem na akademickou půdu patrně souvisí to, že byla Pd vydána jako open-source, tzn. že jsou dostupné jejich zdrojové kódy. Na otázku, proč se tak rozhodl, Miller odpovídá:[19]

[19]KRATOCHVÍL, Matěj. Budoucnost patří algoritmům. *His Voice*. 2006, roč. 6, č. 1, s. 8-9.

> S Maxem jsem získal zkušenost, že pokud se rozhodnete zveřejňovat software a chcete z toho mít zisk, může vám to snadno znemožnit práci na něm. … Tvořit software jako open-source, tedy volně přístupný, pro mě znamená jistotu, že budu moci svou práci vždy šířit. Navíc k mé práci mohou přispívat i ostatní. V současnosti už podíl jiných autorů na Pd zastiňuje ten můj.

Rozhodnutí se pro open-source ve svém důsledku vedlo k dynamickému rozvoji Pd a vzniku širokého okruhu uživatelů. Ač byla Pd navržena primárně pro generování a zpracování audiosignálu, již kolem roku 1995 Mark Danks začíná pracovat na externí knihovně GEM (Graphics Environment for Multimedia), sloužící k zpracování obrazu a videa. Na základě přirozené poptávky a programátorských schopností vznikla k dnešnímu dni celá řada knihoven, které možnosti aplikace Pd rozšiřují. Jen namátkou: knihovna *pmpd* simuluje fyzikální jevy; knihovna *ann* implementuje neuronové sítě; v knihovně *ggee* najdeme rozšíření, která nám umožní posílat z Pd příkazy do systémové konzole; knihovna *Pd_OpenCV* se zaměřuje na detekci a analýzu pohybu a tvarů atd.

Pd jsou krásným příkladem kolektivního vývoje software a vůbec toho, jaký je étos open-source komunity. V eseji *Katedrála a tržiště* od Erika S. Raymonda[20] je tento étos mezi řádky definován trojím: sebemotivací, otevřeností ke vstupům a nízkou hierarchizací. Open-source software je výsledkem sdíleného vědění a nadšení řady programátorů pro danou věc. Připojit se k vývoji některého z projektů se podobá vstupu do dílny, kde se vedle sebe pohybují mistři i učedníci pracující na daném díle – sdílení vědomostí a precizace kódu je samozřejmostí. Výsledkem pak mohou být obecně prospěšné aplikace jako internetový prohlížeč Firefox, webový server Apache, operační systém Linux, encyklopedie Wikipedie. K výhodám, které vycházejí z toho, že na vývoji open-source projektů obvykle participuje více programátorů než je tomu u proprietálních softwarů, patří to, že se ve vývoji rychleji odhalují a odlaďují chyby. Rychleji se také implementují nové návrhy na vylepšení. Nemusíte být totiž nutně programátorem, ale jen uživatelem vyvíjeného software, který na chybu či nápad poukáže. Potenciál open-source vývoje a jeho přednosti si ostatně už dobře uvědomily i velké komerční společnosti, protože jim začal brát vítr z plachet.

Pd za sebou mají od poloviny 90 let 20. století již poměrně dlouhý vývoj. Osobně se domnívám, že to hlavní se v něm již odehrálo. Současný trend poukazuje spíš na implementaci různých knihoven, nebo na přeprogramování grafického uživatelského rozhraní, které za Max/MSP trochu pokulhává. Za dlouhou dobu

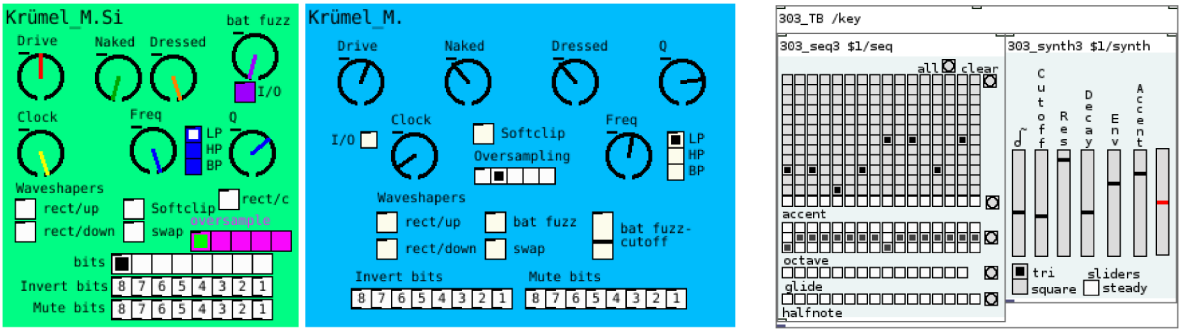
ISSN 1213-2438.

[20] *Root.cz* [online]. [cit. 2014-04-02]. Dostupné z: http://www.root.cz/knihy/katedrala-a-trziste/.

svého vývoje se nicméně Pd stala robustním a prověřeným softwarem. O tom nesvědčí jen to, že jsou používána napříč různými oblastmi v umění, ale také jejich postupné etablování se v akademickém prostředí – na řadě uměleckých škol patří znalost Pd do standardního studijního vybavení.

3.2 | Pd komunita aneb alternativní studijní zdroje

Dobrým pomocníkem ve studiu vám nemusí být jen tato rukověť, ale též komunita vývojářů a uživatelů. Porozhlédnout se po zdrojích dostupných na síti je dobrou příležitostí, jak objevit jiné postupy a témata. Především lze doporučit *Pd Forum*[21], kde najdete řadu tematických sekcí. Je na něm přítomna řada zkušených uživatelů, kteří vám po registraci rádi zodpoví případné otázky – tedy pokud jste je po důkladném hledání na fóru nenašli. V oddíle *patches* se nacházejí nejrůznější ukázky kódů, o které se uživatelé chtějí podělit, nebo je společně vyvíjejí – doporučeníhodné jsou např. balíčky předpřipravených modulů od uživatelů *hardoff* nebo *Maelstorm*. Za všechny bych uvedl jednu hezkou ukázku komunitní spolupráce, jakou byl vývoj osmibitového filtru, který prošel řadou připomínek, až dospěl ke konečné podobě s názvem *krümel_msi*.[22] Porozhlédnout se po Pd fóru určitě stojí za to, i když dobrat se k užitečnému a kvalitnímu patchi někdy může trvat delší dobu. Najít na něm můžete ale i minimální skvosty v podobě patchů, které demonstrují, co všechno jde vytvořit s jediným oscilátorem.[23]



Vlevo dvě různé verze filtru, vpravo variace na syntetizátor Roland TB-303 od *hardoffa*.

V Grazu se kolem Institutu pro elektronickou hudbu a akustiku (IEM) zformovala živá komunita Pd uživatelů, kteří mají na svědomí uspořádání první Pd Convention a následné vydání sborníku Bang Book.[24] Kromě toho se v Grazu nachází

[21] *Pure Data forum* [online]. [cit. 2014-04-02]. Dostupné z: http://puredata.hurleur.com.

[22] *Pure Data forum* [online]. [cit. 2014-04-02]. Dostupné z: http://puredata.hurleur.com/sujet-3903-emulate-oto-biscuit. Patche jsou dostupné ke stažení až po registraci.

[23] *Pure Data forum* [online]. [cit. 2014-04-02]. Dostupné z: http://puredata.hurleur.com/sujet-6441-challenge-osc-dac.

[24] *Bang Pure Data* [online]. Hofheim: Wolke Verlag, 2006 [cit. 2014-05-10]. Dostupné z: http://pd-graz.mur.at/label/book.

domovský server[25] Pd komunity, na němž najdete aktuální informace – ať už jde o vydání nové verze Pd, nebo některé z knihoven. Tento server vám dobře poslouží jako základní rozcestník. Mezi další, i když již ne tak aktivní buňky na síti, patří servery goto10.org a artengine.ca. „Na živo“ jsou k zastižení někteří z členů Pd komunity přes IRC.[26] Pd kanály neabsentují ani na sociálních sítích.

Mezi alternativními studijními materiály bych rád zmínil web Andyho Farnella[27], kde čtenář najde velmi srozumitelný úvod do syntézy zvuku a komponování s pomocí Pd. Andy své tutoriály a řadu nepublikovaných textů sebral do jednoho masivního korpusu jménem *Designing Sound*.[28] Zabývá se v něm především problematikou syntetizování reálných zvuků jako je déšť, oheň, chůze atd. Každá kapitola je uvedena nejprve detailní fyzikální analýzou, pak následuje vytvoření obecného modelu pro syntézu a teprve jako třetí přichází na řadu konkrétní implementace v Pd. Jeho kniha je výbornou učebnicí metodologie programování a také poslouchání.

Pokud by čtenář hledal systematický úvod vedený z perspektivy hudebního skladatele, může nahlédnout do knihy[29] od Johannese Kreidlera, jež je dostupná také on-line. Série tutoriálů uvádějící do Pd ze širší perspektivy se nachází na webu FLOSS Manuals.[30]

3.3 | Instalace a první spuštění

Pd jsou multiplatformní a přenosný software – existují ve verzi pro operační systémy Linux, MacOS i Windows. Kromě toho je ale možné je díky knihovně *libpd*[31] spustit i na mobilních operačních systémech Android a iOS. Při realizaci dlouhodobých či úzce profilovaných projektů potěší, že je podporují i minimální hardwarové platformy jako Raspberry Pi[32] nebo UDOO.[33]

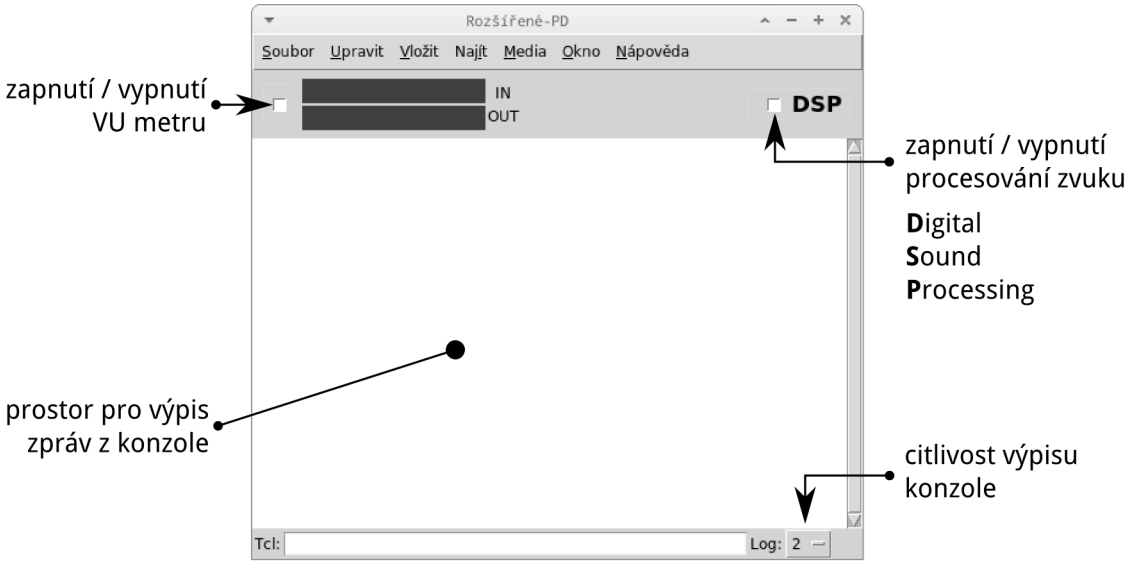
Pd jsou zdarma ke stažení na domovské stránce Millera Pucketta[34], kde najdeme tzv. *Pd-vanilla*. Jde o základní verzi, která neobsahuje žádné externí knihovny, ale pouze základní prostředky ke zpracování zvuku. Robustnější verze, obsahující sbírku externích knihoven, se nazývá *Pd-extended* a lze ji stáhnout z ko-

[25] *Puredata.info* [online]. [cit. 2014-03-20]. Dostupné z: http://puredata.info/.
[26] *#dataflow* [online]. [cit. 2014-04-02]. Dostupné z: irc.freenode.net.
[27] *Obiwannabe* [online]. [cit. 2014-04-02]. Dostupné z: http://obiwannabe.co.uk/.
[28] FARNELL, Andy. *Designing Sound*. Cambridge: MIT Press, 2010. 688 s. ISBN 978-0262014410.
[29] *Pd-tutorial.com* [online]. [cit. 2014-04-02]. Dostupné z: http://www.pd-tutorial.com/.
[30] *Flossmanuals.net* [online]. [cit. 2014-04-02]. Dostupné z: http://en.flossmanuals.net/pure-data.
[31] *Libpd.cc* [online]. [cit. 2014-04-02]. Dostupné z: http://libpd.cc.
[32] *Raspberrypi.org* [online]. [cit. 2014-04-02]. Dostupné z: http://www.raspberrypi.org/.
[33] *Udoo.org* [online]. [cit. 2014-04-02]. Dostupné z: http://www.udoo.org/.
[34] *Msp.ucsd.edu* [online]. [cit. 2014-04-02]. Dostupné z: http://msp.ucsd.edu/software.html.

munitního portálu puredata.info.[35] Pd jsou stále ve vývoji, a tak na Millerově stránce vždy najdete verzi, která bude novější než ta komunitní. Pro vaše pohodlí a ušetření si řady problémů souvisejících s doinstalováváním externích knihoven bych vám ale k instalaci doporučil právě *Pd-extended*. Rozdíl mezi verzemi není ostatně zas až tak velký – k prvnímu čtvrtletí 2014 je *Pd-vanilla* ve verzi 0.45-4 a *Pd-extended* 0.43-4. Pro specialisty, kteří by chtěli větší přesnost v číselné reprezentaci, jsou tady *double precision Pd*.[36]

Se samotnou instalací stažených *Pd-extended* by neměl být problém. Verze pro Windows (lhostejno zda používáte 64 nebo 32 bitovou verzi systému) je distribuována ve spustitelném instalačním .exe souboru. Pokud instalujete pod MacOSem, stáhněte si balíček, který odpovídá vaší architektuře (Intel/PowerPC) a po odkliknutí .dmg souboru přetáhněte ikonu s názvem Pd-extended.app do složky Aplikace. Pod Linuxem (Debian, Ubuntu, UbuntuStudio) je obvykle možné instalovat *Pd-vanilla* a několik málo knihoven (GEM, zexy) přímo ze systémových repozitářů. *Pd-extended* jsou instalovatelné až po přidání speciálního repozitáře do správce balíčků.[37]

Pokud instalace proběhla v pořádku a podařilo se vám *Pd-extended* spustit, měli byste vidět okno, kterému se říká konzole a vypadá takto:



Kliknutím na DSP zapínáme nebo vypínáme zpracování zvuku. Jestliže máte otevřený nějaký patch, který by měl hrát, a není nic neslyšet, pak bývá často chyba právě v tom, že není zapnuto DSP, nebo je chyba v nastavení audio hardware. VU metr zobrazuje intenzitu vstupních a výstupních signálů. Tlačítkem log vpra-

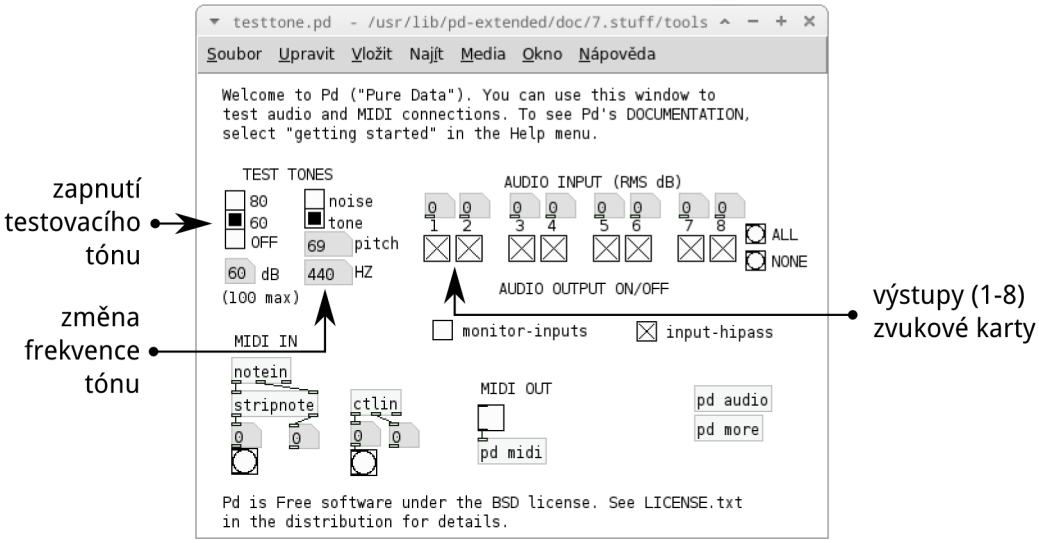
[35] *Puredata.info* [online]. [cit. 2014-03-20]. Dostupné z: http://puredata.info/downloads/pd-extended.

[36] *Katjaas.nl* [online]. [cit. 2014-04-02]. Dostupné z: http://www.katjaas.nl/doubleprecision/doubleprecision.html.

[37] *Puredata.info* [online]. [cit. 2014-03-20]. Dostupné z: http://puredata.info/docs/faq/debian.

vo dole je možné měnit „ukecanost“ konzole – když ji přepnete např. na hodnotu 4, uvidíte v okně podrobný výpis načtených knihoven, přednastavených cest atd. Konzole je základní referenční okno, kam se budete dívat při ladění programu nebo při hledání chyb. Pokud v ní uvidíte nějaký červený nápis, je něco špatně.

Prvním krokem po spuštění Pd bude otestování funkčnosti zvuku. Klikněte na na menu Media (nebo Alt+m) a vyberte položku Test Audio and Midi. Nově objevivší se okno by mělo vypadat takto:

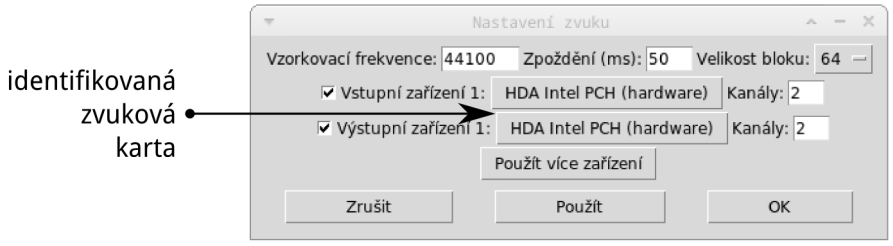


Po zakliknutí testovacího tónu by měl být slyšet spojitý tón komorního A (440 Hz). Klikněte levým tlačítkem myši do tmavšího obdélníku (tzv. číselný box), ve kterém je hodnota 440, držte levé tlačítko myši zmáčknuté a posuňte myš nahoru nebo dolů – tímto způsobem můžete měnit frekvenci. Podobně je možné měnit hlasitost signálu v číselném boxu s označením dB. Zaškrtnutím nebo vyškrtnutím políček s čísly 1 až 8 můžete otestovat, jestli vám hrají dané kanály – standardně jsou funkční pouze první dva kanály. Pokud máte ale připojenu vícekanálovou kartu, pak by tón měl hrát i z dalších kanálů. V tomto testovacím okně lze vyzkoušet, jestli Pd komunikují s připojeným MIDI[38] zařízením (klávesy, pedály, sekvencery aj.).

Pakliže žádný tón neslyšíte nebo je zvuk zkreslený a přerušovaný, je pravděpodobně chyba v nastavení audio hardware. Podívejte se znovu do menu Media. Podle toho, pod jakým operačním systémem jste, uvidíte v menu buď OSS, ALSA, Jack (Linux), portaudio, Jack (MacOS), nebo MMIO, ASIO (Windows). Zde musíte vybrat ovladač, který váš systém podporuje. Pro přesnější nastavení klikněte v menu Media na Nastavení zvuku.

[38]MIDI (Musical Instrument Digital Interface) je komunikační protokol, který umožňuje širokou škálu propojení nástrojových rozhraní – např. MIDI klaviatury s počítačem, na kterém je spuštěný software syntetizér. Protokol zprostředkovává infomace o MIDI kanálu, notě, intenzitě úhozu atd.

V nově otevřeném okně se ujistěte, že je opravdu vybrána zvuková karta, kterou máte instalovánu nebo připojenu. Standardní vzorkovací frekvence by měla být 44100 samplů za vteřinu, velikost bloku 64 nebo 128 vzorků.



Pokud při testovacím tónu slyšíte praskání (dropout), pak zkuste zvýšit hodnotu zpoždění na 100ms, ev. také zvýšit velikost bloku. Když budete Pd chtít používat na zpracování zvuku v reálném čase, pak je nezbytné mít také odpovídající zvukovou kartu. Interní zvukové karty obvykle tento úkol nezvládají příliš dobře – pro naše studijní účely ale budou dostačující.

Aby Pd běžely s prioritou procesování v reálném čase, je třeba v menu Upravit kliknout na položku Předvolby (pokud máte anglické rozhraní, pak je to menu Edit a Preferences) a do kolonky s názvem Startovací praporek (Startup Flags) vložit parametr -rt. Potvrďte kliknutím na OK. Při každém dalším spuštění by Pd měla automaticky běžet v reálném čase.

V praxi se mi osvědčilo používání externí zvukové karty v kombinaci s Linuxovým operačním systémem UbuntuStudio[39] a zvukovým serverem JACK.[40] Pod MacOSem JACK běží, ale neumožňuje směrování MIDI, pod Windows je jeho provoz experimentální a též nemá plnou podporu. Co se stability a možnosti optimálního vyladění systému týká, je doporučeníhodnou volbou Linux. Pod Linuxem navíc odpadají starosti týkající se instalace ovladačů pro hardware (zvukové karty, MIDI kontrolery), jelikož jeho podpora je obvykle již zahrnuta v jádru systému.

Po instalaci a provedení základního nastavení a testů se nyní konečně můžeme začít věnovat studiu.

[39] UbuntuStudio je multimediální odnož distribuce Ubuntu. Její výhoda spočívá v tom, že obsahuje již základní balíčky multimediálních aplikací a má odladěné jádro systému na práci se zvukem a videem v reálném čase. Doporučit lze tzv. LTS verzi, která má dlouhodobou podporu. V prvním kvartálu 2014 vyšla distribuce s pořadovým číslem 14.04. *Ubuntustudio.org* [online]. [cit. 2014-04-02]. Dostupné z: http://ubuntustudio.org/.

[40] JACK je aplikace, která má na starosti správu směrování audio i MIDI signálů v daném systému. V podstatě jde o softwarový rack, s jehož pomocí můžete přepojovat signál z jedné audioaplikace do druhé. *Jackaudio.org* [online]. [cit. 2014-04-02]. Dostupné z: http://jackaudio.org/.

před_tím_než_začneme

Ještě před tím, než se pustíme do samotného studia Pd, bych rád poskytl "návod", jak tuto část rukověti "používat". Látka je strukturována do tří samostatných oddílů. V prvním se budeme zabývat obecnými základy - "gramatikou" a "slovní zásobou" tohoto programovacího jazka. Aspoň povšechná znalost obecných principů zde probraných je předpokladem pro další oddíly. Ve druhém se budeme věnovat zvuku a ve třetím obrazu. Pokud by vás studium první části ve své obecnosti unavovalo, zkuste si "odskočit" do druhého a třetího oddílu, které se věnují již konkrétnějším aplikacím Pd v oblasti zvuku a obrazu.

Rukověť je samozřejmě možné studovat i v tištěné podobě, nicméně její plný potenciál spočívá v tom, že si příklady v ní uvedené vyzkoušíte naživo v Pd. Rukověť je ve zdrojových Pd kódech dostupná na webu vyuka.avu.cz. Soubory prvního oddílu začínají písmenem "A", následovaným pořadovým číslem stránky. Druhý oddíl začíná písmenem "B" a třetí pak písmenem "C". Doporučuji si zdrojové soubory stáhnout a probíranou látku a patche si otestovat, ev. si přímo v patchi zkoušet vytvářet již vlastní malé kódy.

vyuka.avu.cz

Tato rukověť počítá s tím, že používáte verzi Pd-extended, která obsahuje již patřičné externí knihovny a abstrakce. Testována byla na verzi Pd-extended 0.44. Funkčnost všech patchů pro starší verze nebo Pd-vanilla není garantována.

Vždy, když v textu nebo na pravém okaji uvidíte modrý text, jako je tomu v případě druhého odstavce, jde o odkaz na web, který souvisí s probíranou látkou. V Pd je takový text aktivní - můžete na něj kliknout a daný web se vám otevře ve vašem prohlížeči.

Z důvodů typografické "hlaskosti" budeme objekty a zprávy v textu označovat takto:

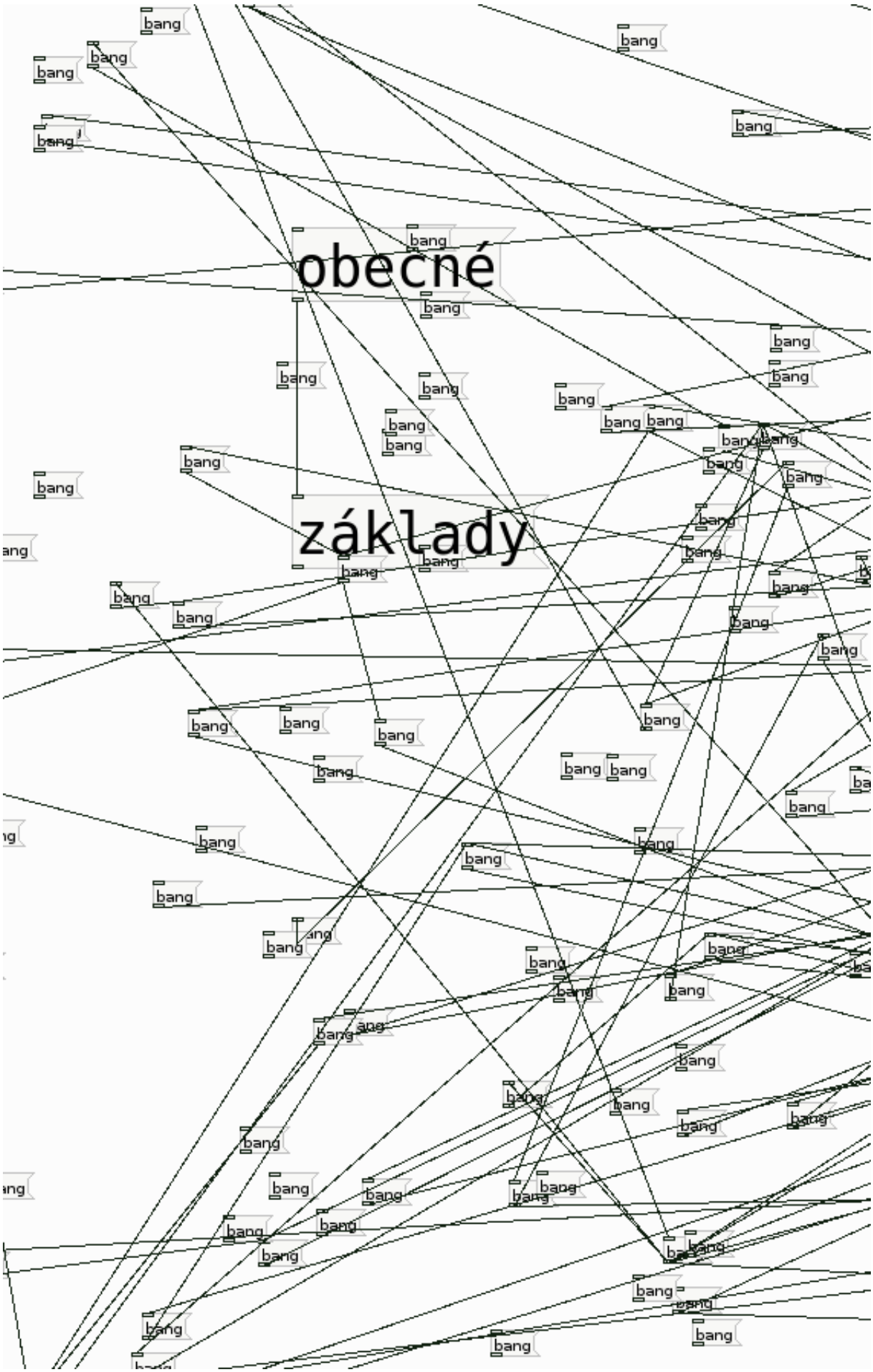
[print] = 

[bang(= 

Ze začátku studia se dívejte často do Pd konzole. Vždy když v patchi uvidíte objekt [print], znamená to, že do Pd konzole se vypisuje nějaká zpráva, jež souvisí s demonstrací a vysvětlením funkčnosti daného kódu. Pokud během studia narazíte na objekt, jehož význam vám nebude znám, mějte na paměti, že se vždy můžete podívat na jeho dokumentaci (viz kapitola "help_i_need_somebody").

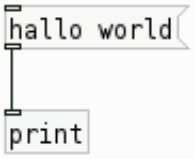
VAROVÁNÍ: Ve druhém oddíle, tj. při práci se zvukem, postupujte vždy tak, že výstup z počítače zapojte do zesilovače, ten ztište a pak spusťte testovací tón (viz předchozí kapitola "Instalace a první spuštění"). Teprve poté si na zesilovači nastavte vám odpovídající hlasitost. Nedoporučuji používat sluchátka, protože při neopatrném a nepoučeném zacházení s Pd může dojít k poškození sluchu.

Přeji vám ve studiu hodně zdaru i zábavy. Nebojte se experimentovat a průběžně zkoušet realizovat vlastní nápady!


obecné
základy

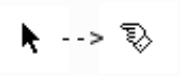
zakladní_stavební_kameny

Nejprve vytvoříme nový projekt: CTRL + n, nebo v menu Soubor -> Nový. Ejhle, máme před sebou "prázdné plátno", neboli patch. Pojmy program, kód a patch budeme v rámci programování v Pd považovat za totožné. Program v Pd sestává z navzájem propojených objektů, které vykonávají specifickou funkci. Ta je obvykle patrná již z jejich názvu. Např. objekt [print] vypisuje zprávy do konzole.



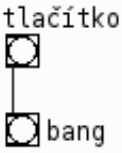
EDITAČNÍ_/_PERFORMAČNÍ_MÓD

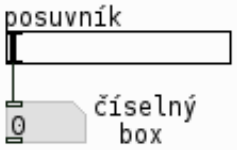
Abychom praxi programování v Pd porozuměli, je třeba se hned ze začátku naučit rozlišovat dva základní módy, ve kterých Pd pracují: editační mód a performační mód. Do editačního módu se přepneme pomocí klávesové zkratky CTRL + e (Cmd + e na Mac OSu), nebo v menu Upravit (Edit) vybereme položku Edit Mode. Kurzor by se měl ze šipky změnit na ruku.



V editačním módu se odehrává vlastní proces programování: vkládání, mazání, přesouvání a propojování objektů. Objekt, který chceme editovat, naklikneme levým tlačítkem myši (dále jen LTM) - měl by být poté zvýrazněný modře.

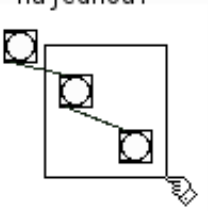
Do performačního módu se přepneme opětovým stiskem CTRL + e a kurzor se opět změní na šipku. V performačním módu nelze psát kód, objekty jsou "zamčené" a není možné s nimi manipulovat. V performačním módu můžeme na určité části kódu (číselný box, posuvník, bang, vypínač atd.) "hrát". Zkuste v performačním módu kliknout levou myší na posuvník, držte LTM a pohněte myší doleva nebo doprava - uvidíte, jak se mění číselná hodnota. S objektem, který obsahuje číslo, to funguje stejně, jen myší hýbeme nahoru a dolů. Pokud držíme SHIFT, pak se pohybujeme v jemnější škále. Zkuste kliknout také na tlačítko, kterému se v Pd říká bang.





Znovu se přepneme do editačního módu (CTRL + e) a klikneme na posuvník. Když držíme LTM, je možné s objektem po patchi pohybovat. Pokud v editačním módu kamkoliv klikneme a držíme LTM, pak můžeme tahem vytvořit obdélník, kterým označíme více objektů najednou. Klávesa BACKSPACE nebo DEL označený/é objekt/y smaže. Dále fungují klasické klávesové zkratky:

označení více objektů najednou:



CTRL + c - zkopíruje označený objekt do schránky C
CTRL + v - vloží objekt/y ze schránky
CTRL + x - vyjme objekt/y a uloží do schránky
CTRL + d - zduplikuje označený/é objekt/y

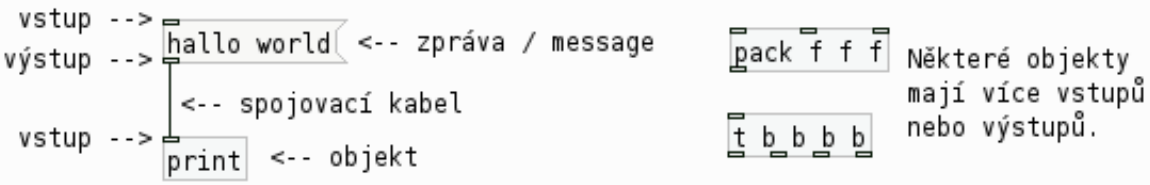
CTRL + z - undo - pozor, Pd si pamatují jen jeden krok zpět!
CTRL + SHIFT + z - redo

Samotné objekty vkládáme přes menu Vložit (Put) nebo klávesovou zkratkou CTRL + 1..5. Doporučuji přepnout na anglickou klávesnici - vyhnete se tak krkolomné kombinaci CTRL + SHIFT + číslo. Také pozor na CAPSLOCK. Pokud se nemůžete přepnout do editačního módu, může za to obvykle tato klávesa.

Základních stavebních kamenů, když nepočítáme grafické prvky, je v Pd pět: objekty, zprávy, čísla, symboly a komentáře.

OBECNÁ_STRUKTURA_KÓDU

V Pd programujeme tím, že vytváříme vizuální struktury sestavené z pospojovaných objektů, které obvykle mají vstupy (inlet) a výstupy (outlet).



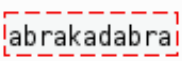
Když kliknete LTM v performačním módu na zprávu [hallo world(, tato "doteče" po kabelu až do objektu print. Objekt print tiskne to, co mu "přiteče" na vstup do konzole.

OBJEKT

Když zmáčkneme CTRL + 1, objeví se v patchi na místě kurzoru prázdný box vykreslený čerchovanou čarou, který myší můžeme libovolně přemisťovat. Pokud jsme byli v performačním módu, vložení objektu nás vždy automaticky přepne do editačního módu. Kliknutím LTM se box ukotví, je v něm viditelný kurzor a my do něj můžeme psát. Po druhém kliknutí LTM mimo box se Pd pokusí námi napsaný text interpretovat. Pokud Pd dané slovo "znají", pak vytvoří funkční objekt. Tak je tomu v případě zapsání slova print. Správě vytvořený objekt je vykreslený spojitou linkou.

--> správně vytvořený objekt

Pokud do boxu ale napíšeme slovo, které Pd neznají a neví, jak je interpretovat, dopadne to takto:



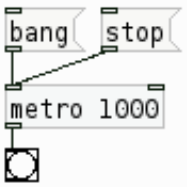
--> špatně vytvořený / neznámý objekt

O tom, že si Pd s interpretací výše uvedeného slova nevěděla rady, jsme navíc informováni i v konzoli. Podívejte se do ní. Měl by tam být červený nápis: "abrakadabra ... couldn't create". Pokud otevřete cizí patch a Pd konzole vypisuje tuto hlášku, nebo vidíte červeně lemované objekty, obvykle to znamená, že vám chybí některá z knihoven, nebo jsou špatně nastavené cesty k abstrakcím (viz později).

Jaká slova do prázdného boxu můžeme psát, a tím jej specifikovat a dát mu funkci, nyní ještě nevíme. S postupem času se seznámíme s celou řadou slov / příkazů, která Pd znají.

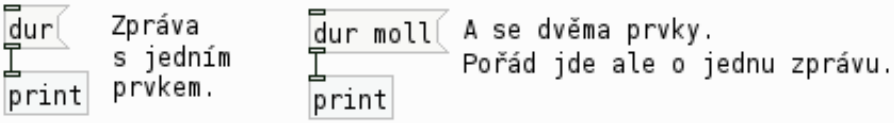
ZPRÁVA_/_MESSAGE

Objekty v Pd obvykle přijmají a posílají dál zprávy. Zpráva může být jméno souboru, které posílám do nějakého objektu, od kterého očekáváme že s daným jménem bude umět něco udělat. Nebo zprávou může být příkaz, kterým objektu "říkám", co má začít dělat. Např. objekt metro čeká na vstupu zprávu [bang(, která jej zapne.

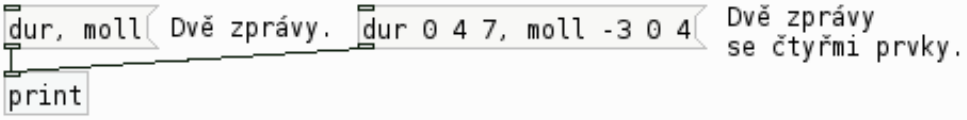


Prázdnou zprávu (message box) do patche vložíme s pomocí klávesové zkratky CTRL + 2 (nebo složitěji menu Vložit -> Zpráva) a poznáme ji tak, že na rozdíl od objektu má "vykrojenou" pravou stranu. Stejně jako u objektu: první klik message box ukotví, pak můžeme psát a druhý klik mimo zprávu ji definitivě vytvoří.

Zprávy mohou nést jeden nebo více prvků, oddělených mezerou:

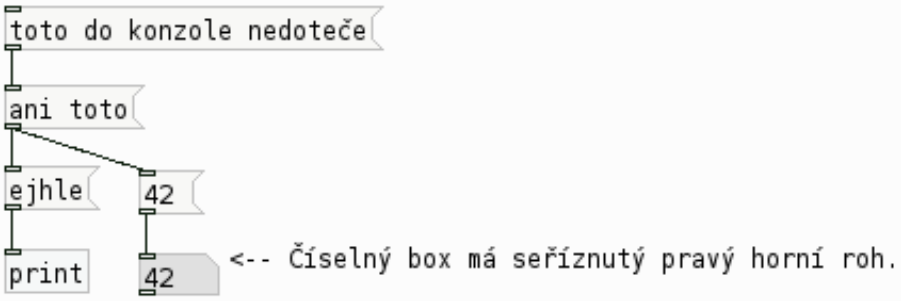


V jednom message boxu ale také můžeme mít více zpráv, pak jsou jednotlivé prvky odděleny čárkou. Ve zprávě lze kombinovat čísla se symboly. Do vstupu jednoho objektu lze směřovat více zpráv:



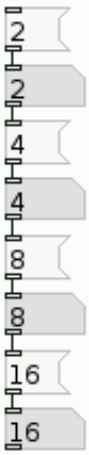
Podívejte se na rozdíl výstupu v konzoli. Zatímco pokud zpráva obsahuje jeden a více prvků neoddělených čárkou, tiskne se vše na jeden řádek - pořád jde o jednu zprávu. Pokud jsou ale odděleny čárkou, pak je na výstupu každá zpráva na novém řádku - jsou to dvě zprávy. Vidíme také, že prvky jsou v konzoli vytišteny ve stejném pořadí jako ve zprávě - tj. zleva doprava.

K vlastnosti zpráv patří to, že uchovávají svůj obsah, můžeme do nich uložit informace o frekvenci tónu, barvě RGB kanálu atd. Zpráva je prostředek komunikace objektů mezi sebou. Kliknutím na zprávu v performačním módu zprávu inicializujeme - "posíláme ji po kabelu". Další vlastnost zpráv spočívá v tom, že pokud na svém vstupu něco obdrží, předchozí zprávu pozastaví a dál pošle jen svůj obsah:

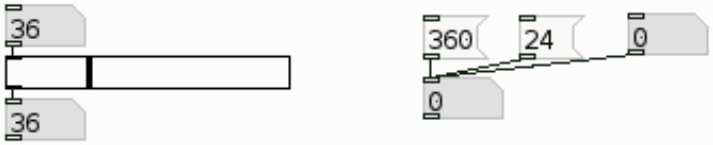


ČÍSELNÝ_BOX_-_NUMBER_BOX

Další stavební prvek, který patche obvykle obsahují, je tzv. číselný box (klávesová zkratka CTRL + 3). Již jsme se s ním seznámili při prvním spuštění Pd. Jeho funkce spočívá buď v zobrazení nějakého čísla nebo v možnosti vložení hodnoty. Pokud číselnému boxu na vstup doteče nějaké číslo, zobrazí je a pošle dál. Na rozdíl od zprávy si číselný box neuchovává svůj obsah - po zavření a otevření patche budou všechny hodnoty, co jsme v nich nastavili, ztraceny.

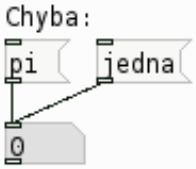


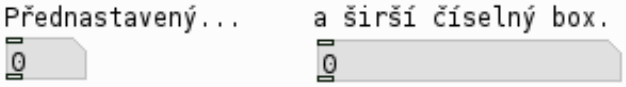
Klikněte přímo do číselného boxu LTM, podržte je a hýbejte myší nahoru/dolů. Nastavíte tím jeho hodnotu - výstupem jsou vždy celá čísla. Pokud držíte navíc SHIFT, škála je jemnější a výstupem jsou čísla s desetinnou čárkou.

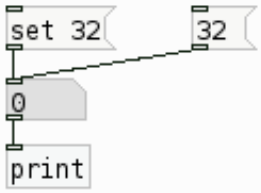


Číselný box lze inicializovat také přesným vložením hodnoty
z klávesnice: klikneme do něj LTM, napíšeme hodnotu
a potvrdíme klávesou Enter.

Přednastavený číselný box umí zobrazit hodnoty v rozmezí
-9999 a 9999.Pokud chceme vidět vyšší hodnoty, pak na něj
klikneme pravým tlačítkem myši (dále jen PTM) a v nově
otevřeném menu vybereme Vlastnosti (Properties).
V otevřeném okně pak změníme jeho šířku, ev. ho můžeme
pojmenovat. Potvrdíme OK.





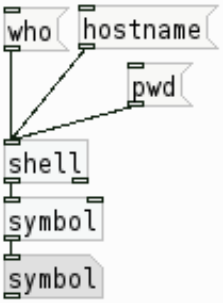


Zpráva [set 32(pouze nastaví hodnotu číselného boxu,
ale nepošle ji ven z jeho výstupu. Kdežto zpráva
32 jím "proteče" hned ven. Klikněte a podívejte se
do konzole.

Poznámka pro programátory: Pd uchovávají všechny hodnoty
jako float, takže i když 1 vypadá jako integer, ve
skutečnosti je to 1.0000000.

SYMBOL_BOX

CTRL + 4 do patche vloží symbol box. Vypadá takto: symbol
Jeho funkce je podobná jako u číselného boxu, jen slouží
pro zobrazení a vkládání libovolného textu (včetně čísel).
S jeho pomocí můžeme dynamicky zadávat jména souborů, atd.



Pod Linuxem (a MacOSem) je např. možné s pomocí objektu
[shell], kterému posíláme zprávou systémové příkazy, do
symbol boxu dostat jméno uživatele, počítače, aktuální
adresář atd.

KOMENTÁŘ

Posledním základním kamenem a velmi užitečným prvkem,
kterým je ostatně psán i tento text je komentář. Vložíme
jej klávesovou zkratkou CTRL + 5.

Patří k dobré programátorské praxi, že se kód komentuje.
Jednak je to užitečné pro jiné programátory, kteří váš
patch díky komentářům rychleji pochopí a jednak pro vás
samotné - často se prostě stává, že pokud se ke svému
neokomentovanému patchi vrátíte po delší době, tak již
nebudete vědět, co která část kódu dělá.

CVIČENÍ:

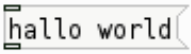
- zkuste v patchi vytvořit objekty print, metro, float
- označte objekty, které jste vytvořili, zduplikujte je
  a přemisťte je
- vytvořte číselný box a zkuste v něm měnit hodnotu
- vytvořte zprávu a vložte do ní číselnou hodnotu
- zkuste patch uložit (menu Soubor -> Uložit, nebo CTRL + s),
  zavřete ho a znovu otevřte (Soubor -> Otevřít, CTRL + o)
  Co se stalo s obsahem číselného boxu a zprávy?

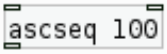
píšeme_kód_/_taháme_kabely_/_hallo_world

Zatím jsme si pouze ukázali, jak vkládat do patche základní prvky, ze kterých kód sestává. Ty ale samy o sobě nic nedělají. Ve vizuálním programování, stejně jako v nastavení modulárního syntetizátoru, je důležité to, jak jsou jednotlivé prvky propojeny - to, kam jaký signál putuje. Jak propojení vytvářet si ukážeme na následujícím příkladu.

Tradiční součástí učebnic programování bývá prográmek Hello World, který vypíše pozdrav. Vzhledem k tomu, že jsou ale Pd jazyk sloužící primárně k procesování signálů, trochu si ho přizpůsobíme.

Nejprve vytvoříme tři objekty (CTRL + 1), do prvního vepíšeme "ascseq 100", do druhého "osc~" a do třetího "dac~". Následně vytvoříme ještě jednu zprávu a vepíšeme do ní hallo world. Objekty rozmístíme v ploše takto:





<-- nezapomeňte napsat i mezeru

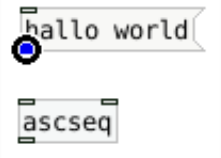
Objekty osc~ a dac~ obsahují znak vlnovky: "~". Tento znak mají všechny objekty, které mají co do činění se zvukem.

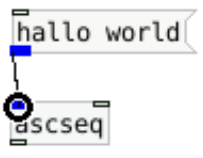
Pokud znak "~" marně hledáte,podívejte se do levého horního rohu klávesnice. Přepněte klávesnici na anglickou, pak SHIFT + klávesa, na které je vlnovka.

Nyní vytvoříme vlastní propojení. Objekty mají, jak jsme si řekli, vstupy (inlet) a výstupy (outlet), jež jsou situovány na horní a spodní straně objektu. Propojujeme vždy výstupy do vstupů, jinak to nejde.

Přepneme se do editačního módu a kurzorem najedeme na výstup zprávy [hallo world(- kurzor by se měl změnit na kolečko, klikneme LTM, podržíme a přetáhneme kurzor na levý vstup objektu acseq - měli bychom vidět kabel, který vychází ze zprávy a sleduje kurzor.

Když kabel natáhneme na vstup, měl by se kurzor opět změnit na kolečko - v tu chvíli můžeme pustit LTM a propojení objektů by mělo být vytvořeno.

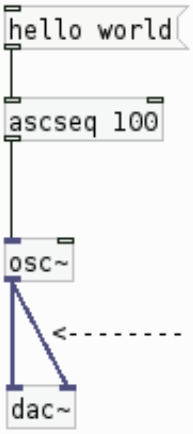




Zapojení samozřejmě nejsou náhodná, s postupujícím studiem se seznámíme s řadou objektů i s tím, jaký typ zprávy či signálu na tom kterém vstupu očekávají a jaký typ zprávy nebo signálu vysílají.

Pokud jsme vytvořili nějaké spojení, které chceme zrušit, pak v editačním módu klikneme LTM na kabel - měl by zmodrat - a pak klávesou BACKSPACE nebo DEL spojení smažeme.

Propojíme i zbytek objektů, abychom obdrželi následující strukturu:



Ve zprávě posíláme nějaké znaky.

Objekt [ascseq] je součástí knihovny cxc, která je již součástí Pd-extended a převádí znaky, které přijme na ASCII hodnotu, ty pak vysílá jediným výstupem dál v intervalech 100 ms.

Oscilátor na levém vstupu očekává číselnou hodnotu, ta odpovídá frekvenci, kterou pak hraje.

<-------- Kabely, které vedou zvukový signál, jsou silnější.

"Digitálně analogový konvertor" pošle zvukový signál k interpretaci zvukové kartě.

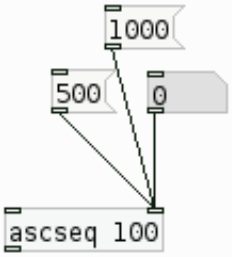
Vytvoříme také ještě dvě zprávy, do kterých vepíšeme text ";pd dsp 1" a ";pd dsp 0". Jde o tzv. interní zprávy a poznáme je tak, že začínají středníkem. Prozatím nám bude stačit, když si zapamatujeme, že jsou to zprávy, které "hovoří" přímo k Pd. S jejich pomocí lze psát metaprogramy (programy, které vytvářejí programy) - o tom ale později.

<-- Tyto zprávy zapnou / vypnou zvuk. Nemusíme klikat na DSP v konzoli.

Pakliže jste vytvořili a propojili správně všechny objekty, zapněte DSP a klikněte na zprávu [hello world(. Měli byste slyšet krátký kolísavý zvuk a pak jednolitou frekvenci (je to 100 Hz, protože poslední znak zprávy je písmeno "h" a to má v dekadické soustavě ASCII hodnotu 100).

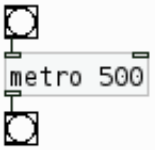
ARGUMENTY_OBJEKTŮ

Objekt [ascseq] má tzv. argument - číslo, které jej specifikuje a inicializuje. Podobně jsme to viděli již u objektu metro. V případě objektu [ascseq] je možné zadat jeden argument, který specifikuje to, jak rychle budou z jeho výstupu vysílány ASCII hodnoty. 100 znamená 100 milisekund, takže 1000 je jedna vteřina.



Když bychom do pravého vstupu objektu [ascseq] připojili výstup z číselného boxu nebo zprávy s předdefinovanými hodnotami v milisekundách, pak bychom mohli rychlost výstupu dynamicky měnit.

Podobně tak objekt metro, který posílá bang (viz později), lze specifikovat jedním argumentem. Ten určuje délku prodlevy intervalů.



Metro(nom) tikající 2x za vteřinu.

Některé objekty nemají žádné argumenty a některé jich mohou mít víc (např. zvuková obálka, posun po osách X, Y, Z atd.)

CVIČENÍ:

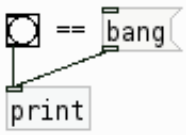
- změňte argument objektu [ascsq] na 500
- změňte objekt [osc~] na [phasor~]
- připojte na výstup objektu [ascseq] číselný box
- zprávu [hello world(změňte na nějakou vlastní
- klikněte v performačním módu na změněnou zprávu

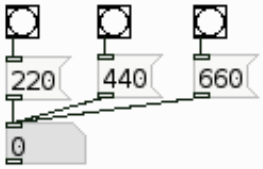
tlačítka_posuvníky_vypínače

Pd patch netvoří jen prvky, které jsme si ukázali. Představa, že v Pd např. postavíme nějaký nástroj a jeho rozhraní by byla změť objektů, zpráv a kabelů, je poněkud odstrašující. Proto Pd disponují sadou grafických ovládacích prvků jako tlačítko (Bang), vypínač (Toggle), posuvník (Hslider), přepínač (Hradio) atd. Najdeme je opět v menu Vložit (Put).

BANG

CTRL + SHIFT + b vloží do patche bang, což je prosté tlačítko. Jeho funkcí je inicializovat zprávy nebo objekty. Kliknutím na bang jako bychom poslali impuls nebo řekli "udělej to a to".

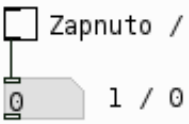


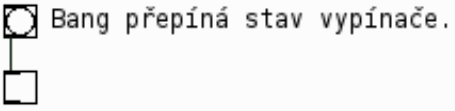


Inicializuje číselný box na danou hodnotu ze zprávy.

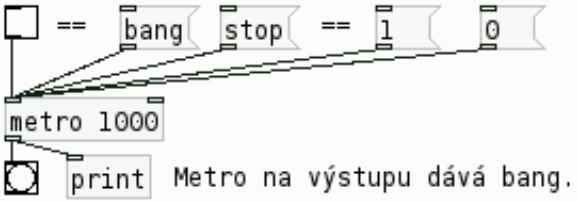
TOGGLE_-_VYPÍNAČ

Toggle je vypínač (klávesová zkratka CTRL + SHIFT + t). Na výstupu dává hodnotu 1 nebo 0 a pokud do něj pošleme bang, přepne ho do druhého stavu.





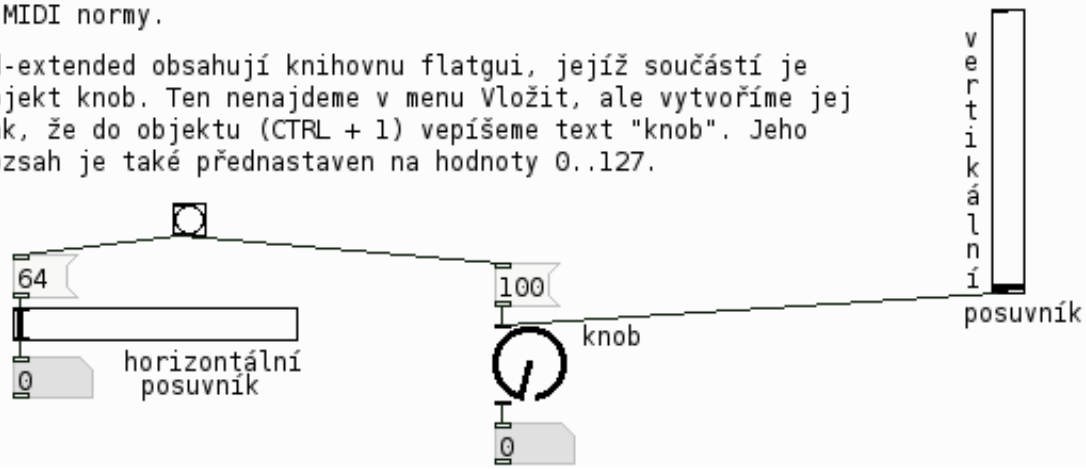
Vypínačem lze ovládat např. objekt [metro], který na levém vstupu čeká na zprávu [bang(/ [stop(nebo na hodnotu 1 / 0.



VSLIDER_-_POSUVNÍK

Ovládací prvek, který znáte např. z audio mixpultu, je posuvník. V Pd máme k dispozici dvě varianty: horizontální (CTRL + SHIFT + h) a vertikální (CTRL + SHIFT + v). Na výstupu má přednastaveny hodnoty 0 až 127, což vychází z MIDI normy.

Pd-extended obsahují knihovnu flatgui, jejíž součástí je objekt knob. Ten nenajdeme v menu Vložit, ale vytvoříme jej tak, že do objektu (CTRL + 1) vepíšeme text "knob". Jeho rozsah je také přednastaven na hodnoty 0..127.



HRADIO_-_PŘEPÍNAČ

Pod touto záhadnou zkratkou se skrývá horizontální přepínač (CTRL + SHIFT + i), který má podobně jako posuvník i vertikální variantu (a CTRL + SHIFT + d). Na výstupu ma přednastaveny hodnoty 0 až 7. Využijeme ho např. k přepínání presetů - stavů syntetizátoru, nebo systémů, které postavíme.

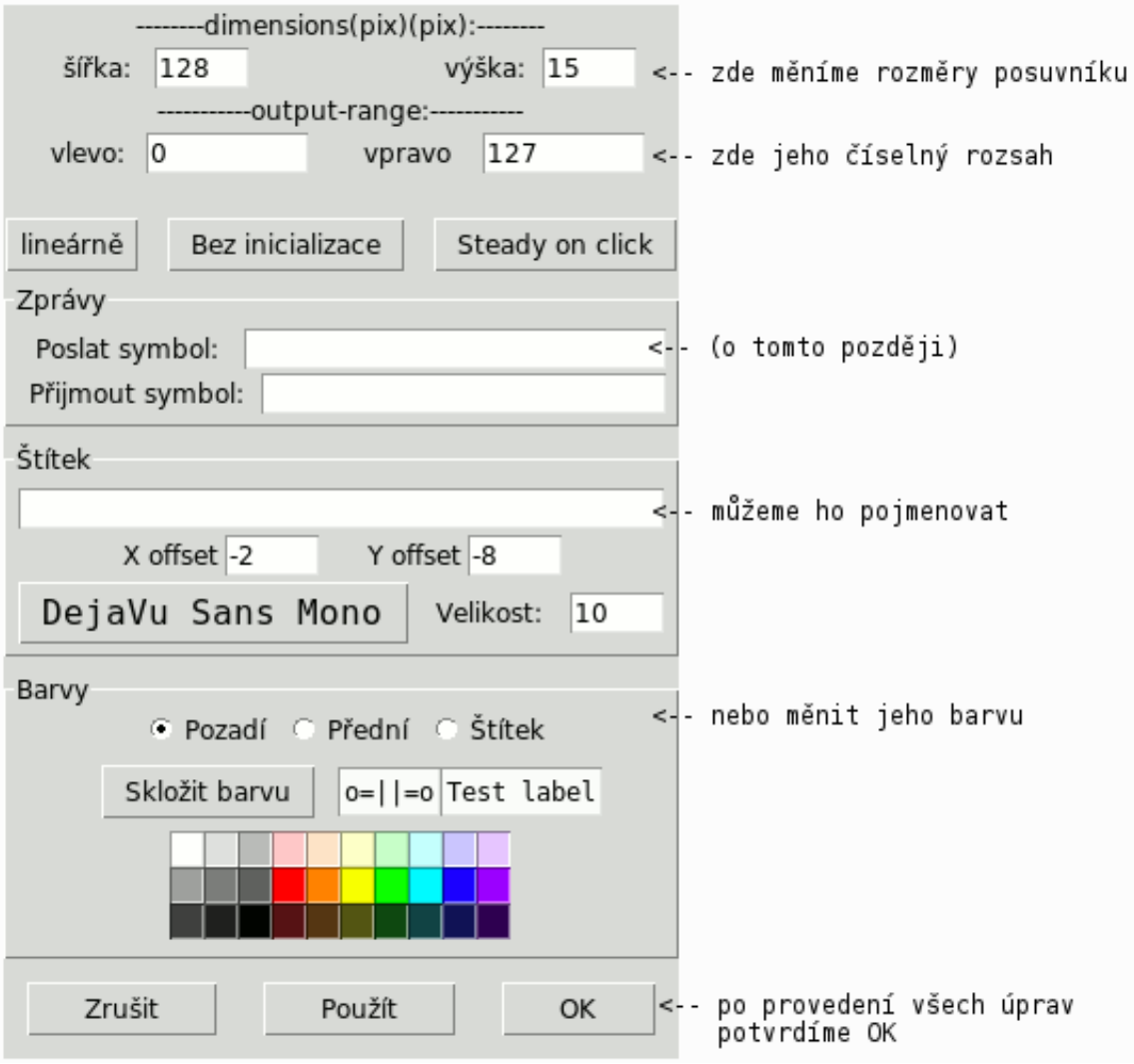
PLÁTNO_-_CANVAS

Prázdné plátno, neboli canvas, se skrývá pod klávesovou zkratkou CTRL + SHIFT + C. Využijeme je zejména při pokročilejší práci s grafickým uživatelským rozhraním (GUI). Později si ukážeme, jak vytvořit uživatelsky přístupné a jednoduché grafické rozhraní.

plátno

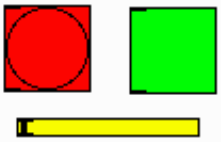
PŘIZPŮSOBENÍ

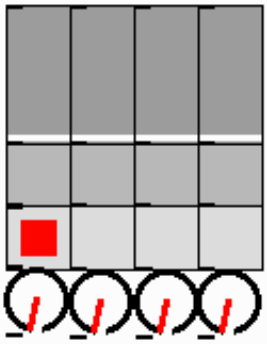
Všechny zde vyjmenované grafické prvky je možné přizpůsobovat - měnit jejich velikost, barvu, číselný rozsah atd. Stačí na daný prvek, např. na posuvník, kliknout PTM a z menu vybrat položku Vlastnosti (Properties). V případě plátna je na pravý klik aktivní pouze jeho levý horní roh. Otevřít by se mělo toto okno:



CVIČENÍ:

Pokuste se vytvořit několik grafických ovládacích prvků a přizpůsobit je podobně jako vidíte níž. Tip: pro jemnější a přesnější manipulaci s označeným objektem nemusíte používat myš, ale kurzorové šipky na klávesnici.





Vidíte tedy, že Pd nabízí velkou míru svobody v tom, jak jednotlivé prvky grafického uživatelského rozhraní rozvrhneme. Můžeme si postavit GUI, které bude vyhovovat přesně našim potřebám.

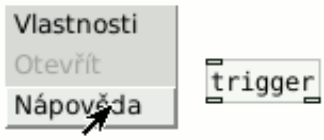
Podobnou míru svobody jako je tomu v navrhování grafického "povrchu" nám Pd dávají pochopitelně i v návrhu kódu. To uvidíme později...

- vytvořte objekt [dac~], dva objekty [osc~] a dva posuvníky
- nastavte rozsah prvního posuvníku na 110 až 220 a druhého na 220 až 440
- propojte výstup prvního posuvníku s levým vstupem prvního [osc~] a výstup druhého posuvníku s levým vstupem druhého [osc~]
- propojte výstup prvního osc~ s levým vstupem dac~ a výstup druhého osc~ s pravým vstupem dac~
- pohrajte si s posuvníky

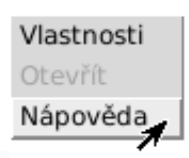
help_i_need_somebody

Na předchozích stránkách jsme hovořili o objektech, jejich atributech, o tom, jaké mohou nebo nemohou přijmat zprávy atd. S postupem času a probrané látky toho v tomto ohledu opravdu nemálo prozkoumáme, nicméně zvědavcům a samostudia-chtivým nyní můžeme doporučit ke studiu dokumentaci a referenční soubory.

Když na jakýkoliv objekt v Pd kliknete PTM a z menu vyberete položku Nápověda (Help), otevře se okno s podrobným popisem funkce daného objektu, včetně popisu vstupů a výstupů.



Kliknutím PTM na prázdnou bílou plochu a zvolením Nápovědy z menu obdržíte seznam základních "slov", která Pd znají - přehledně roztříděných do sekcí podle funkce: čas, MIDI, matematické operace, audio objekty atd.



K další detailní nápovědě a materiálům vhodným k samostudiu se dostanete (kupodivu) přes menu Nápověda -> Pd Help Browser (klávesová zkratka CTRL + b). V první kolonce nově otevřeného okna naklikněte položku "Pure Data" a ve druhé "5.reference". Ve třetí se zobrazí seznam referenčních souborů k jednotlivým objektům. Ke studiu jsou dobré soubory ze sekce all_about_...

V Pd Help Browseru můžete také prozkoumat dokumentaci ke všem externím knihovnám (první kolonka), které jsou v Pd-extended obsaženy, a oficiální Millerův úvod.

CVIČENÍ:

- prozkoumejte nápovědu k objektům [float], [trigger] a [osc~]

dataflow

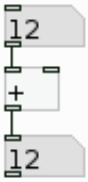
V této kapitole se blíž seznámíme s problematikou dataflow programování. Těm, kteří již mají s programováním nějakou zkušenost, bude připadat asi dost podivná a najdou v ní několik příležitostí pro ohnutí své programátorské mysli. Netknutí čtenáři pak, právě proto, že do světa programování vstupují skrze dataflow paradigma, můžou být nehezky poznamenáni pro budoucnost. Nicméně, právě díky své jinakosti má dataflow paradigma i svůj půvab. Pokud vám zde prezentovaná látka přijde nejasná, nic si z toho nedělejte a pokračujte ve studiu dál. V budoucnu se k této látce můžete vrátit.

JAK_DATA_TEČOU_I

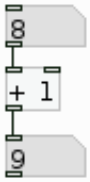
Dřív jsme programování v Pd přirovnali k práci na vodním díle. Voda jsou data a signály. Přehrady a čističky jsou pak různé objekty, které signál mění.

Představíme si jednoduché aritmetické operace:

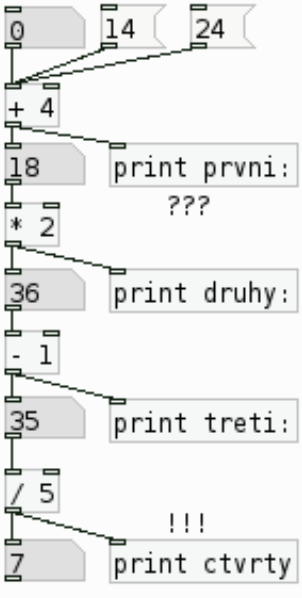
Bez atributů přičítáme 0, odečítáme 0 atd.



Aritmetické objekty specifikované číselným atributem: přičítáme 4, odečítáme 1 atd.



Také jsme již řekli, že k charakteristice dataflow programovacích jazyků patří, že změna nějaké proměnné v kódu vede k bezprostřednímu přepočítání hodnot s touto proměnnou svázaných. Jak aritmetické operace tečou?



Podívejme se podrobně na tento kód. Hodnota z číselného boxu nebo zprávy nejprve putuje do objektu [+ 4], ze kterého pak putuje do dalšího číselného boxu, kde se ukazuje mezivýsledek, a také do objektu print. A pak dolů dál...

Když změníme hodnotu v číselném boxu, vidíme změnu také bezprostředně ve všech ostatních číselných boxech. Jak jsou ale výsledky vyhodnocovány v čase? Podívejme se na ně pod drobnohledem. Budeme debuggovat s pomocí objektu [print], který vypisuje mezivýsledky do konzole. Čeká nás malé překvapení...

Data "jako by" tečou shora dolů, a tak bychom intuitivně řekli, že vyhodnoceno bude nejprve přičtení 4 a že se nám na výpisu v konzoli objeví jako první - tak to ale není. Místo toho se v konzoli jako první objeví vyhodnocený výsledek, který je v řetězci operací úplně nejníž!

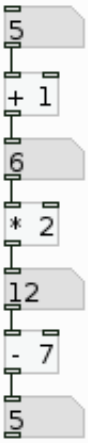
Této podivné vlastnosti se říká "DEPTH FIRST". Pd se pokoušejí v dané struktuře jít vždy až na nejhlubší úroveň, tu vyhodnotí, a teprve pak postupují ve vyhodnocování směrem nahoru.

Vypadá to, že data tečou shora dolů, ale vyhodnocována jsou odspoda nahoru.

"TYPOGRAFICKÁ"_KONVENCE

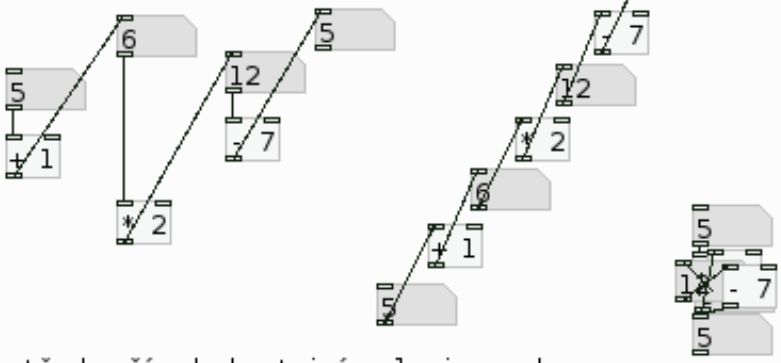
V dataflow jazycích je důležité propojení objektů. To, jak kód vypadá vizuálně, je v podstatě lhostejné. Přesto je ale dobré dodržovat určité konvence, které vám usnadní jeho čtení.

Již samotné propojování objektů a to, že výstupy zapojujeme do vstupů, nás intuitivně vede k organizaci kódu od shora dolů. Proti této intuici můžeme jít a rozmístit kód obráceně, na přeskáčku, nebo chaoticky. Výsledná podoba ale nebude tak elegantní a srozumitelná.



ok

Takto prosím ne:

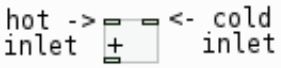


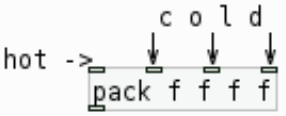
Výsledek bude ve všech třech případech stejný, ale je snad na první pohled patrné, která z ukázek kódu se dobře čte. Pokud chcete, aby váš patch byl dobře čitelný, můžete se podívat i na další konvence psaní zde: PD programming conventions.

AKTIVNÍ_A_PASIVNÍ_VSTUPY

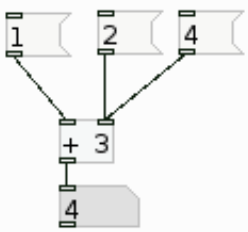
Dalším důležitým bodem, bez kterého se programování v Pd neobejde a který souvisí s tím, jak data tečou a jak jsou vyhodnocována, je rozlišení aktivních a pasivních vstupů.

Objekty, jež přijímají a zpracovávají zprávy, mají jeden tzv. aktivní vstup (hot inlet), který je situovaný úplně vlevo, a žádný nebo více tzv. pasivních vstupů (cold inlet). Např. objekty s aritmetickými operacemi, které jsme si ukázali, mají dva vstupy: jeden aktivní a jeden pasivní. Objekt [pack f f f f] (se kterým se seznámíme později) má jeden aktivní a tři pasivní vstupy.



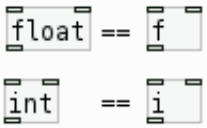


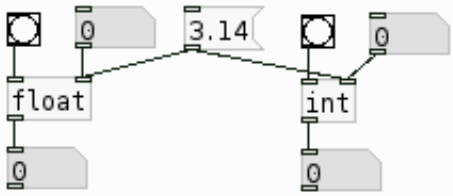
V čem spočívá rozdíl? Zpráva poslaná na aktivní vstup je bezprostředně zpracována, vyhodnocena a poslána objektem dál. Pasivní vstupy slouží k aktualizaci vnitřního nastavení objektu (změna argumentů). Zprávy přijaté na pasivní vstup objekt aktualizují, ale neposílají nic na výstup.



Když klikneme na zprávu [1(, uvidíme v číselném boxu hned výsledek, protože zpráva s jedničkou je zapojena na aktivní vstup. Po kliknutí na zprávu [2(nebo [4(do číselného boxu nic nedoteče, protože jsou tyto zprávy zapojeny do pasivního vstupu. Změní ale atribut objektu, takže již nebudeme přičítat 3, ale 2 nebo 4.

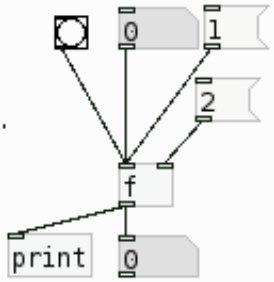
Podobně je tomu i u objektů, které umí přijmout hodnotu a podržet ji, dokud na aktivním vstupu neobdrží bang. Jsou to objekty [float] a [int] - přirovnat bychom je mohli k proměnným z klasických programovacích jazyků.



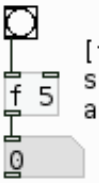


Float umí podržet hodnotu s desetinnou čárkou, int pouze celá čísla. Číslo, které těmto objektům přiteče na pasivní vstup, je v nich "uschováno" a ven je z nich dostaneme, až když jim do aktivního vstupu pošleme bang.

Když do aktivního vstupu objektu [float] posíláme hodnotu, ať už z číselného boxu nebo zprávy, tato jím hned proteče dál. Zároveň je v něm ale dál "uschována" a může být buď změněna zprávou z pasivního vstupu nebo poslána ven bangem.



Možná si kladete otázku, k čemu jsou tyto obskurnosti a nesrozumitelné konvence dobré? Rád bych předeslal, že právě díky těmto vlastnostem budeme moci postavit např. počítadlo, které je základem řady dalších konstrukcí. Bez znalosti těchto podivných základů bychom nemohli postavit ani sekvencer...



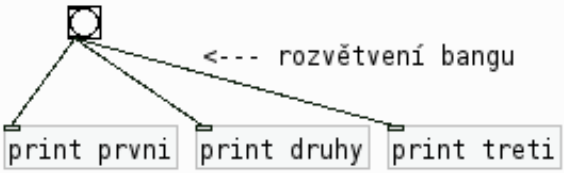
[float] specifikovaný argumentem

CVIČENÍ:

- mějme k dispozici 3x 0 , 1x * , + a print
- pokuste se z těchto objektů sestavit patch pro výpočet formule x = (a * b) + c. X tiskněte do konzole
- v jakém pořadí je třeba číselné boxy inicializovat, aby byl výsledek hned viditelný v konzoli?

NEJASNOSTI_V_TOKU_DAT

Při psaní kódu v dataflow se vám často stane, že potřebujete poslat danou zprávu do více objektů. Posílání zpráv se může větvit, čímž dochází k malé komplikaci: jak zachovat posloupnost provádění kódu?



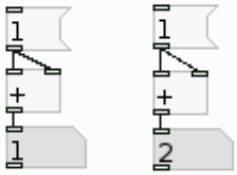
Do konzole se nejprve vypíše "druhý", poté "třetí" a teprve naposledy "první" - to je dáno posloupností, ve která byla vytvořena jednotlivá spojení - nejprve byl bang spojen s [print druhy], poté s [print treti] a teprve nakonec s [print prvni].

CVIČENÍ:

- pokuste se předrátovat tento příklad tak, aby vytiskl do konzole pořadí první, druhý, třetí

I když je tímto způsobem možné také dosánout provadění kódu v pořadí, které vyžadujeme, je třeba si uvědomit, že takto konstruovaný kód je pro druhého nečitelný! Pokud se někdo druhý na patch dívá, na první pohled nepozná v jakém pořadí jsou dráty zapojeny. Může na to sice přijít, ale takový proces je v komplikovaných kódech zdlouhavý.

Další problém, který vyvstává z možnosti posílat zprávu kabelem na více míst a který je příčinou řady nevysvětlitelných chyb v kódu, může vypadat např. takto:



Vizuálně oba kódy vypadají stejně a čekali bychom, že se budou stejně i chovat - tedy že budou sčítat dvě stejné hodnoty. Výsledek je ale přesto jiný, což je způsobeno různým pořadím v zadrátování, které na první pohled nepoznáme.

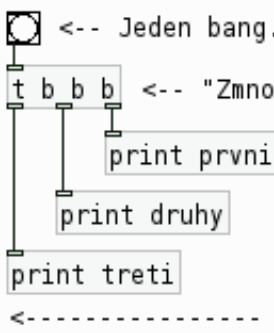
Rozeberme si tok dat detailně:
V prvním případě zpráva [1(do objektu [+] teče nejprve do levého (hot) vstupu a pak teprve do pravého (cold). Při změně hodnoty nebude výsledek nikdy správný, protože tento kód nejprve sčítá a pak mění atribut. V druhém případě byly dráty zapojeny naopak, takže se nejprve mění atribut a pak sčítá. Výsledek bude vždy n + n.

JAK_DATA_TEČOU_II_-_USMĚRŇOVÁNÍ

Jak zmíněným problémům a nejasnostem s tokem dat po více větvích předcházet? Odpovědí je objekt [trigger], s jehož pomocí se dá přesně definovat, jak mají akce postupně probíhat. [trigger] "rozmnoží" jednu zprávu tolikrát, kolik mu přidělíme atributů, a od počtu atributů se odvíjí i počet výstupů, ze kterých pak zprávu posílá dál.

trigger

Zprávy z objektu trigger se vyhodnocují zprava doleva!

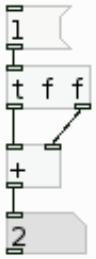


V konzoli vidíme pořadí provádění akcí:
bang teče 3x zprava doleva.
První se inicializuje ten nejvíce vpravo, pak prostřední a jako poslední ten nejvíce vlevo.

t b b == trigger bang bang

Správně, tj. čitelně by předchozí sčítací patch měl být zapsán tako:

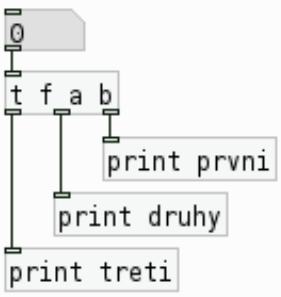
t f f == trigger float float



<-- jedna zpráva typu float

<-- Zmnožíme zprávu z číselného boxu 2x (je to zpráva typu float), nejprve ji pošleme do pasivního (cold) pravého vstupu, tím změníme atribut - nastavíme hodnotu, která se bude přičítat, a pak teprve pošleme tu samou zprávu do aktivního (hot) levého vstupu, ke které bude přičtena zpráva nastavená v atributu.

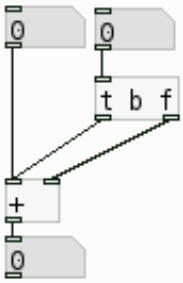
Zkusme si zapamatovat, že tok dat je v Pd třeba usměrňovat s pomocí objektu [trigger] a že trigger vykonává akce postupně zprava doleva.



Jeho atributy můžou být: bang (b), float (f), symbol (s), list (l), pointer (p) a any (a). O tom později.

t b f s l p a všechny možné argumenty

[trigger]em je možné zprávu také tzv. přetypovat - číselnou hodnotu typu float můžeme změnit např. na zprávu typu bang. Díky přetypování tak lze aktivizovat pasivní vstupy.

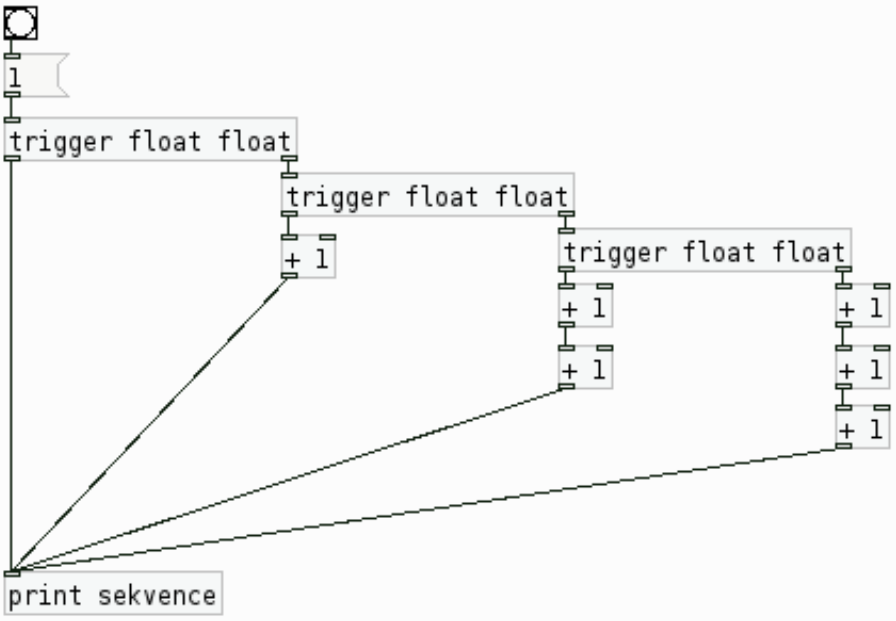


Do pasivního vstupu objektu [+] pošleme s pomocí [trigger]u nejprve číselnou hodnotu a hned poté do jeho aktivního vstupu bang. Hodnota je tedy jednak v objektu [+] "uschována" jako atribut a jednak bang inicializuje přičtení hodnoty, která byla poslána na aktivní vstup objektu [+] ev. dříve. Pokud jsme na aktivní vstup [+] nic neposlali, "dřímá" v něm nula.

SYNTÉZA_PRAVIDEL

S největší pravděpodobností vám zmíněná pravidla budou připadet krkolomná a bude trvat nějakou dobu, než se vám dostanou "pod kůži". Nicméně bez jejich pochopení a správné aplikace není možné mít tok dat v patchi pod kontrolou.

Po tom všem, co jsme si zatím ukázali, by bylo dobré ještě provést syntézu pravidel "odspodu nahoru" a "zprava doleva". Jako ilustrační patch poslouží následující "žebřík". Představte si, že Pd při vyhodnocování událostí vždy "slezou dolů" po žebříku, který je nejvíc vpravo, tam vyhodnotí situaci, a pak postupují nahoru. Nahoře pak postoupí o jednu pozici směrem doleva a zase "slezou" úplně dolů a postupují nahoru atd.

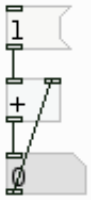


CVIČENÍ:

- pokuste se napsat patch pro výpočet formule $a^2 + 2ab + b^2$.
- pro výpočet mocniny použijte objekt [pow]
- ev. patch navrhněte tak, aby výsledek tiskl při změně jakékoliv proměnné
- podívejte se na dokumentaci k objektu [expr] a pokuste se vzorec vyjádřit s jeho pomocí

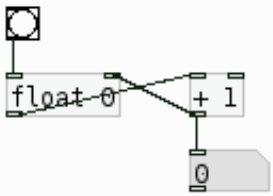
počítadla_a_čas

Počítadla v Pd hrají důležitou roli, stojí např. za počítáním pozice v sekvencerech, díky jejich modifikacím je možné vytvářet nevšední způsoby přehrávání videa, s jejich pomocí lze automatizovat určité procesy atd.

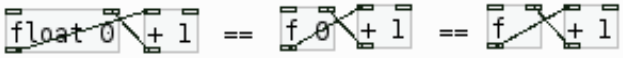


Toto počítadlo nejpve pošle jedničku do aktivního vstupu objektu [+] a výslednou hodnotu z číselného boxu vrátí po kabelu nahoru do jeho pasivního vstupu. Tím se zároveň změní i atribut, který se přičítá.

Klasické počítadlo vypadá takto:



Struktura na první pohled nemusí být zcela srozumitelná, přesto však známe již vše, co je třeba k jejímu pochopení. Každý bang poslaný na aktivní vstup objektu float nejprve protlačí hodnotu, která je v něm uložena. Ta doteče do aktivního vstupu objektu [+ 1], kde je k ní přičtena jednička, pak putuje kabelem do pasivního vstupu objektu [float], kde je uložena a čeká na další "protlačení" bangem.

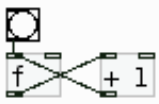


Různé zápisy téhož.

NEKONEČNÉ_SMYČKY

Počítadlo v Pd je krásným příkladem kódu, ve kterém vyhodnocení jednoho stavu determinuje stav následující. Výsledek výpočtu je poslán "zpátky nahoru". Krása autodeterministických kódů v sobě ale nese nebezpečí vytvoření nekonečné smyčky - bezvýchodného zacyklení kódu.

V případě počítadla k tomu postačí špatně zapojit jeden kabel:

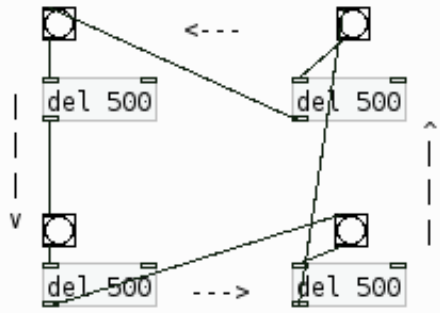


Zde dochází k zacyklení, jelikož z [+ 1] posíláme hodnotu do aktivního vstupu objektu [f], který ji ihned zas předá objektu [+ 1] a ten ihned... a objekt [f] ihned... a ...

Na chyby související s nešetrným posíláním vstupu na výstup jsme vždy upozorněni v Pd konzoli, kde uvidíme hlášku "stack overflow", neboli přetečení zásobníku. Pokud toto varování uvidíte, je třeba kód revidovat, chybu způsobující přetečení zásobníku najít a odstranit. Tato chyba totiž vede k chybné funkčnosti kódu nebo k uplnému "zamrznutí" programu.

Další příklady nebezpečných patchů, vedoucích k zacyklení, vypadají takto:

Zacyklení číselných boxů a bangů.

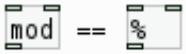


V případě zacyklení bangů lze přetečení zásobníku předejít "zbrzděním" bangu objektem [del] (delay), jehož atribut určuje, o kolik milisekund se má vyhodnocení zprávy zpomalit.

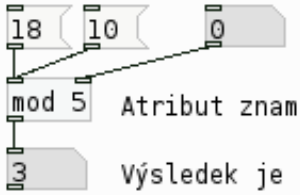
Postavili jsme tak vlastně primitivní a nekonečný kruhový sekvencer.

MODULO

V předchozí variantě počítadla jsme neošetřili horní maximální hranici, do které budeme počítat. Jednak bychom po dlouhé době dospěli k hodnotě, která by překračovala možnosti reprezentace v počítači, a jednak v případě kódů, ke kterým pomalu směřujeme, spíš budeme potřebovat počítat jen stále se opakující řady. K tomu nám pomůže objekt [mod].



Objekt [mod] nám na výstupu dá zbytek po dělení. 18 / 5 jsou 3 se zbytkem 3.

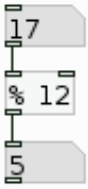


Na pasivním vstupu měníme atribut objektu [mod].

Atribut znamená jakou hodnotou dělíme.

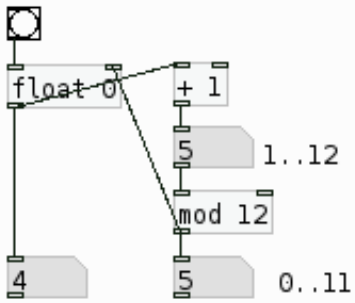
Výsledek je zbytek po dělení.

Pomůckou pro pochopení chování objektu mod a modulární aritmetiky nám může být způsob počítání hodin na ciferníku. Když k šesti hodinám přičteme čtyři, je deset hodin. Když k šesti ale přičteme sedm, je jedna hodina.



Jako bychom počítali hodiny na ciferníku.

Když konstrukci počítadla rozšíříme o mod, budeme mít kód, který bude počítat opakující se řady.



Z různých míst v kódu dokážeme získat různé výsledky.
Z objektu [float] dostaneme řadu 0..11, která bude počítána "dřív" než výstup z objektu [mod], protože pochopitelně ještě neprošla objekty [+ 1] a [mod 12].

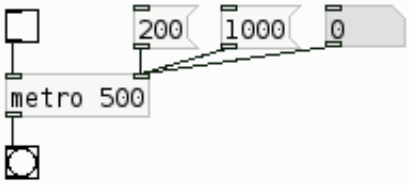
CVIČENÍ:

- pokuste se sestrojit alternativní počítadla, která přičítají dvojku, nebo počítají mocniny dvou, tří atd.
- sestrojte počítadlo dávající řadu 10..20.
- jak počítadlo vynulujeme?

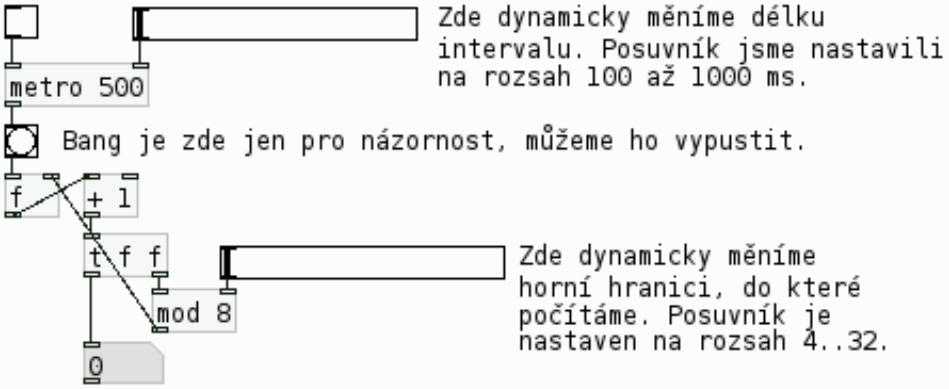
METRO(NOM)

Umíme již sestavit počítadlo, ale zatím nám funguje, jen když ručně klikáme na bang. Možná jste si již položili otázku: jak počítadlo automatizovat? Vzpomeňme si na kapitolu o tlačítkách a vypínačích - ano, odpověď najdeme v objektu [metro].

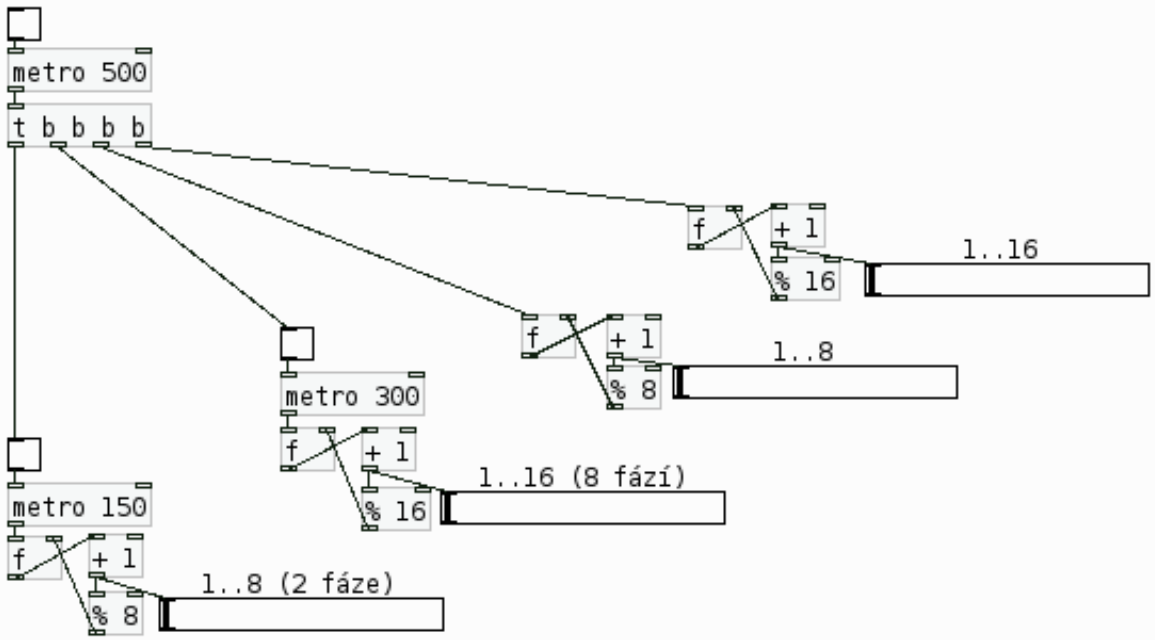
Objekt [metro] je vlastně metronom, který v pravidelných intervalech posílá bang. Zopakujme si, že se zapíná zprávou [bang(nebo [1(poslanou do aktivního vstupu a vypíná se zprávou [stop(nebo [0(. Délku intervalu buď můžeme specifikovat přímo argumentem, nebo dynamicky zprávou poslanou do pasivního vstupu. Délka intervalu se zadává v milisekundách.



Spojením metra s počítadlem dostaneme pomyslný "motor" - jednotku sloužící k pohánění různých částí kódu.

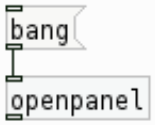


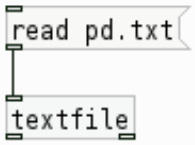
Jedním metrem můžeme samozřejmě pohánět více počítadel, spouštět jiná metra atd.



zprávy_pod_drobnohledem

Na předchozích stránkách jsme se seznámili se zprávami, jejichž funkce obvykle spočívá v tom, že jejich prostřednictvím mezi sebou objekty komunikují. Zpráva může po daném objektu něco vyžadovat: aby změnil svůj atribut, otevřel soubor, provedl výpočet atd. Kromě zvukových signálů považují Pd všechna ostatní data za zprávy. V této kapitole se na ně podíváme podrobněji, i když to bude spíš v duchu "že je to tak a tak" než proč tomu tak je.





K základní práci s Pd nejsou informace v této kapitole zcela nezbytné, nicméně její terminologii byste znát měli, protože s ní dále budeme pracovat. Pokud vám ale přijde nesrozumitelná, přeskočte ji a ev. se k jejímu studiu vraťte později.

ATOMY_A_ANATOMIE_ZPRÁVY

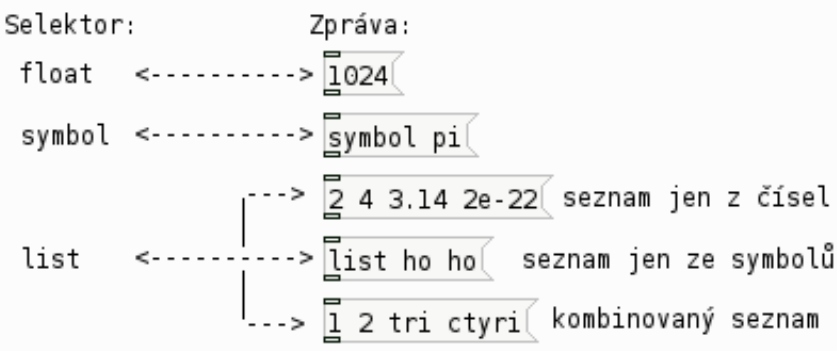
Atomy jsou prvky obsažené ve zprávě. Základní dva typy atomů jsou čísla a symboly. Čísla jsou např.: 1, -24, 2e+10, 3.14, 0. Vše ostatní, co Pd neidentifikuje jako číslo, je chápáno jako symbol: stop, open, zero, 1..10 atd. Těmto dvěma atomům odpovídá číselný box a symbol box, do kterých můžeme buď to nebo ono poslat. Obecně zpráva v Pd sestává z jednoho prvku zvaného selektor (nečíselný atom) a jednoho nebo více dalších atomů, tedy čísel a/nebo symbolů.

Jednotlivé atomy seznamu s implicitním selektorem list.

SELEKTOR

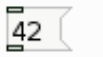
Zpráva v Pd může obsahovat jeden atom číslo, jeden atom symbol, nebo více atomů - pak jde o seznam (ten obsahuje čísla a/nebo symboly). Od této typologie se odvíjejí také různé typy tzv. selektorů: float, symbol, list atd.

V Pd jsou předdefinované tyto selektory:
- float
- symbol
- list
- bang
- pointer

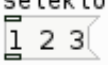


Co to selektor vlastně je? Selektor je první a nečíselný atom. Je ve zprávě buď implicitně obsažený nebo explicitně zapsaný a označuje jaký typ dat v sobě zpráva nese. Díky němu daný objekt, do kterého zprává doputuje, "pozná", o jakou zprávu jde a co s ní může dělat (tzn. vybere pro její zpracování vhodnou metodu). Pokud např. pošleme zprávu [jedna(do objektu [float], nebude si s ní vědět rady, protože čeká na zprávu, která začíná selektorem float. V konzoli uvidíme chybovou hlášku: "float: no method for 'jedna'".

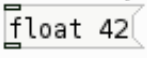
Implicitní (neviditelný) selektor float:

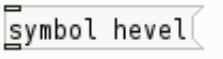


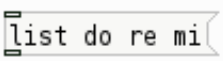
Implicitní selektor list:

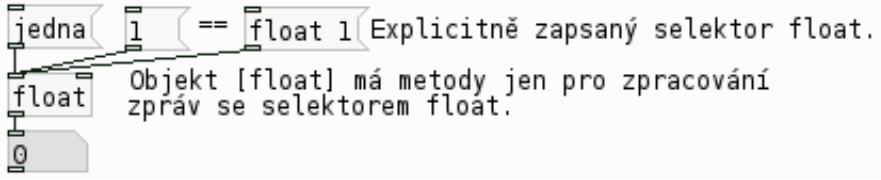


Explicitní selektory:







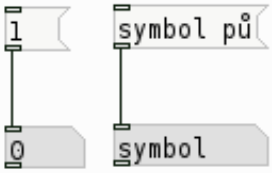


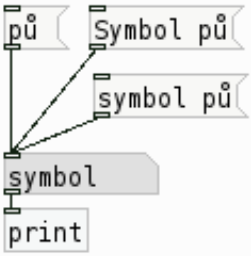
Doposud jsme se v Pd obešli bez použití selektorů a vše zdá se fungovalo, to ostatně proto, že jsme pracovali převážně s čísly, která selektor float nebo list implicitně mají.

Právě proto, že v Pd se často zachází s čísly, jsou selektory pro jedno číslo a seznam začínající číslem implicitní. Psát před každým číslem float by bylo trochu únavné.

ATOMICKÁ_ZPRÁVA

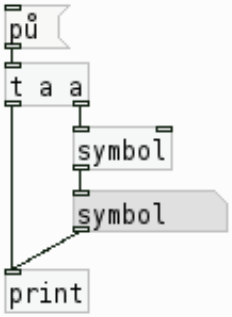
Atomická zpráva obsahuje jeden atom: buď číslo nebo symbol - takže se akorát "vejde" do číselného nebo symbol boxu. Již jsme viděli, že u atomické číselné zprávy není třeba selektor explicitně uvádět, u symbolu ale ano!





Symbol box (CTRL + 4) čeká na zprávu začínající explicitním selektorem "symbol". Proto neumí interpretovat zprávu [pů(ani [Symbol pů(. Pd jsou tzv. Case Sensitive - rozlišují malá a veká písmena. Podívejte se na chybová hlášení do konzole.

nebo:



Jediný správný tvar zprávy pro symbol box má dva prvky: první je selektor "symbol" a druhý je nečíselný atom. Jakékoliv další atomy oddělené ve zprávě mezerou budou ignorovány.

Symbol box "schovává" selektor symbol, ve výstupu na konzoli ho ale vidíme. Stejně tak když do symbol boxu něco napíšeme a potvrdíme enterem - v konzoli uvidíme nejprve selektor symbol a pak zadaný text.

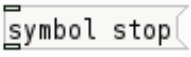
ZPRÁVA_-_PŘÍKAZ

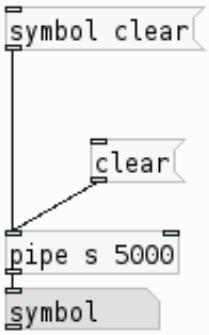
A co zprávy jako [stop(, jež nespadají do selektorů, které jsou v Pd předdefinovány (tedy float, symbol, list, bang, pointer)? Můžeme o nich uvažovat jako o uživatelsky definovaných selektorech nebo jako o příkazech pro daný objekt.

Uživatelsky definovaný selektor/ příkaz stop:

Rozlišovat mezi uživatelsky definovaným selektorem a atomickou zprávou obsahující symbol je důležité, protože některé objekty jinak reagují na symbol a jinak na selektor. Jako příklad nám poslouží objekt [pipe]:

Symbol stop:





Objekt [pipe] umí přijmout atomickou zprávu typu symbol jako tuto a po pěti vteřinách ji poslat dál.

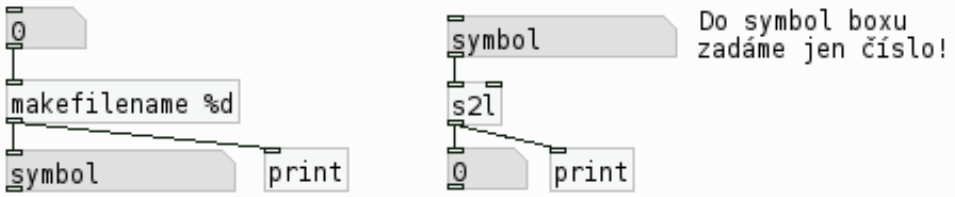
Ale příkaz [clear(objekt [pipe] vyčistí. Když na něj kliknete do pěti vteřin od poslání symbolu clear, na výstupu nebude nic.

Zdůvodnění tohoto rozlišování si můžeme přiblížit ještě takto: Kdyby Pd nerozlišovala mezi "clear" a "symbol clear", jak by pak objekt [pipe] mohl "vědět", zda chceme poslat dál "zbrzděnou" zprávu nebo jej vyčistit?

Když budeme chtít někam poslat atomickou zprávu typu symbol, pak ji budeme muset vždy selektorem symbol specifikovat! Jinak si Pd "myslí", že první nečíselná zpráva je selektor.

KONVERZE_SELEKTORŮ

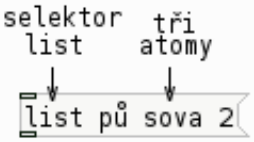
Pokud bychom náhodou potřebovali v nějakém speciálním případě změnit selektor float na symbol a naopak, je to možné provést následujícím způsobem:

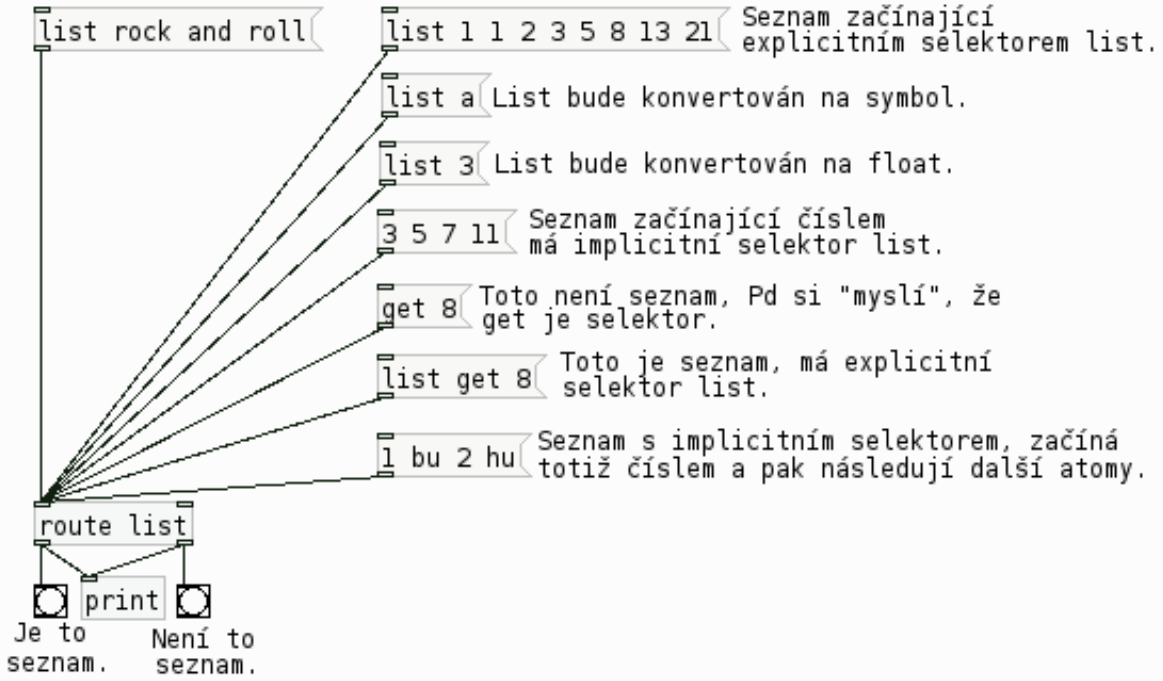


Rozborem objektů [makefilename] a [s2l] se zde nebudeme zabývat, ale můžete si zkusit projít jejich dokumentaci.

SEZNAMY

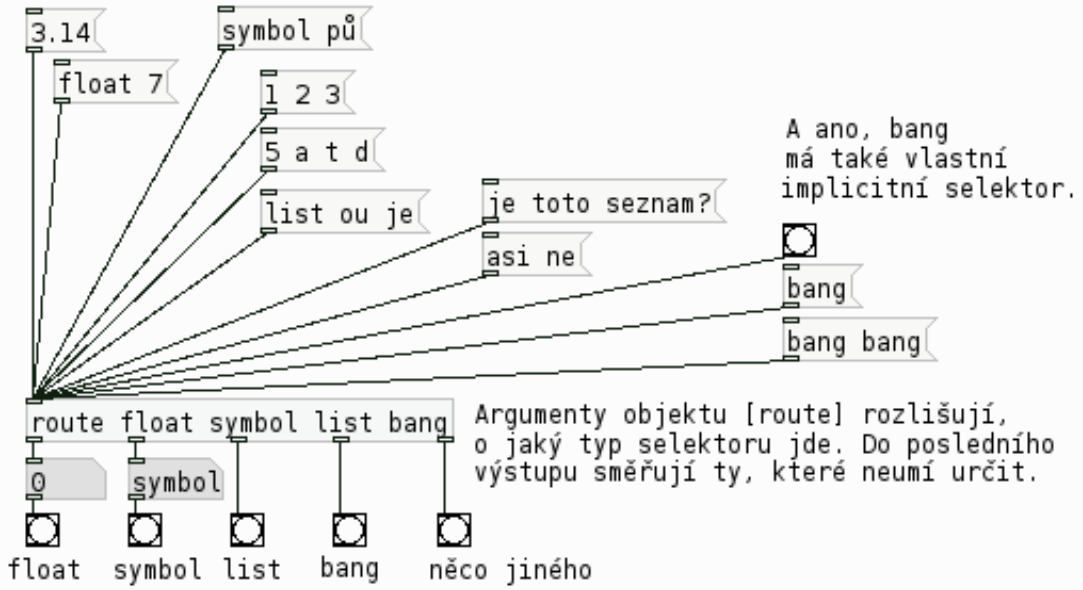
Zprávy obsahující více než jeden atom jsou seznamy. V případě, že začínají číslem, mají implicitní selektor list. Jinak je třeba je selektorem specifikovat (Pd opět interpretují první nečíselnou zprávu jako selektor)! Pokud za list selektorem následuje pouze jeden atom, je list převeden na float nebo symbol.





V tomto příkladu jsme použili objekt [route], se kterým se seznámíme později. Funguje zde jako detektor seznamu (tj. zprávy s implicitním nebo explicitním selektorem list). Když jde o list, pak pošle bang do levého výstupu, jinak pošle bang do pravého výstupu.

Abychom si předchozí informace o zprávách, implicitních a explicitních selektorech i o atomech shrnuli, v rozšířené podobě jej použijeme na následující straně ještě jednou...



CVIČENÍ:

- pakliže máte ještě dost sil a odvahy, zkuste v předchozím kódu zdůvodnit, proč je daná zpráva interpretována tak a tak, kde je a kde není třeba použít explicitní selektor, jaké zprávy mají implicitní selektor atd.
- pakliže ne, doporučuji vydat se na procházku...

REKAPITULACE

- selektor je první nečíselný atom ve zprávě
- v Pd jsou předdefinovány selektory: float, symbol, list, bang a pointer
- u čísel a seznamu začínajícího číslem je implicitní selektor float a list, u atomické zprávy typu symbol a u seznamu začínajícího symbolem je třeba zprávu selektorem symbol a list vždy(!) specifikovat
- různé objekty umí různě reagovat na uživatelsky definované selektory neboli příkazy; o tom, jaké to v případě toho kterého objektu jsou, se dočteme v dokumentaci

KAPITULACE?

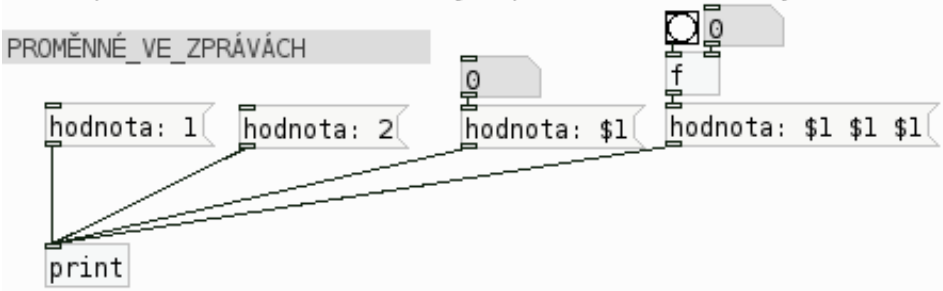
Pokud jste se do studia této kapitoly přeci jen pustili a dočetli až sem, doufám, že vás příliš nevystrašila. Chtěl bych vás trochu uklidnit, protože další probírané pasáže snad již tak abstraktní nebudou a po rozšíření "slovní zásoby" se konečně dostaneme i k praktickým příkladům.

Uklidnit vás může také fakt, že k tomu, abychom si v Pd něco naprogramovali tuto látku nemusíme detailě znát. Až v praxi narazíte na chybu související se selektory, nebo až vám symbol box v konzoli bude hlásit chybu, třeba si na ni vzpomenete. Pokud je mezi čtenáři ale někdo, komu v detailnosti nestačila, může se podívat do manuálu (Nápověda -> Pd Help Browser -> Pure Data -> 1.manual) nebo na dokumentaci all_about_atoms a all_about_messages (Nápověda -> Pd Help Browser -> Pure Data -> 5.reference). K základním definicím se lze dobrat i zde. Teprve pak uvidíte, kolik jsme toho nezmínili.

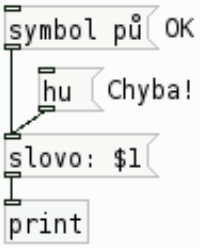
zprávy_a_proměnné

V praxi se nám často přihodí, že budeme ve zprávě chtít měnit dynamicky její obsah. Představte si, jak nepraktické by bylo mít "natvrdo" definovaných 3 x 256 tlačítek jen proto, abychom mohli měnit barevnost v červeném, zeleném a modrém 8bitovém kanále. Praktičtější řešení vede ke třem posuvníkům, které dynamicky mění hodnotu v seznamu o třech atomech.

Nebo si představte situaci, kdy budeme chtít měnit noty v MIDI sekvenceru, nastavovat obálku, která ovládá hlasitost zvuku v syntetizátoru, otevírat různé soubory atd. Zkrátka: v programování se bez možnosti, jak nahradit něco něčím jiným, neobejdeme. K tomu, abychom tyto změny mohli efektivně provádět, slouží proměnné. Ty se v Pd značí jako $1..$n. O proměnné $0 si něco řekneme až později. Zatím pro nás budou existovat jen proměnné od $1 a výš.

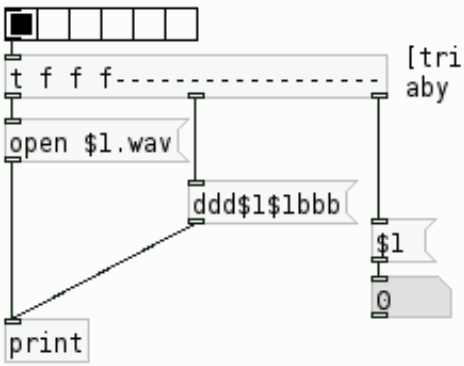


V prvních dvou případech je hodnota ve zprávách "natvrdo" zapsána. Dynamicky ji můžeme měnit ve třetí a čtvrté zprávě, kde vidíme znak "$1". Kdykoliv narazíme na znaky $1, $2..$n, můžeme si místo nich představit prázdná místa, která čekají na zaplnění nějakou zprávou (číslem, symbolem).



Ve čtvrté zprávě je proměnná třikrát zopakována a výsledkem je zpráva, kterou uvidíme v konzoli po inicializaci bangem - bude v ní třikrát tatáž hodnota, kterou jsme zadali do číselného boxu.

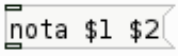
Pokud jste studovali předchozí kapitolu, dodejme, že zpráva pro proměnnou je ve tvaru selektor atom. Čísla fungují automaticky díky implicitnímu selektoru, ale u symbolu a seznamu, který nezačíná číslem, selektor musíme specifikovat.



[trigger] si můžeme nadstavit znakem "-", aby byl patch čitelnější.

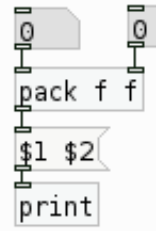
Na vedlejším příkladu je demonstrováno, že proměnné nemusí být odděleny mezerou.

Asi jste si již položili otázku, jak předávat v jedné zprávě více různých proměnných. Jednak musíme ve zprávě vytvořit více proměnných - v případě noty a její hlasitosti to budou dvě proměnné $1 a $2.



PACK

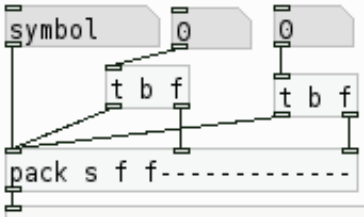
A dále musíme do této zprávy poslat seznam "v jednom balíku" za pomoci objektu [pack]. [pack] podle daného počtu argumentů typu číslo (float, f) nebo symbol (s) vytvoří seznam se selektorem list a pošle ho dál,když obdrží zprávu na aktivním vstupu. Počet argumentů v objektu [pack] by měl odpovídat počtu proměnných ve zprávě, do které je posíláme.



Posíláme proměnné jako seznam.

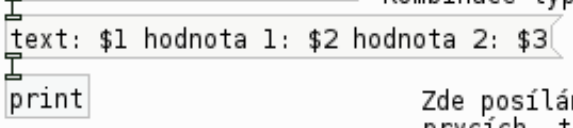
Chyba:

posíláme pouze atomickou zprávu do zprávy, která čeká na seznam o třech prvcích.



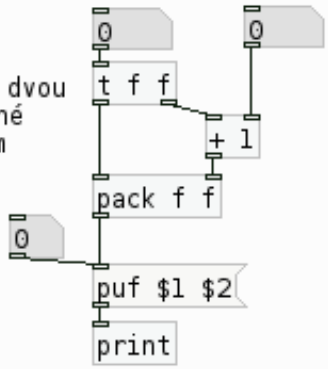
Změna pasivních vstupů objektu [pack] na aktivní s pomocí [trigger]u, který posílá nejprve hodnotu do pasivního vstupu, a pak bang do aktivního.

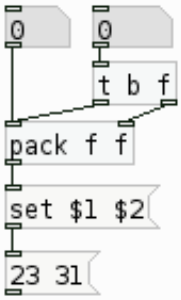
Kombinace typů symbol a float,



Zde posíláme seznam o dvou prvcích, takže proměnné v [puf $1 $2(máme čím "nakrmit".

Tady ale posíláme jen jednu proměnnou, takže $2 už čím "nakrmit" nemáme -> chyba.



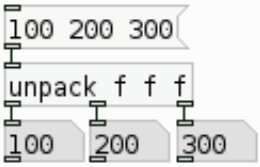


S pomocí zprávy začínající selektorem "set" následovaným určitým počtem proměnných lze dynamicky měnit obsah zprávy, která v ní po uložení patche zůstane. Využijete toho, když budete chtít, aby si váš patch nějaká nastavení pamatoval.

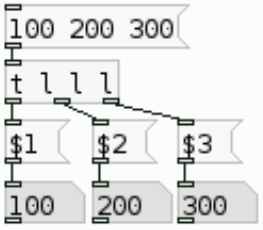
Zkuste prozkoumat, co se stane, když selektor "set" nahradíte selektorem "add" nebo "add2".

UNPACK

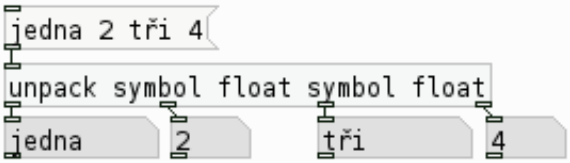
Opačný proces, než je zabalení atomů do jednoho seznamu objektem [pack], je rozbalení seznamu do více atomů. Objekt, který to umí, se jmenuje [unpack].



Složitěji bychom to také mohli napsat takto:



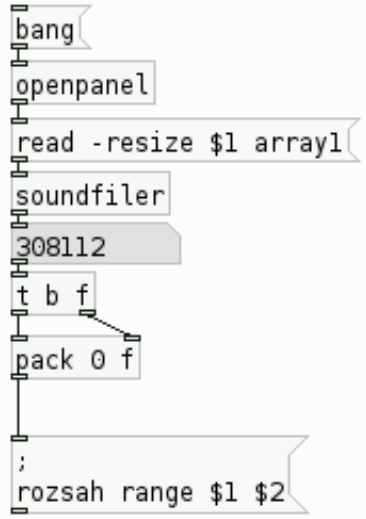
Vyber ze seznamu jen první, jen druhou a jen třetí proměnnou.



Pořadí a typ dat v objektu [unpack] se odvíjí od toho, jak jsou typy dat řazeny ve zprávě.

SCRATCHER

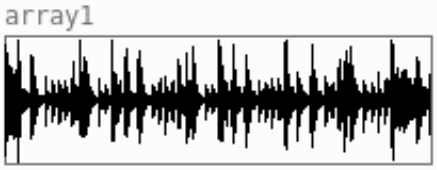
Opustíme teď výklad teorie ve prospěch případové studie, ve které sice nebudeme ještě všemu rozumět, ale uvidíme na ní, jak se pracuje s proměnnými. Ty v následujícím patchi reprezentují název souboru, který přehráváme, a délku samplu, která určuje rozsah "scratchovacího" posuvníku.



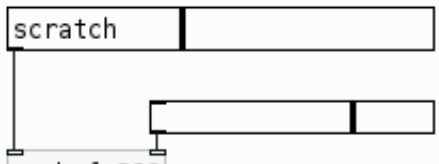
[openpanel] po bangnutí otevře okno, ve kterém vybereme audio soubor s koncovkou .wav. Jeho název posíláme do zprávy, kde nahradí $1. Zpráva je dál poslána objektu [soundfiler], který soubor .wav načte do pole s názvem "array1" a přizpůsobí jeho velikost.

Tato zpráva začíná středníkem, tzn. že posílá něco do právě otevřeného patche. Zde posíláme dvě čísla do posuvníku.

Proměnné $1 a $2 reprezentují dolní a horní hodnotu posuvníku - tedy $1 je vždy 0 a $2 je celkový počet vzorků.

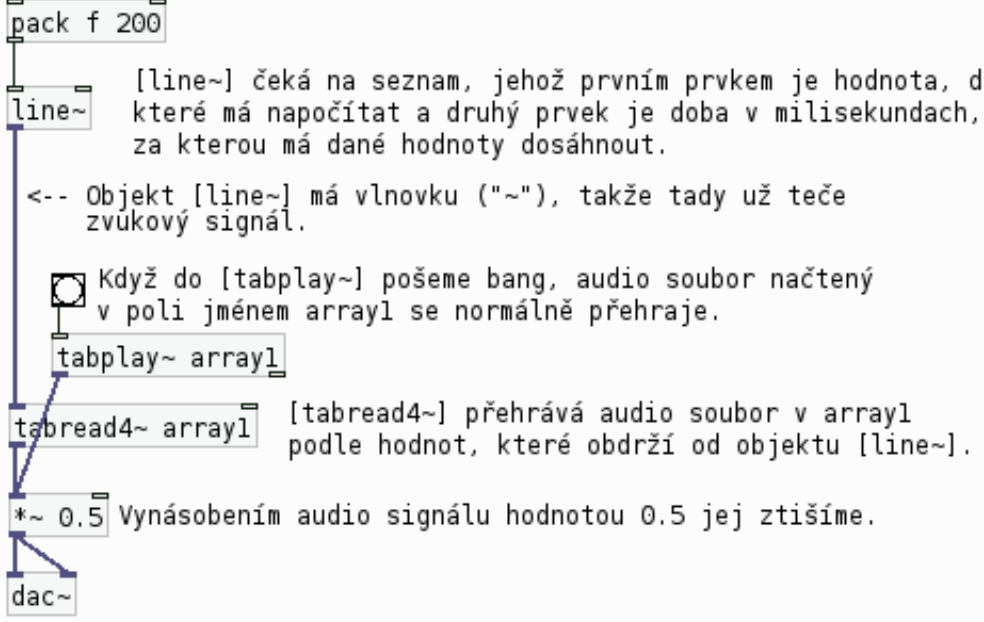


Pole (menu Vložit -> Pole) je datová struktura, do které můžeme uložit různá data. V tomto případě to bude audio soubor typu WAV, jenž do pole array1 nahraje objekt [soundfiler].



Tímto posuvníkem "procházíme" po audio souboru. Podívejte se na jeho vlastnosti (PTM), jaký má rozsah a jaký symbol přijímá?

Druhým posuvníkem měníme rychlost "náběhu".



Ke kompletnímu vysvětlení funkčnosti tohoto patche se dostaneme ve druhé části rukověti, která se věnuje zvuku. Konkrétně to bude v kapitole věnované objektu [tabread4~].

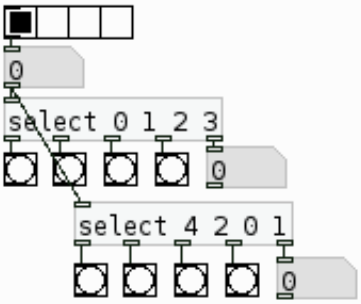
rozšiřujeme_slovní_zásobu

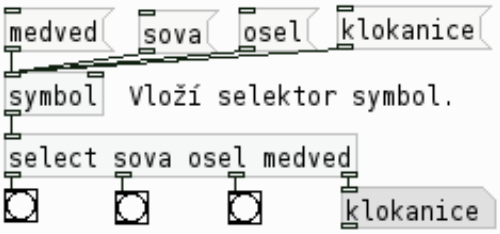
Základní principy programování v Pd máme již za sebou.
Předmětem této kapitoly bude rozšíření "slovní zásoby".
Seznámíme se s objekty, které ve své programátorské praxi
budete jistě často potřebovat. Je dobré je aktivně znát.

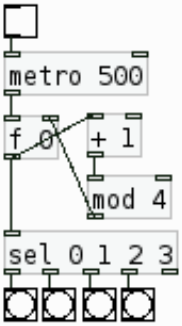
[SELECT]

Objekt [select] je "detektor shody". Čeká na vstupu na
atomickou zprávu, a pokud tato zpráva odpovídá některém
z jeho argumentů, pak na daný výstup pošle bang. [select]
má vždy o jeden výstup víc, než je počet argumentů - do
posledního výstupu totiž "zahazuje" vše, s čím se neshodne.
V jakém pořadí atributy napíšeme, je lhostejné.

select == sel







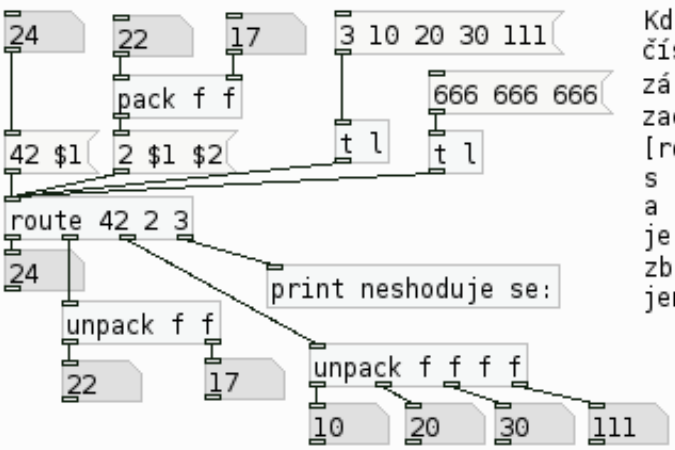
V kombinaci s modulárním počítadlem
a objektem [metro] tvoří [select]
základ sekvenceru.

Kód známý z předchozích stránek.

Těmito bangy pak můžeme spouštět nějaké
akce, nebo jimi inicializovat pozice
v sekvenceru atd.

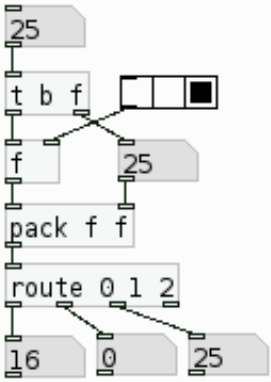
[ROUTE]

Objekt [route] se chová podobně jako [select], jen s tím
rozdílem, že pracuje se seznamy a ne s atomickými zprávami.
Již jsme ho jednou použili jako detektor typů dat, které
jsou ve zprávě - pak jsou jeho argumenty: float, symbol,
bang a list. Jeho argumentem může být ale i číslo nebo
libovolný symbol. Na vstupu [route] čeká na seznam, jehož
první atom porovnává se svými agrumenty. Pokud detekuje
shodu, pak zbytek seznamu za shodným atomem pošle na daný
výstup.



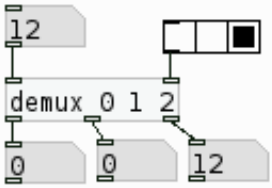
Když změníme hodnotu v prvním
číselném boxu, vytvoříme tím
zároveň seznam, který bude vždy
začínat číslem 42.Ten doteče do
[route], který detekuje shodu
s prvním atomem v tomto seznamu
a na prvním výstupu (protože 42
je první argument [route]) pošle
zbytek seznamu (v tomto případě
jen jedno číslo).

[pack] spolu s objekty [select] a [route] jsou základem něčeho, co bychom mohli pojmenovat jako "presetová vrstva" - tedy část kódu, která slouží k vytváření a nastavování parametrů, jež pak posíláme např. do syntetizéru.



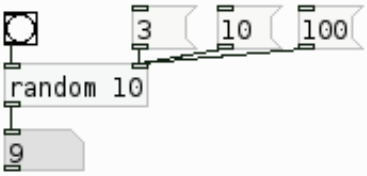
Se znalostí předchozí látky bychom také měli již být s to sestavit "rozbočovač" zpráv. Podívejte se na vedlejší patch a pokuste se zdůvodnit použití objektu [f] a [pack f f].

S využitím objektu [demux] z knihovny zexy je to trochu jednodušší:



RANDOM

Když objektu [random] na levý aktivní vstup pošleme bang, vygeneruje pseudonáhodnou hodnotu v rozsahu od 0 až do argument-1.

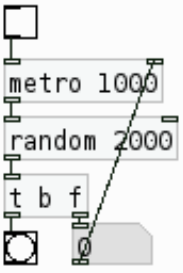


Číselnou hodnotou poslanou do do pravého vstupu dynamicky měníme horní hranici generovaných hodnot.

Generátor náhodných MIDI not v rozsahu jedné oktávy.

C4 = 60, C5 = 72

Zde máme rozsah 60..72.



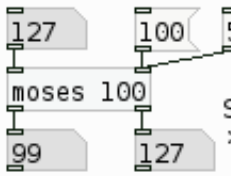
Aperiodický časovač.

Pseudonáhodná čísla využijete, kdykoliv budete potřebovat "ušpinit" příliš čistý proud dat, nebo v aleatorické kompozici. Nebo si s pomocí objektu [random] postavte primitivní Pd věštírnu.

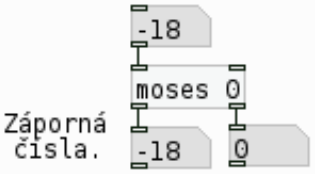
Problematika generování pseudonáhodných čísel je velmi zajímavá. Pokud by vás zajímaly podrobnosti,podívejte se sem, sem a sem, nebo na stánku Kena Perlina - autora věhlasného Perlinova šumu.

[MOSES]

Objekt [moses] přijímá na levém vstupu čísla. Pokud je přijaté číslo větší nebo se rovná argumentu, je posláno do pravého výstupu, jinak do levého. Pravý vstup opět slouží k nastavení argumentu.



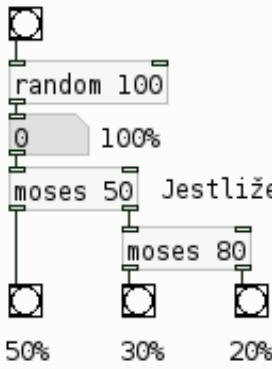
Sem půjdou čísla >= 100.



Záporná čísla.

Kladná čísla.

Následuje ukázka, jak kombinací objektu [random] a kaskády objektů [moses] generovat náhodné události s váhou. Objekty [moses] rozdělí řadu čísel 0..99 do tří sekcí (0..49, 50..79, 80..99), což odpovídá 50%, 30% a 20% pravděpodobnosti aktivace daného bangu.

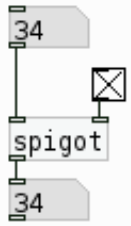


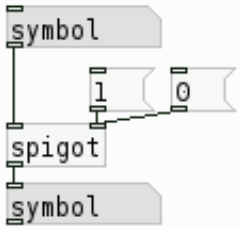
Jestliže je x >= 50, pak čísla tečou doprava.

X >= 50 a zároveň x < 80, pak dotečou do levého výstupu, jinak do pravého.

[SPIGOT]

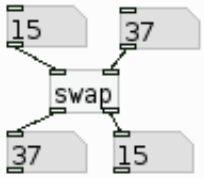
Objekt [spigot] využijete vždy, když budete potřebovat "přiškrtit" proud dat - ať už jde o čísla, symboly nebo seznamy. Když do jeho pravého vstupu pošleme jedničku, data přicházející na jeho levý vstup propouští, a když nulu, tak ne.



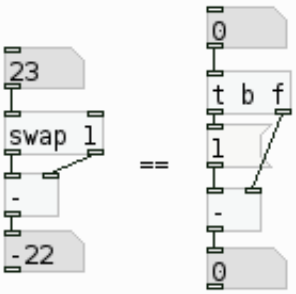


[SWAP]

Na první pohled vypadá objekt [swap] velmi primitivně - prohazuje číselné hodnoty na vstupech a výstupech. Proč ho pak v patchi nenahradit překřížením drátů? Pokud mu totiž specifikujeme jeden číselný argument, prohazuje vždy tento argument s příchozím číslem. Využijeme ho, když budeme potřebovat např. nějakou hodnotu odečíst od konstanty nebo získat podíl konstanty a příchozí hodnoty.

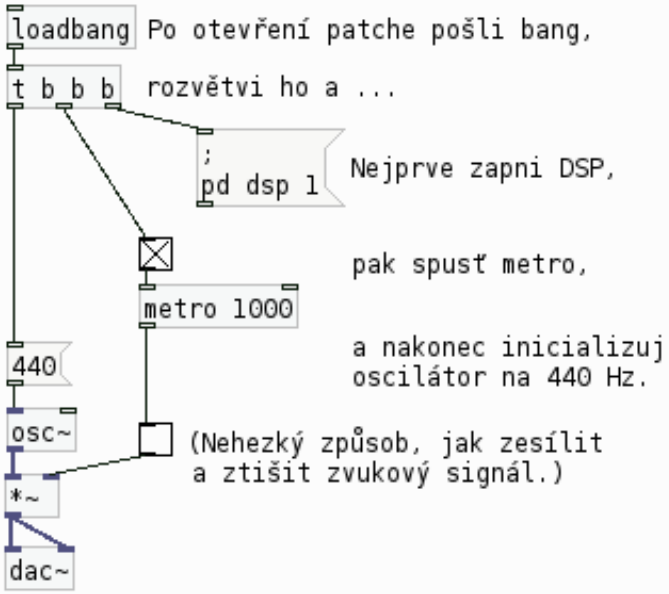


Ve vedlejším příkladu odečítáme příchozí hodnotu z číselného boxu od jedničky. Dokážeme to sice již zapsat s pomocí objektu [trigger], ale kód s objektem [swap] je jednodušší.



[LOADBANG]

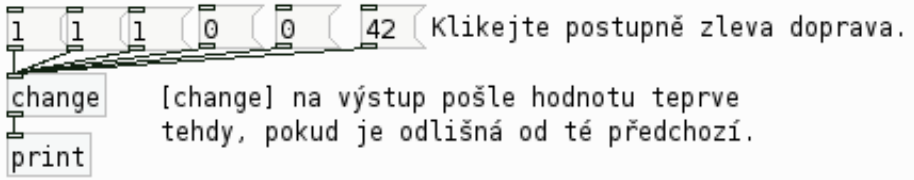
Až v Pd naprogramujete svůj první projekt, např. nějaký audio-vizuální nástroj, pak narazíte na problém týkající se jeho inicializace a přednastavení. Inicializovat hodnoty ručně vždy po otevření patche by nás zbytečně obtěžovalo. Řešením je použití objektu [loadbang], který pokaždé, když otevřeme patch, pošle bang. Ten obvykle vede do zpráv s přednastavenými hodnotami, nebo spouští jiné akce související s inicializací systému.

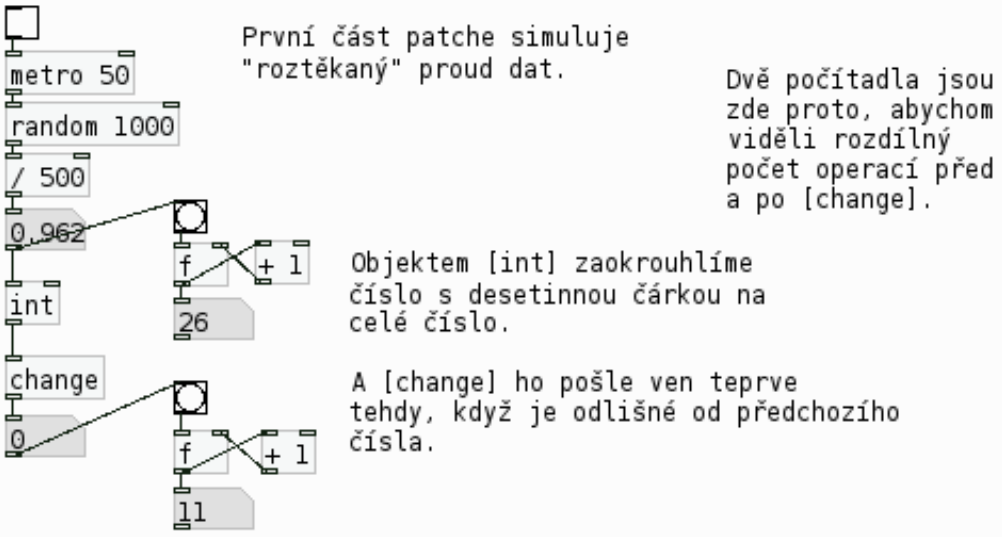


Zkuste si tento patch přepsat a uložte ho jako samostatný soubor. Pak ho zkuste zavřít a znovu otevřít...

[CHANGE]

Objekt [change] najde využití v situacích, kdy budeme potřebovat provést nějakou akci pouze při změně přijaté hodnoty. Takovou situací je např. filtrování proudu čísel, jaký obdržíme při čtení dat z analogových senzorů (fotorezistor, potenciometr, joystick atd.).

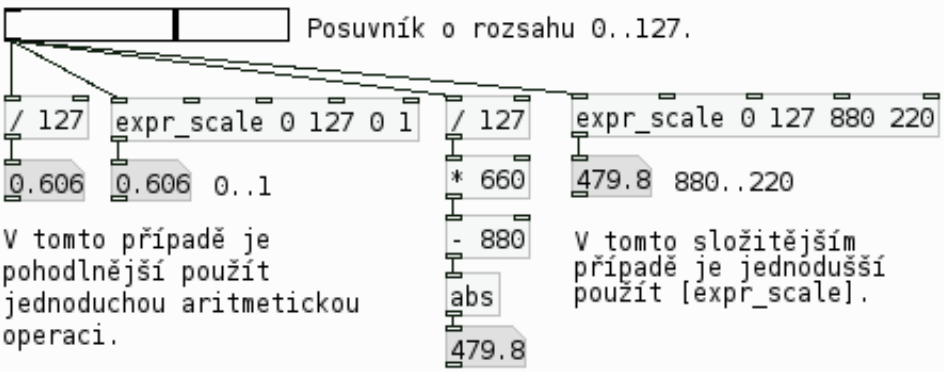




[EXPR_SCALE]

[expr_scale] je velmi užitečný objekt (přesněji: jde o abstrakci z knihovny jmmmp, která je součástí Pd-extended, můžete ji LTM otevřít a podívat se dovnitř. O abstrakcích pojednáme později), který umí lineárně přeškálovat hodnotu v daném rozsahu na hodnotu z jiného rozsahu. Lze to provést i pomocí jednoduchých aritmetických operací (např. posuvník o rozsahu 0..127 přeškálujeme na rozsah 0..1 pouhým dělením hodnoty číslem 127). S objektem [expr_scale] je to ale velmi jednoduché i ve složitějších situacích.

První dva argumenty jsou spodní a horní rozsah příchozí hodnoty, třetí a čtvrtý argument pak spodní a horní hodnota rozsahu, na který chceme vstup přeškálovat.

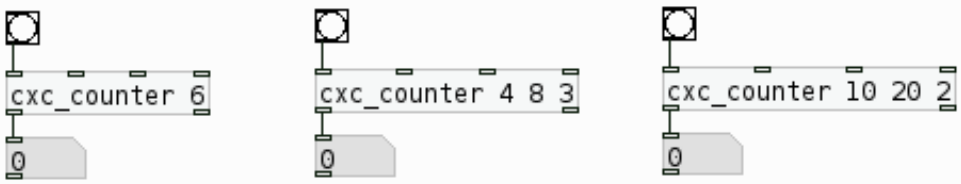


Pozor: nepleťme si objekt [expr_scale] a [scale]. Druhý jmenovaný je součástí knihvny GEM a slouží ke škálování vykreslované scény. Když bychom do něj poslali nějaké číslo, upozornila by nás na to konzole upozornila hláškou: "scale: no method for 'float'".

[CXC_COUNTER]

Podobně jako nám [expr_scale] umožňuje obejít konstrukci aritmetických formulí při škálování, pomůže nám bjekt [cxc_counter] při navrhování složitějších počítadel. Toto počítadlo je součástí Pd-extended a pochází z knihovny cxc.

Pokud chceme počítat od jedničky výš, můžeme [cxc_counter] specifikovat až třemi argumenty: první je minimum, druhý maximum a třetí je způsob počítání (1 = nahoru, 2 = dolů, 3 = ping-pong).

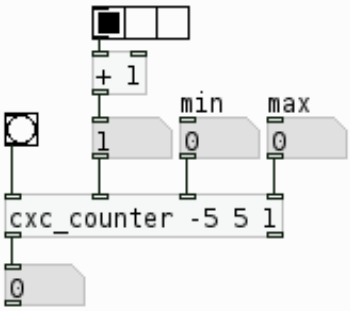


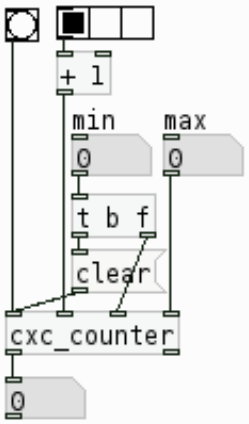
V prvním případě jde o modulo počítadlo, jehož výstupem bude vzestupně se opakující řada 1..6. Druhý [cxc_counter] bude počítat od čtyř do osmi, a protože jsme třetím argumentem specifikovali i způsob počítání (v tomto případě ping-pong), bude výstupem vzestupně a sestupně se opakující řada v rozsau 4..8. Výstupem třetího příkladu bude klesající řada o rozsahu 20..10.

První argument nesmí být nikdy větší než druhý, jinak nebude [cxc_counter] fungovat správně.

Když budeme chtít z objektu [cxc_counter] dostat řadu začínající nulou nebo zápornou hodnotou, pak minimum sice také můžeme specifikovat argumentem, ale ještě před tím, než pošleme do levého vstupu bang, je třeba do něj poslat zprávu |clear(- jinak bude [cxc_counter] počítat od jedné.

Způsob počítání, maximum i minimum lze dynamicky měnit posíláním hodnot na vstupy. Druhý vstup určuje způsob počítání (1 = nahoru, 2 = dolů, 3 = ping-pong), třetí je minimum a poslední je maximum. Pokud ovšem chceme, aby bylo počítadlo vždy inicializováno na minimum, pošleme do levého vstupu pomocí [t b f] ještě zprávu [clear(, jak ukazuje druhý příklad.





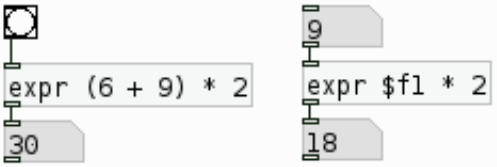
Zde nemáme ošetřeno, aby maximum bylo vždy větší než minimum a aby minimum bylo vždy menší než maximum.

Dokázali byste pomocí objektů [<], [>] a [spigot] tento problém vyřešit?

[EXPR]

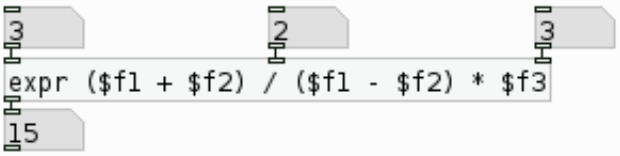
Objekt [expr] vám bude dobrým pomocníkem vždy, když budete potřebovat zapsat jednoduše matematický výraz. Jde to samozřejmě i v dataflow zápisu, ale ti z vás, kteří se pokusili vypracovat cvičení týkající se výrazu $a^2 + 2ab + b^2$, snad potvrdí, že by byli vděční za jednodušší způsob, jak jej vyjádřit.

"Na tvrdo" zapsaný výraz do [expr] je vyhodnocen bangem. Vyhodnocovat jde samozřejmě i výrazy s proměnnými.



Pozor: na rozdíl od zpráv, do kterých se proměnné zapisují jako $1, musíme v [expr] proměnnou psát jako $f1..$fn, což značí proměnnou typu float!

Počet vstupů objektu [expr] se odvíjí od počtu proměnných:



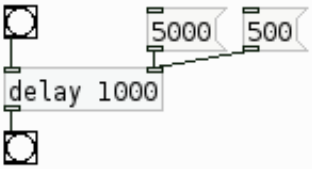
$$\frac{a + b}{a - b} * c$$

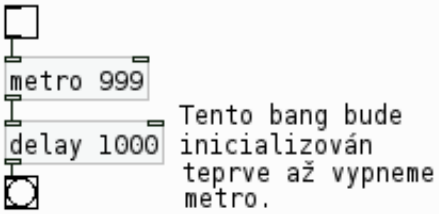
Výraz bude vyhodnocen při poslání hodnoty na aktivní vstup.

Do objektu [expr] lze zapisovat i mocniny, odmocniny, logaritmy, goniometrické funkce, podmínky atd. Podrobnější informace najdete ev. v dokumentaci (Nápověda -> PD Help Browser -> Pure Data -> 5.reference -> all_about_expr...). Později se také seznámíme s audio dvojčetem [expr~].

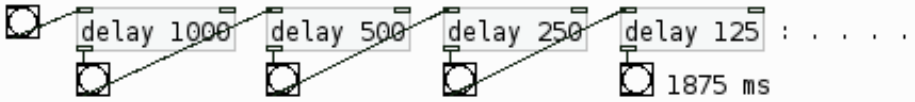
[DELAY]

[delay] je "zbrždovač" bangů. Bang, který mu dorazí na levý vstup, pozdrží o tolik miliseknund, kolik má specifikováno v argumentu. Pokud v čekání obdrží nový bang, starší bang se vymažte a interval se počítá od začátku. Časovou prodlevu specifikujeme buď přímo argumentem nebo posláním číselné zprávy do pravého vstupu.



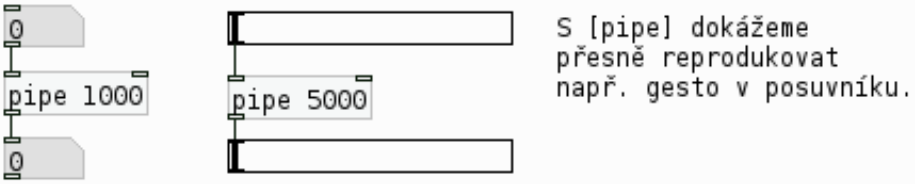


Zřetězením objektů [delay] získáme strukturu, díky které můžeme jedním tlačítkem spouštět další akce s přesným zpožděním.



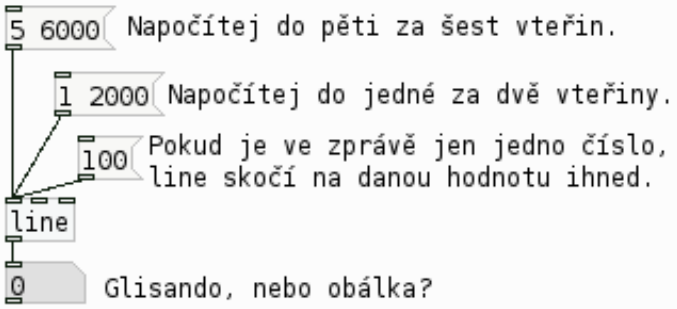
[PIPE]

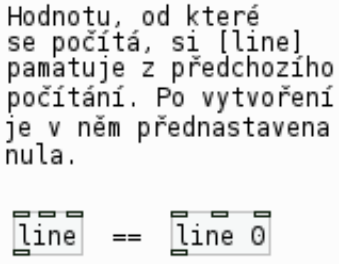
[pipe] funguje stejně jako [delay], jen nepracuje s bangy, ale s čísly a symboly. Přijaté zprávy pozdrží po daný čas a z výstupu je vyšle ve stejném pořadí, jako je přijal.



[LINE]

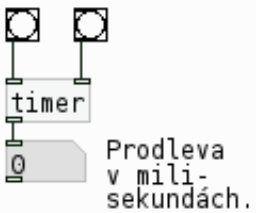
Třetím do sbírky objektů, které pracují s časem, je [line]. Využijeme ho zejména později při konstrukci obálek. Stručně řečeno: [line] se umí dopočítat určité hodnoty za určitý čas. Když mu pošleme zprávu se dvěma čísly, pak první označuje hodnotu, do které má [line] napočítat, a druhá určuje, za kolik milisekund dané hodnoty dosáhne.





[TIMER]

A konečně posledním objektem na této straně, jenž má co do činění s časem, je [timer]. Klikněte na levý, pak na pravý bang a [timer] vám na výstupu řekne, jak dlouhá mezi kliknutími byla prodleva. S [timer]em postavíme trenažéra přesnosti úderů pro bubeníka, nebo tzv. TAP button, jímž můžeme nastavit tempo sound systému podle poslechu vnějšího signálu.



stavíme_krokový_sekvencer

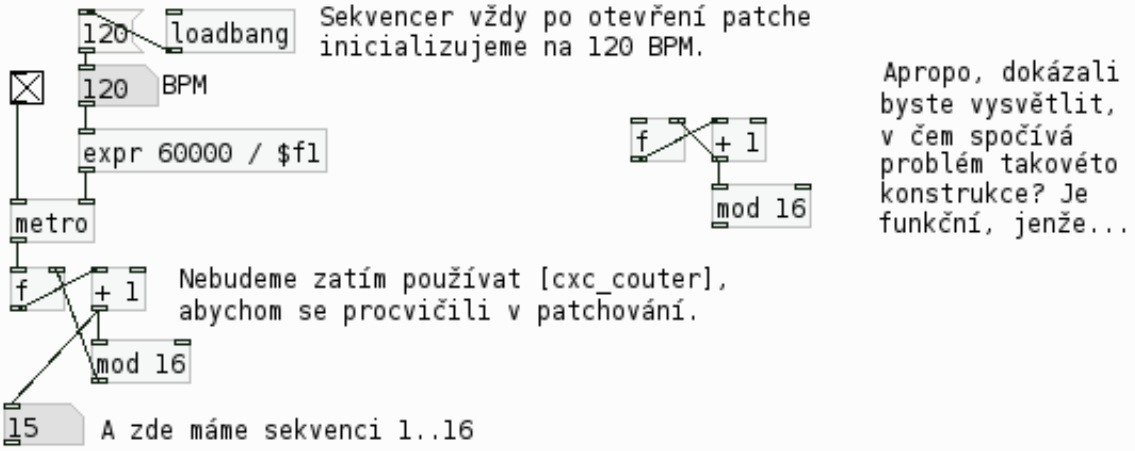
Poté, co jsme si rozšířili "slovní" zásobu, jsme již dostatečně vybaveni k tomu, abychom se mohli pustit do další případové studie. Opustíme obecný výklad ve prospěch konkrétního úkolu, kterým bude stavba krokového sekvenceru. V procesu jeho vytváření zároveň zúročíme to, čím jsme doposud prošli.

Sekvencer je obecně přístroj, jenž umí v čase vytvářet a provádět události - např. zahrát tón nebo zobrazit snímek. Můžeme si ho představit jako řídící jednotku hudební, nebo vizuální kompozice.

SRDCE_SEKVENCERU

Vyjdeme z předpokladu, že náš sekvencer bude mít 16 pozicí (4 takty po 4 dobách). Abychom tyto pozice mohli cyklicky přehrávat, budeme potřebovat modulo počítadlo kombinované s objektem [metro], jehož konstrukci již známe.

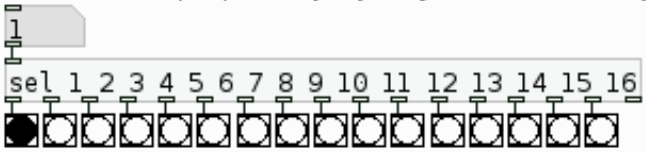
První problém k řešení se týká vhodného zadávání tempa. Milisekundy nejsou zcela vyhovující, standardem je jednotka BPM (počet úderů za minutu). 1 minuta je 60000 milisekund, 1 BPM = 60000 / 1, 2 BPM = 60000 / 2 -> x BMP = 60000/x. Zapíšeme tento výraz do objektu [expr].



Víme také, že [mod] nám umožňuje za běhu měnit maximální hodnotu, do které počítáme, takže kromě tempa budeme moci měnit i délku přehrávané sekvence.

ROZPROSTŘENÍ

Zatím máme k dispozici jen řadu čísel 1..16. Co ale s ní? Jak tyto hodnoty "rozprostřít", abychom každou z pozic viděli lépe jako jedinečnou událost v čase a mohli ji pak něčemu přiřadit? Nebo ještě jinak: jak z čísla udělat bang? Za tímto účelem použijeme objekt [select]. Mohli bychom použít i objekt [route], ten ale slouží k práci se seznamy, takže v tomto případě by bylo jeho užití nadbytečné.

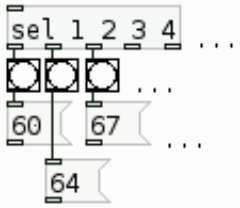


Do číselného boxu natáhneme kabel z postaveného modulo počítadla. Mohli bychom brát i hodnotu z objektu [f], ale pak by [select] musel začít hodnotou 0.

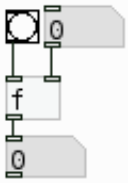
UDÁLOSTI

Teoreticky bychom každý z bangů nyní mohli napojit na jakékoliv kódy, takže bychom sekvencovali 16 rozdílných událostí. Dejme tomu, že ale budeme chtít mít sekvencer vytvářející události stejného druhu: např. MIDI noty v rozsahu jedné oktávy.

Nota C1 odpovídá ve standardu MIDI hodnotě 60. Apropo, tím se dostáváme oklikou k úvaze, proč se Pure Data jmenují právě Pure Data. Může nám to přijít trochu nezvyklé, ale ve svém důsledku je cokoliv, s čím v počítači pracujeme, redukovatelné na číslo. Cokoliv na monitoru vidíme nebo reproduktorů slyšíme, je jen interpretace čísel. Pakliže jsou noty reprezentovatelné v MIDI jako čísla, mohli bychom je "natvrdo" zapsat do zpráv inicializovaných bangem. Bang by v následující ukázce vlastně ani nemusel být, je tam jen pro názornost.

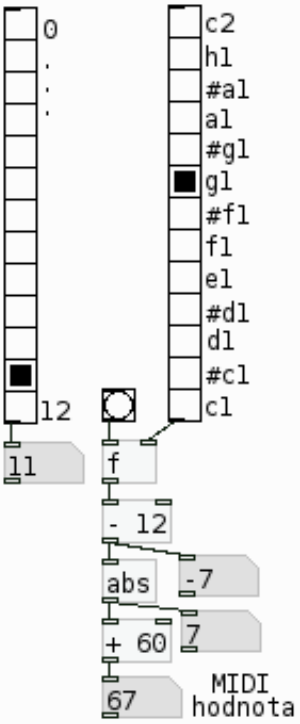


Takový "tvrdý" a jednoúčelový sekvencer by ale těžko hledal uplatnění za hranicemi své melodie. Pokusíme se spíš o konstrukci, která by nám umožnila noty (hodnoty) pohodlně měnit. V tomto bodě využijeme našich znalostí týkajících se vlastnosti Pd objektů - konkrétně půjde o důsledek vyplývající z toho, že rozlišujeme aktivní a pasivní vstupy. Víme, že objekt [f] funguje jako "úschovna" pro čísla. Pokud mu na pravý, tj. pasivní vstup pošleme číslo, neprovede žádnou akci, pouze si ho podrží v sobě. Dostat ho z něj ven lze pomocí bangu poslaného na aktivní vstup.



To je přesně to, co v případě sekvenceru hledáme. Nechceme, aby hodnoty hrály ve chvíli, kdy je zadáváme (to bychom spíš stavěli piáno), ale teprve ve chvíli, kdy jsou v sekvenci inicializovány bangem.

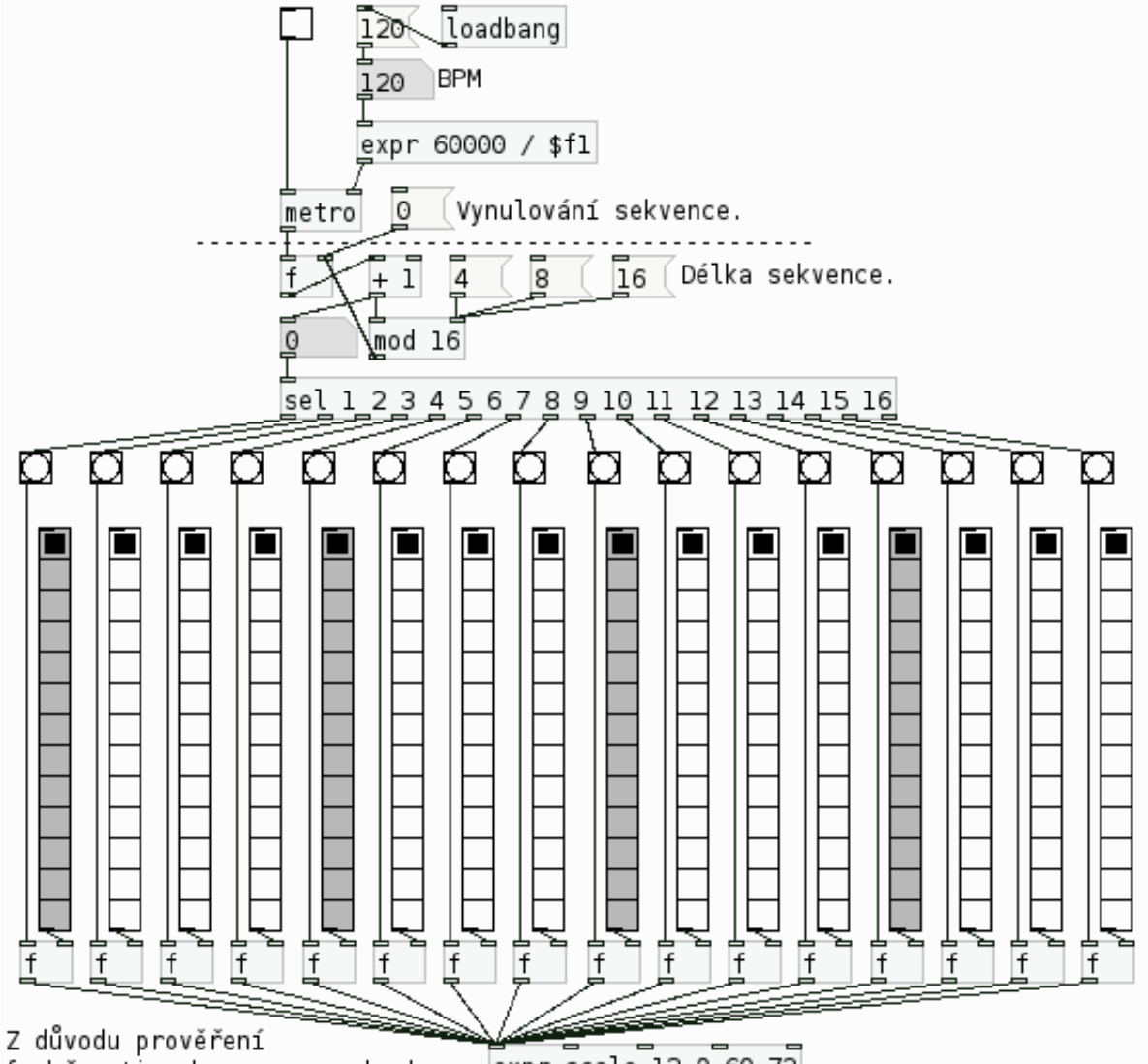
Dalším krokem konstrukční úvahy je otázka, jaký vhodný a uživatelsky přátelský způsob zadávání hodnoty zvolit? Číselný box zrovna uživatelsky moc příjemný není, takže raději sáhneme po nějakém grafickém prvku - nabízí se posuvník nebo přepínač. Posuvník by byl vhodný v situaci, kdy bychom chtěli vytvářet mikrotonální sekvence, protože nám poskytuje "jemnou" škálu hodnot. Pokud chceme ale posílat přesně vyladěné hodnoty, jako jsou noty v MIDI, pak bude pro náš záměr vhodnější vertikální přepínač.



Vytvoříme přepínač (CTRL + SHIFT + d) a ve vlastnostech (PTM -> Vlastnosti/Properties) do položky "number" zadáme 13, což bude odpovídat tónům v rozsahu jedné oktávy. Je zde ale problém: přepínač má dole hodnotu 12 a nahoře 0, to neodpovídá zápisu v notové osnově, kde jsou "vysoké noty nahoře a nízké dole". To vyřešíme objekty [- 12] a [abs]. Když od hodnoty, která je v přepínači nahoře (tj. 0), odečteme 12, dostaneme -12 a objekt [abs] nám vrátí absolutní (tj. nezápornou) hodnotu. Máme za sebou konstrukci části kódu, jejímž výsledkem je jedna událost - hodnota noty v MIDI standardu.

PROPOJUJEME

Poté, co jsme postavili jednotlivé základní části sekvenceru, nám zbývá jen poslední krok, a tím je jejich propojení. U objektu [mod] jsme navíc doplnili hodnoty pro různě dlouhé sekvence. Segment vytvářející jedinečnou MIDI notu jsme museli pochopitelně 16x duplikovat. Nebylo přitom nutné duplikovat celý segment kódu, ale pouze přepínač s objektem [f], protože konverze je pro všechny noty stejná. Bangy jsme v patchi nechali, aby bylo provádění sekvence dobře viditelné. Konverzi na MIDI jsme nahradili objektem [expr_scale]. Nuže, výsledný prototyp vypadá tako:



Z důvodu prověření funkčnosti sekvenceru a abychom nezůstali jen u čísel, je sekvencer doplněn velmi primitivním syntetizérem. Ten převádí MIDI hodnoty na frekvence a ty dál přeposílá oscilátoru.

Nejprve do přepínačů "naklikejte" melodii, a poté sekvencer spusťte. Za běhu můžete měnit tempo, délku sekvence i jednotlivé tóny melodie. Nezapomeňte zapnout DSP.

Náš prototyp sekvenceru má řadu nedostatků: kód je příliš "rozlehlý", neumožňuje ukládání melodií, takže po zavření patche přijdeme o všechna nastavení v přepínačích i číselných boxech, změna délky sekvence se projeví okamžitě a ne až se začátkem cyklu, chybí informace o intenzitě noty atd.

Tyto nedokonalosti se pokusíme částečně vyřešit v nádledujících kapitolách.

schováváme_kód_do_subpatche

Při psaní rozsáhlejšího projektu v Pd se snadno stane, že se kód rozroste a není již dobře přehlédnutelný. Také při hraní na sestavený instrument spíš doceníme jasnost a kompaktnost uživatelského rozhraní, než to, že vidíme samotné ustrojení kódu (neplatí o livecodingu). Sekvencer, který jsme sestavili, je případ jednoduchého patche, a přesto zabírá celkem dost místa. Kdybychom v rozšiřování jeho kódu pokračovali v jedné ploše, záhy se ztratíme mezi objekty a kabely.

K ušetření místa a schování části kódů slouží tzv. subpatche. S jejich pomocí lze kód také strukturovat a zpřehlednit tím, že pro každý samostatný segment kódu se specifickou funkcí vyčleníme jeden subpatch.

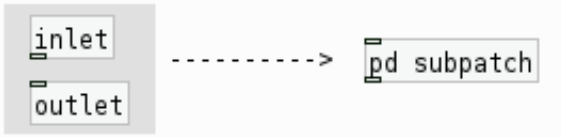
Subpatch vytvoříme tak, že do objektu vepíšeme nejprve "pd" a poté název subpatche. Otevře se nové okno, do kterého můžeme psát kód, nebo do něj vložit (CTRL + v) kód vyjmutý (CTRL + x) z hlavního (rodičovského) patche. Když okno subpatche zavřeme, uvidíme jen objekt a prostým kliknutím LTM na něj subpatch znovu otevřeme.

pd klikni_na_me

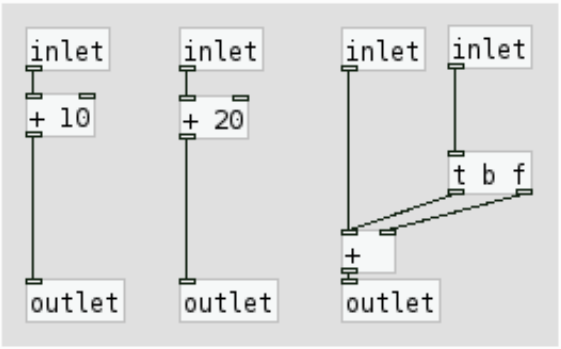
pd toto_je_subpatch

Vidíte, že tento subpatch nemá žádné vstupy ani výstupy.

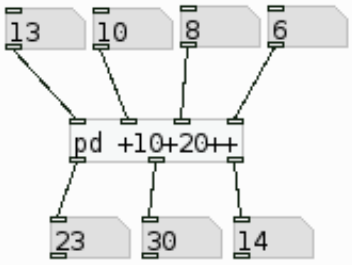
Aby subpatche mohly komunikovat s rodičovským patchem, ve kterém jsou "zahnízděny", nebo s dalšími subpatchi, pak je třeba jim nadefinovat vstupy a výstupy. To provedeme tak, že otevřeme subpatch a do něj umístíme objekty [inlet] a [outlet].



Mezi [inlet] a [outlet] pak můžeme umístit kód, který bude subpatch provádět.



kód umístěný
---------->
v subpatchi



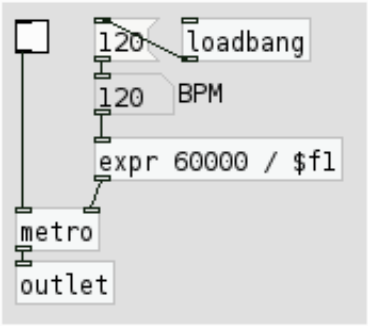
Vstupů a výstupů v subpatchi může být víc. Uvedený subpatch k prvnímu vstupu přičítá 10 a výsledek posílá do prvního výstupu. Ke druhému vstupu přičítá 20.Třetí a čtvrtý vstup sečte a pošle ho na třetí výstup.

Předchozí případovou studii sekvenceru bychom v případě členění patche do subpatchů asi rozdělili tak, jak je v něm naznačeno přerušovanou čarou. V jednom subpatchi by byl umístěn generátor bangů (metro a konvertor BPM na milisekundy), ve druhém by byl objekt [select] s přepínači a kódem zajišťujícím škálování na MIDI hodnoty. Konečně ve třetím by byl umístěn samostatný syntetizér.

GRAPH_ON_PARENT

Co když se ale stane to, že jsou uvnitř subpatche prvky, které chceme mít přístupné? V případě našeho sekvenceru by to byly přepínače nebo číselný box k zadávání BPM. Jak je zpřístupnit? Kromě řízení subpatche přes vstupy a výstupy lze v Pd vytvářet i grafická uživatelská rozhraní přímo v subpatchi. Takové prvky se v Pd terminologii nazývají Graph on Parent (GOP).

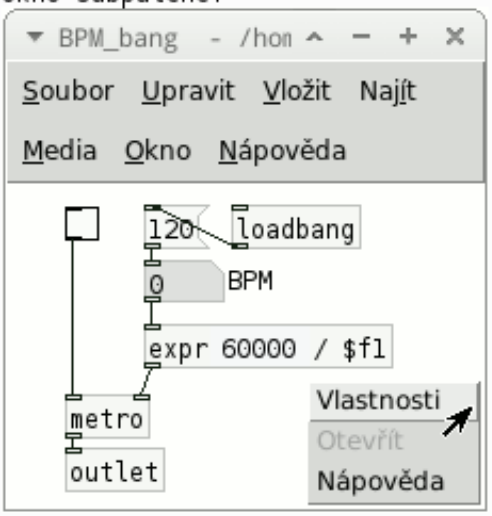
Ukážeme si "jak na to" na kódu z našeho sekvenceru - konkrétně na části, která generuje bangy. Nejprve vytvoříme subpatch se jménem BPM_bang a vložíme do něj následující kód:



U číselného boxu ještě v jeho vlastnostech nastavíme spodní a horní limit na 60 a 300.

----------> pd BPM_bang

okno subpatche:

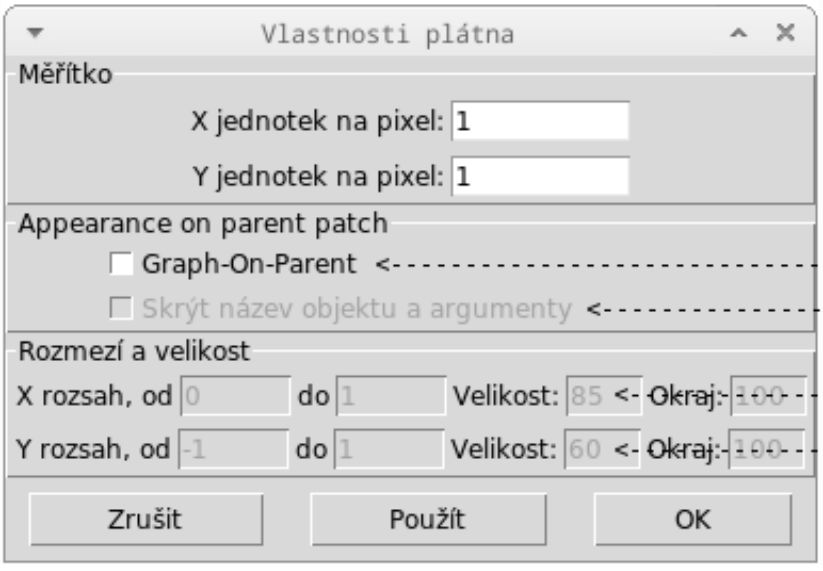


Subpatch otevřeme LTM a v jeho okně klikneme PTM na libovolné bílé místo. Z menu vybereme Vlastnosti, mělo by se otevřít okno s názvem "Vlastnosti plátna".

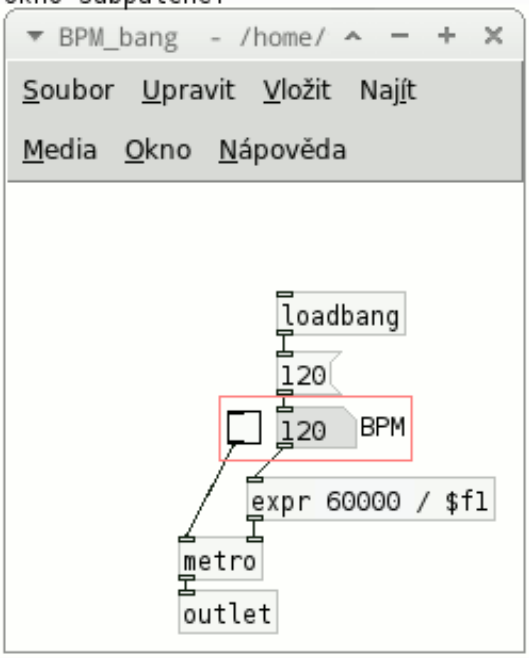
V okně "Vlastnosti plátna" vyškrtneme položku "Graph-On-parent" i "Skrýt název objektu...". Dále do políček "Velikost" zadáme hodnotu 90 (v X rozsahu) a 30 (v Y rozsahu).

Tím jsme nastavili subpatch tak, aby v poli o rozměrech 90 x 30 bodů zobrazoval grafické ovládací prvky jako bang, posuvník atd.

Nastavení potvrdíme kliknutím na OK.

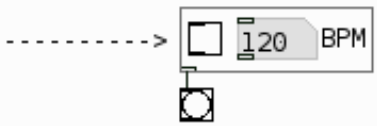


okno subpatche:



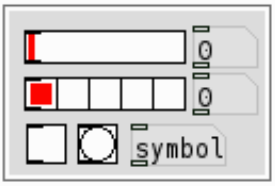
Po kliknutí na OK by se v okně subpatche BPM_bang měl zobrazit červený obdélník, do kterého přemístíme grafické uživatelské prvky tak, jak vidíme na vedlejším obrázku.

Když pak toto okno zavřeme, uvidíme v rodičovském patchi místo objektu [BPM_bang] obdélník s vypínačem a číselným boxem. Sláva, máme za sebou základy tvorby grafického uživatelského rozhraní v subpatchi.



Vyzkoušejte funkčnost kliknutím na vypínač.

Do plochy vyznačené červeným rámem je v subpatchi možné vkládat posuvníky, přepínače, vypínače, bangy, číselné boxy, symbol boxy, knoby, komentáře, plátno a ještě některé další grafické prvky, které jsme zatím nezmínili.

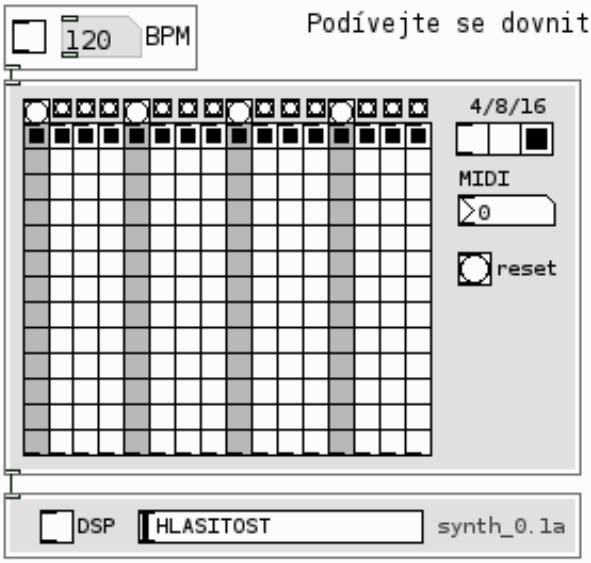


Podívejte se dovnitř.

Nebudeme zde opakovat celý proces vytváření GOP pro sekvencer a syntetizér. Jen na ukázku je zde výsledek. Vidíte, že se nám rozlehlost kódu podařilo zmenšit a uživatelské rozhraní vylepšit.

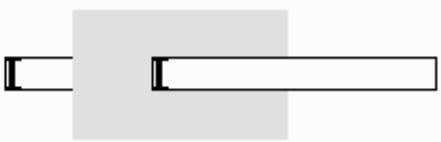
Nahlédněte prosím do subpatchů ve zdrojových kódech k této rukověti.

Další výhodou, o níž jsme se zatím nezmínili, je to, že subpatche jde lehce duplikovat. Vytvořit komplexní systém o několika sekvencerech je teď již otázkou zmáčknutí několika málo kláves.



KRYTÍ_PLÁTEN_-_Z_ORDER

Možná jste při práci s objektem plátno (Canvas) narazili na situaci, kdy vytvořené plátno překrývalo jiné grafické prvky. Pd si totiž pamatují, v jakém pořadí byly objekty vytvořeny, a v tom pořadí je i překrývá. Proto je třeba, abychom plátno vytvořili vždy dříve než další ovládací prvky.



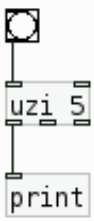
Nebo lze dané prvky jako posuvník aj. označit, vyjmout (CTRL + X) a následně opět vložit (CTRL + v). Budou tak "novější" než naposled vytvořné plátno - tzn. že budou viditelné.

slovní_zásoba_II

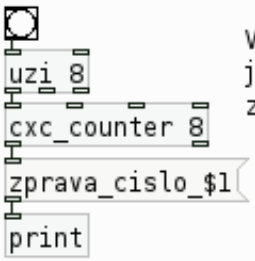
Ještě před tím než se pustíme do následující kapitoly, jež se bude týkat stavby systému, který by si pamatoval nastavení sekvenceru, budeme si muset rozšířit slovní zásobu o objekty a postupy, které k tomu budeme potřebovat.

[UZI]

[Uzi] je objekt, který pošle sérii bangů rychle za sebou. Počet bangů se odvíjí od argumentu. Budeme jím generovat řadu čísel pomocí počítadla a tyto hodnoty pak používat jako proměnné pro vytváření jedinečných zpráv.



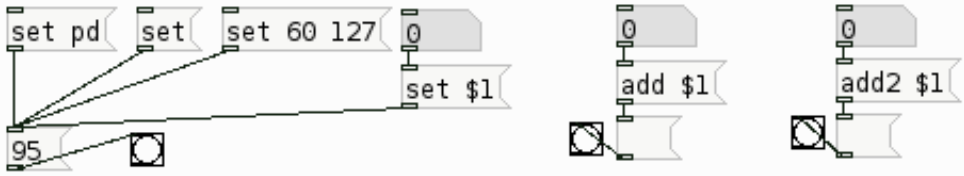
Pošle do konzole pět bangů.



Vytvoří osm jedinečných zpráv.

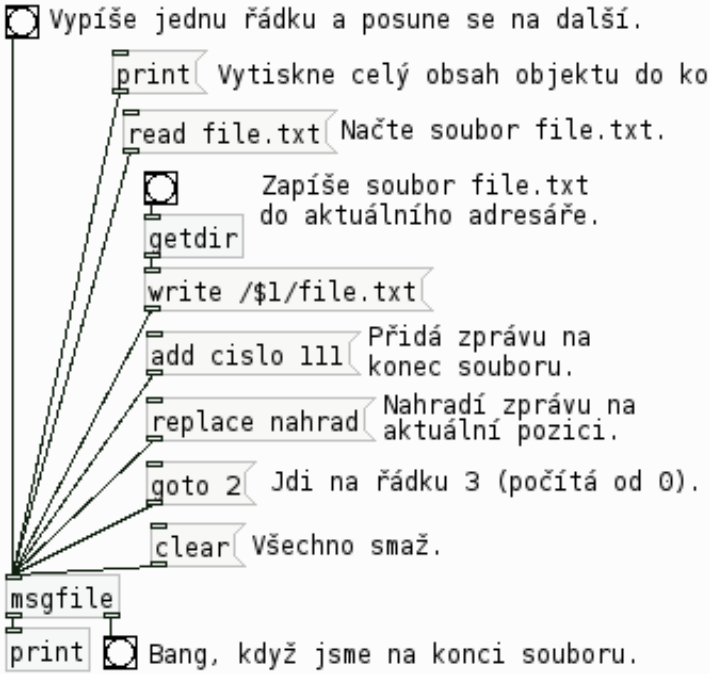
VYTVÁŘENÍ_ZPRÁV

Zpráva [set ...(nastaví obsah zprávy podle argumentu. Jenom [set(ji vymaže. [add ...(přidá argument do zprávy jako další zprávu (je oddělena středníkem) a [add2 ...(přidá argument jako nový atom. Vyzkoušejte si následující příklady. Všiměte si, že [set(, [add(, ani [add2(negenerují bang. Šlo by toho nějak využít?



[MSGFILE]

[msgfile] je objekt, který umí číst/zapisovat zprávy z/do souboru. Je rozšířenou verzí objektu [textfile] a obdobou objektu [coll] z Max/MSP.

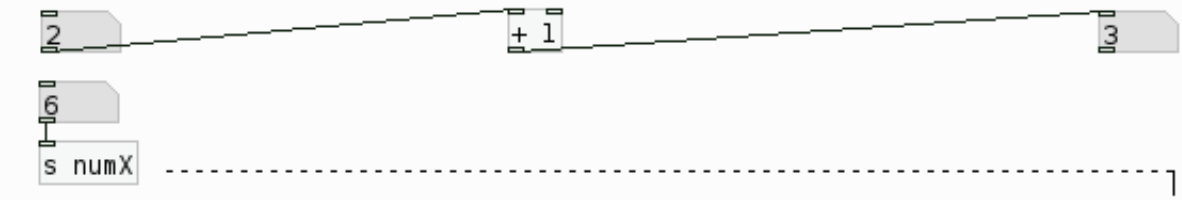


[msgfile] jde řídit spoustou dalších příkazů. Nám ale budou stačit tyto. Pokud by vás zajímaly další, tak se podívejte na jeho nápovědu.

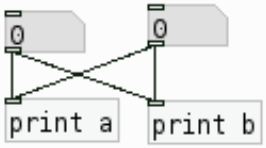
(PTM -> Nápověda)

[SEND]_A_[RECEIVE]

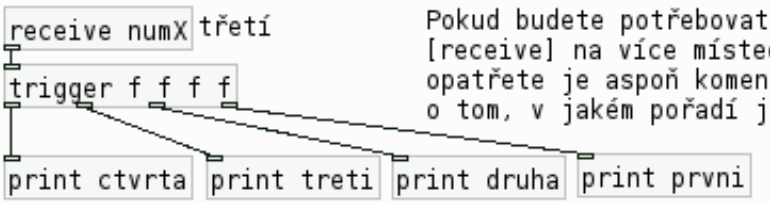
Když se nám stane, že budeme potřebovat poslat nějakou zprávu ve větším patchi z "jednoho konce na druhý", nemusíme nutně použít kabel, ale "bezdrátový" způsob přenosu. Doceníme ho také v situacích, kdy se nám patche rozrostou a budou vizuálně husté - přidáváním kabelů bychom čitelnost patche zhoršili. K bezdrátovému přenosu zpráv slouží objekty [send] a [receive].



V prvním případě jde o nehezkou ukázku toho, jak zprávu posílat "na dálku" po kabelu. Ve druhém případě posíláme hodnotu z číselného boxu do objektu [s numX], který ji přeposílá do objektu [r numX]. Vidíme, že [send] i [receive] mají stejný argument. To zaručuje, že zpráva doteče tam, kam má. Číst ji můžeme na více místech, což je ale často příčinou problému nejasného chování kódu. Vzpomeňte si na případ, kdy z jednoho objektu vede více kabelů do různých míst (místo takového zápisu používáme objekt [trigger]!) - zpráva doteče nejprve na to místo, kam jsme jako první zapojili kabel. Objekt [receive] se ale chová opačně - první zpráva bezdrátově doteče na objekt [receive], jenž jsme vytvořili v patchi jako poslední. V našem případě je to [r numX] opatřený komentářem "třetí".



Tomuto zmatení v pořadí posílání zpráv se vyhneme opět objektem [trigger]. Pokud vám to situace umožní, doporučuji použít jeden [receive] a ten pak rozvětvit s pomocí objektu [trigger], takže v pořadí provádění událostí bude jasno.

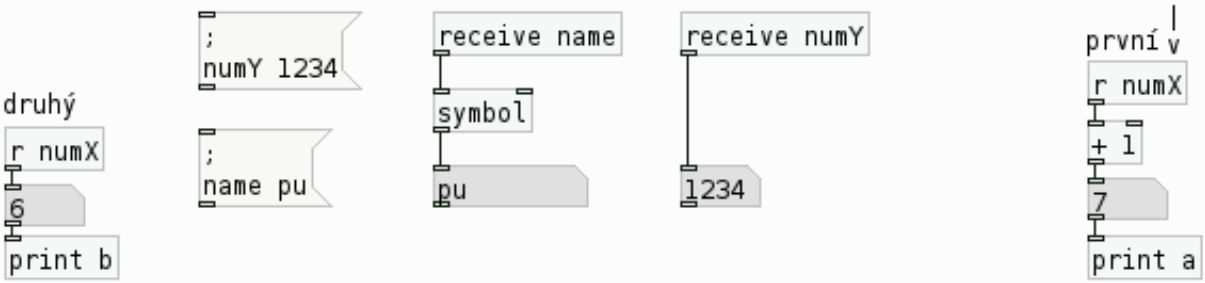


Pokud budete potřebovat mít [receive] na více místech v patchi, opatřete je aspoň komentářem o tom, v jakém pořadí jste je vytvořili.

SEND_ZE_ZPRÁVY

Alternativním způsobem, jak poslat číslo nebo symbol bezdrátově, je zapsat je do zprávy, která začíná středníkem, pak následuje jméno "místa doručení", a třetí je samotný obsah, který chceme poslat.

Podobného druhu je i zpráva, kterou již známe: [; pd dsp 1(. Neposílá ale zprávu na nějaký [receive], ale přímo do Pd.



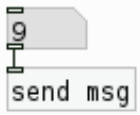
GLOBÁLNÍ_A_LOKÁLNÍ

Toto téma by si zasloužilo samostatnou velkou kapitolu. Nicméně problematika globálního a lokálního posílání určitých zpráv s objekty [send] a [receive] bezprostředně souvisí, takže ji uvádíme již nyní. Pojmů jako je "abstrakce" se nelekejte, vysvětlíme si je později. Možná vám zde uvedená látka přijde neaplikovatelná, její význam ale zhodnotíme při konstrukci jakéhokoliv nástroje, který budeme chtít v patchi mít víckrát než jednou, nebo při vytváření systémů sloužících k uchování hodnot (presety).

Vlastností objektu [send] je to, že umí posílat zprávy buď globálně nebo lokálně. Co to znamená? Představme si každý aktuálně otevřený patch jako jeden byt v domě. A množinu všech patchů jako dům.

Když používáme např. [send msg], pak do jakéhokoliv otevřeného patche (a je to případ i subpatchů a abstrakcí), ve kterém bude přítomný objekt [receive msg], tato zpráva doputuje. Je to jako bychom ji dávali vědět celému domu.

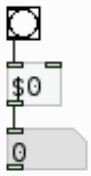
Pokud budeme chtít "mluvit pouze v jednom bytě", tj. poslat zprávu pouze v rámci jednoho patche (a jeho subpatchů) a nebo v rámci abstrakce, použijeme před "jménem destinace" předponu "$0-".



Globální vysílač.

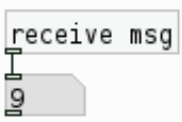
Lokální vysílač.

$0 je zvláštní symbol, je to vlastně identifikátor, díky kterému Pd dokážou rozlišit např. dva stejné a současně otevřené patche. Obvykle nabývá hodnot kolem 1000 a výš. Jakou hodnotu má identifikátor aktuálně otevřeného patche, zjistíme následujícím drobným kódem.

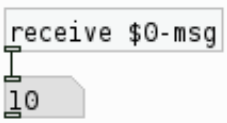


Vytvořte nový patch (CTRL + n) a do něj vložte tento kód. Poté inicializujte bangy jak v tomto, tak v nově vytvořeném patchi, a porovnejte je. Identifikátor bude mít pro každý patch jinou hodnotu.

Nyní zkuste z obou [send]ů nahoře poslat nějakou hodnotu. Uvidíte, že v obou případech dotečou do objektů [receive] dole. V prvním případě ale proto, že je hodnota poslána globálně. V druhém pak proto, že je poslána lokálně a posíláme ji v rámci jednoho patche (jednoho bytu), takže identifikátor $0 má stejnou hodnotu jak pro [send], tak pro [receive].



Globální přijímač.

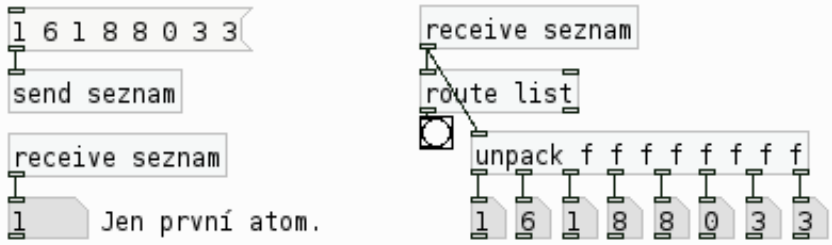


Lokální přijímač.

Pakliže objekty [receive msg] a [receive $0-msg] umístíme do nově vytvořeného patche a znovu z obou [send]ů pošleme hodnoty, v novém patchi dotečou jen do [receive msg]. [send msg] posílá totiž zprávy globálně - do všech otevřených patchů. V novém patchi má identifikátor $0 jinou hodnotu než v tomto, proto mu zpráva z tohoto patche nebude doručena. Je to úplně jiný byt.

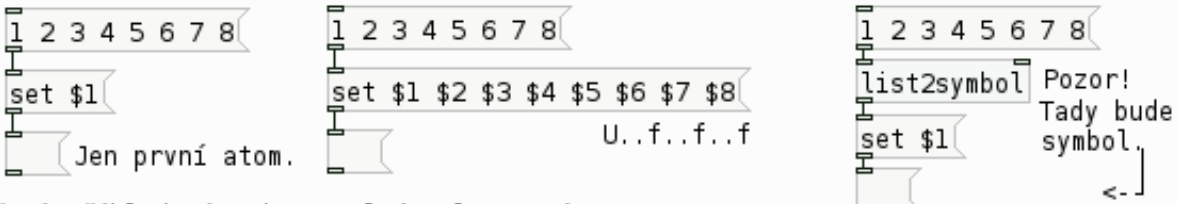
[SEND]_[RECEIVE]_A_SEZNAMY

Objekty [send] a [receive] nejsou omezeny jen na posílání atomických zpráv, ale umí posílat i seznamy. Pouze si musíme dát pozor na to, že když seznam vede do objektu, který čeká na atom (číselný box, symbol box, [f] atd.), je přijata pouze první hodnota a ostatní jsou ignorovány. To ostatně platí ale i pro obyčejné posílání seznamu po drátě.



[LIST2SYMBOL]_-_[SYMBOL2LIST]

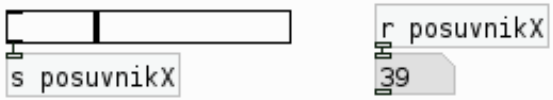
[list2symbol] a [symbol2list] slouží ke konverzi selektorů, jak naznačují již názvy objektů. Oba jsou součástí knihovny zexy. První využijeme v situaci, kdy si budeme chtít ulehčit práci se psaním proměnných ve zprávách. Zjevnější to snad bude po vyzkoušení následujících příkladů.



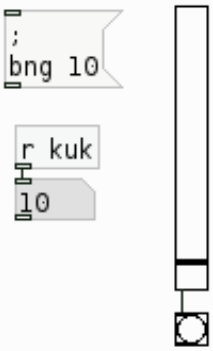
Třetí příklad nám sice vyplní celou zprávu "jakoby" seznamem, když ale na vyplněnou zprávu klikneme, objektem [route list] zjistíme, že o žádný seznam nejde!!
Abychom nedefinovaný selektor opět změnili na seznam, proženeme zprávu objektem [list] a [symbol2list].

[SEND]_[RECEIVE]_A_GUI

Grafické uživatelské prvky jako tlačítko, vypínač, posuvník aj. můžou také vysílat a přijímat zprávy bezdrátově. Přitom ale není potřeba to dělat takto:



Stačí na ně kliknout PTM (z menu zvolíme Vlastnosti) a do kolonek "Poslat/Přijmout symbol" zadat patřičné názvy. Grafické uživatelské prvky, které vysílají nebo přijímají zprávy, poznáme podle toho, že nemají zvýrazněný vstup a výstup. V nastavení posuvníku vpravo jsme do kolonky "Poslat symbol" zadali "kuk" a do "Přijmout symbol" "bng".



Do těchto kolonek lze samozřejmě též před název "místa doručení" zadat identifikátor $0-. Daný grafický uživatelský prvek pak bude posílat zprávu dostupnou pouze lokálně. Této vlastnosti využijeme zejména při tvorbě presetů a abstrakcí s grafickým uživatelským rozhraním.

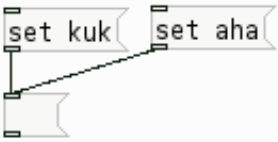
paměť_patche_-_presety

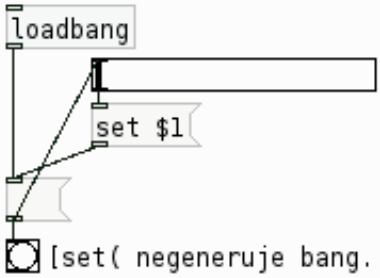
Dalším nedostatkem sekvenceru, který jsme naprogramovali, je to, že "zapomíná" svá nastavení. Není to jen případ tohoto patche, ale obecně všech programů napsaných v Pd. Posuvníky, přepínače, číselné boxy atd. vždy po zavření zapomínají své hodnoty. Zůstávají jen údaje zapsané ve zprávách. Obzvláště začátečníky, kteří přijdou o některé ze svých pracně vytvořených nastavení, může tento neduh Pd odradit od dalšího prozkoumávání a studia.

V této kapitole se podíváme na elementární metody, jak přimět patche k tomu, aby si své nastavení pamatovaly.

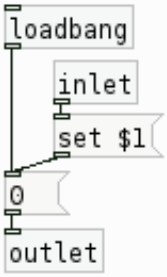
[LOADBANG]_-_[SET(_-_ZPRÁVA

Nejjednodušším a zároveň nejtěžkopádnějším způsobem, jak vytvořit jednoduchý presetový systém (tj. systém uchovávající daná nastavení), spočívá v kódu sestaveného z objektu [loadbang], ovládacího prvku, jehož hodnotu chceme uchovat, a dvou zpráv. Vypadá takto:





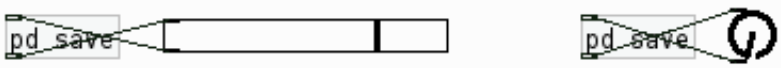
A toto je verze pro subpatch:



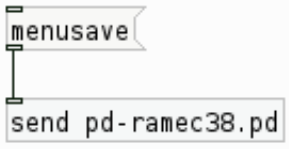
Zopakujme si: Příkaz [set(zapsaný ve zprávě a doplněný o argument zapíše do jiné zprávy tento argument, ať už jde o číslo nebo symbol. Když nahradíme argument proměnnou $1, [set(do zprávy zapíše vždy danou proměnnou, která mu přiteče na vstup. Tato akce přitom ve zprávě, do níž zapisujeme, negeneruje bang na výstupu, takže po kabelu vedoucím zpátky do posuvníku při změně nic neteče a hlášku upozorňující nás na přetečení zásobníku v konzoli neuvidíme.

Další část kódu je již elementární: do zprávy, ve které je uložena hodnota, vede kabel z objektu [loadbang], takže vždy po otevření daného patche do ní bude poslán bang a hodnota ze zprávy doteče do posuvníku.

Verzi kódu pro subpatch umístíme do subpatche [pd save] a překříženě spojíme s daným grafickým uživatelským prvkem. Ejhle, máme k dispozici primitivní presetový systém.



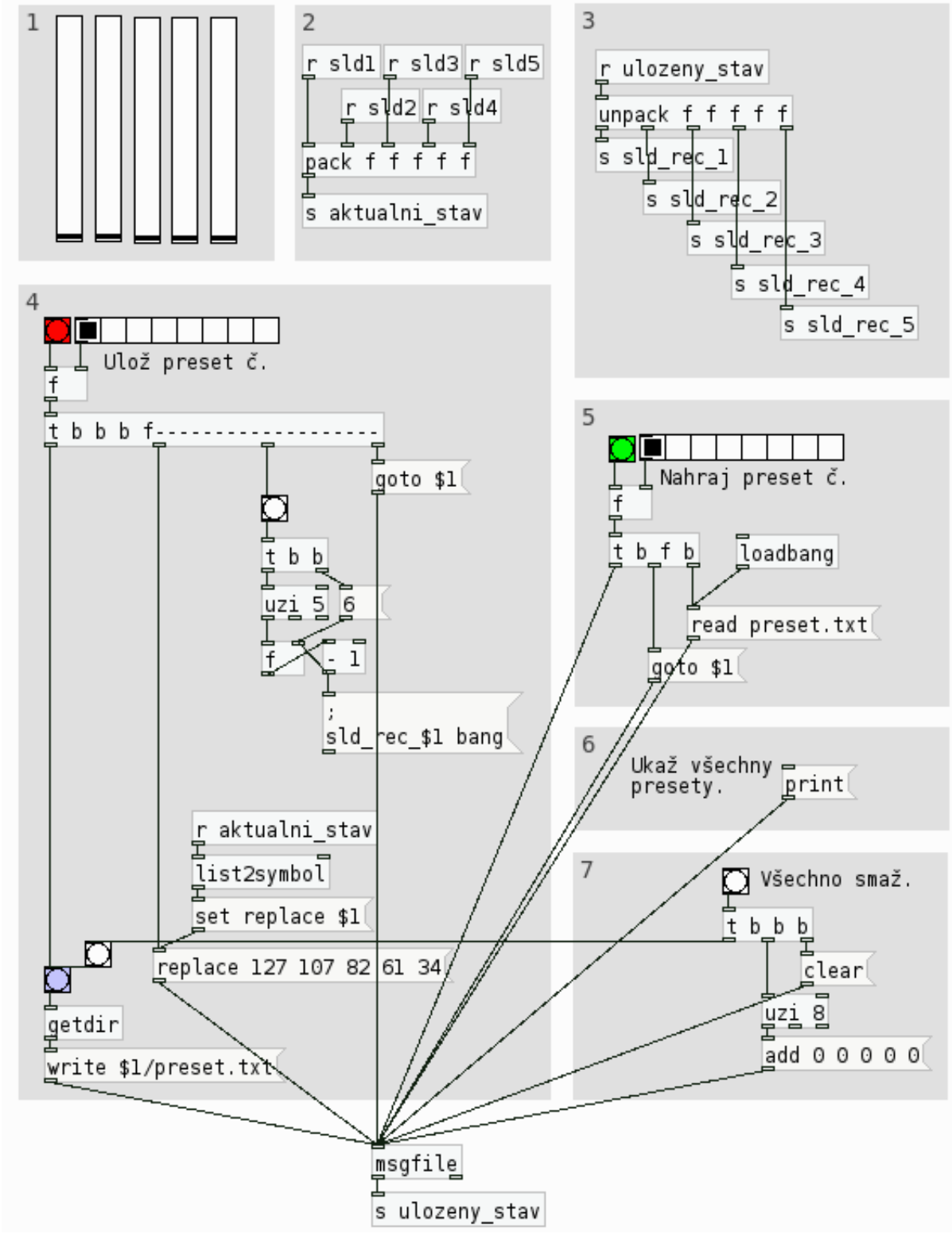
Poslední krok, na který nesmíme zapomenout, je uložit patch před tím, než ho zavřeme. Jinak by se hodnota nastavená ve zprávě pochopitelně neuložila. Buď patch uložíme "ručně" přes klávesovou zkratku CTRL + s, nebo s pomocí krátkého kódu:



Objekt [send] má parametr začínající v tomto případě vždy předponou "pd-" následovanou jménem patche, ve kterém jsme.

UKLÁDÁME_STAVY_DO_SOUBORU

Na této stránce vidíme trochu míň těžkopádný, ale přesto ne zcela optimální patch, jenž ukládá nastavení do souboru preset.txt. Pokuste se sami analyzovat, pochopit a vysvětlit jednotlivě očíslované segmenty kódu. Pokud vám něco nebude jasné, zkuste se vrátit k předchozím kapitolám nebo nahédnout do nápovědy k jednotlivým objektům.
K důkladnému prozkoumání a ověření funkčnosti patche je samozřejmě nutné se podívat přímo na jeho zdrojový kód a otestovat ho za běhu v Pd. Jsou v něm nějaké chyby? Šel by kód ještě optimalizovat?



ANALÝZA_PŘEDCHOZÍHO

Věřím, že vám "četba" a prozkoumávání činnosti předchozího patche v Pd nedělala potíže. Analýzou patchů se toho sami můžete hodně naučit. Nuže, nyní si rozebereme segment po segmentu.

Segment 1 a 2: Pět posuvníků, které mají ve vlastnostech nastaveno posílání zpráv se jménem sld1..5 a přijímání zpráv sld_rec_1..5. Jediné místo, kam hodnoty z posuvníků bezdrátově putují, je v segmentu č. 2, kde z nich objekt [pack] vytvoří seznam a ten pak putuje do kódu v Segmentu 4.

Segment 3: Objekt [unpack] rozbalí seznam, který mu bude bezdrátově doručen z objektu [msgfile]. Jednotlivé hodnoty se rovnou bezdrátově přenášejí do posuvníků, které přijímají zprávu sld_rec_1..5).

Segment 4: Zde se řeší ukládání stavu všech posuvníků do souboru. Bang pošle číslo, které je v objektu [f] nastaveno přepínačem na jednu z hodnot v rozsahu 0..7 - to jsou čísla presetů. [trigger] provede nejprve to, že v souboru nastaví danou řádku (v souboru preset.txt je 8 řádků a každý je vyčleněn pro jeden preset). Druhá akce [trigger]u do všech posuvníků pošle bang, čímž se z nich "protlačí ven" aktuální hodnoty. Vytvoří se z nich seznam (viz Segment 2) a ten je bezdrátově poslán ke třetí akci, kterou [trigger] vykonává. Seznam "aktualni_stav" je nejprve konvertován na symbol objektem [list2symbol]. To z důvodu, abychom v následující zprávě [set replace $1(nemuseli vypisovat všechny proměnné. Když je posuvníků 5, není to ještě takový problém, pokud by ale preset měl mít např. 32 položek, při psaní bychom to již pocítili. V [set replace $1(se tedy nahradí $1 hodnotami z posuvníků a třetí bang pošle zprávu k zapsání na daný řádek. Zpráva je sice uložena uvnitř objektu [msgfile], ale ještě není zapsána fyzicky do souboru. To provádí poslední - čtvrtý bang v [trigger]u. Objekt [getdit] na výstupu dává cestu k aktuálnímu adresáři, jež se pak ve zprávě spojí s názvem souboru - tedy s textem "/preset.txt".

Segment 5: Podobně jako u segmentu, který řeší nahrávání, i zde posíláme bang, který "protlačí" hodnotu s číslem presetu - ale tentokrát ho budeme chtít nahrát. Vidíme, že soubor preset.txt je automaticky objektem [msgfile] nahrán vždy před čtením řádku s presetem. To je potřeba provést z důvodu odstranění selektoru symbol, který si jinak [msgfile] pamatuje. Ve zprávě [set replace $1(jsme totiž selektor seznamu změnili na symbol, abychom ho mohli pohodlně do zprávy vložit (viz Segment 4). Soubor preset.txt se také vždy automaticky nahrává po otevření patche, což zajišťuje [loadbang]. Soubor je tedy nejprve nahrán - jsou v něm pouze řádky s čísly, takže každý řádek má automaticky selektor list. Poté [trigger] inicializuje příkaz [goto $1(, čímž se nastavíme na řádek presetu, jenž chceme číst, a konečně poslední bang jej z objektu [msgfile] "protlačí" ven. Seznam čísel s hodnotami posuvníků je bezdrátově poslán ke zpracování do objektu [r ulozeny_stav] (viz Segment 3), který ho rozbalí na jednotlivé hodnoty a ty pošle do posuvníků.

Segment 6: Příkazem [print(poslaným do [msgfile] se můžeme snadno informovat o jeho aktuálním obsahu a stavu jednotlivých presetů.

Segment 7: Zde bang rozvětvený [trigger]em provádí tři akce. Nejprve kompletně smaže obsah, který si [msgfile] pamatuje, poté příkazem [add 0 0 0 0 0(vytvoří osm řádků, na kterých budou jen nuly a nakonec tento obsah uloží do souboru preset.txt. Bang posíláme po drátě části kódu, který již máme v Segmentu 4 - nemusíme ho duplikovat.

Tolik k vysvětlení kódu. Řešení ukládání stavů, jaké jsme předvedli, má oproti první těžkopádné verzi tu výhodu, že si patch "nezahustíme" řadou objektů [pd save], a též v tom, že je možné uložit libovolný počet presetů. Problém tohoto řešení pak spočívá v tom, že kdybychom jej implementovali např. do našeho šestnáctistopého sekvenceru, bude fungovat pouze s jeho jednou instancí. To je způsobeno tím, že hodnoty, které bezdrátově posíláme z/do posuvníků, a také další hodnoty proměnných mají globální platnost. Ve více spuštěných instancích sekvenceru by tedy názvy proměnných byly ve vzájemném konfliktu.

Lepší řešení spočívá v použití abstrakcí a indentifikátoru $0-, díky kterým bychom zaručili bezchybnou funkčnost presetových systémů ve více otevřených sekvencerech. Abstrakcím se ještě budeme věnovat později. Navzdory těmto omezením je náš příklad snad dobře aplikovatelný ve středně velkých projektech.

Pokud máte ještě dost sil k průzkumu, doporučuji se podívat na nápovědu k objektu [coll] z knihovny cyclone, kterým by šel presetový systém také pojednat.

coll

PRESETOVÉ_SYSTÉMY_Z_KOMUNITY

Za dobu existence Pd se v komunitě uživatelů přirozeně objevila poptávka po presetových systémech. V "dřevních dobách" vyvinul Frank Barknecht systém RRADical, který však již není ve vývoji. Frank jej nahradil lepší a uživatelsky přívětivější verzí, která se jmenuje Stupidsupersimplistic State Saving ADVANCED, neboli Sssad. Informace o projektu najdete na uvedené stránce.

Sssad

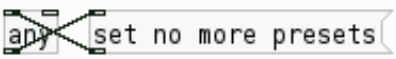
K pochopení a rozběhnutí Sssad je ale přeci jen třeba již určité uživatelské a programátorské zručnosti (patche jsou ke stažení pouze za pomoci správce zdrojových kódu svn).

Dalším pokusem o řešení problémů s presety je externí knihovna jménem pool od Thomase Grilla, která je ke stažení na jeho domovské stránce. Pool není součástí Pd-extended.

grrrr.org

Asi posledním pokusem o jednoduchý a relativně přívětivý presetový systém je kolekce patchů, jejímž autorem je Mike Moser-Booth (id Maelstorm na Pd Fóru). Ke stažení je, spolu s řadou dalších užitečných patchů, na GitHubu. Po rozbalení se ev. podívejte na soubor s názvem my.hero.mmb-help.pd, ve kterém je popis tohoto systému včetně ukázek.

mmb GitHub



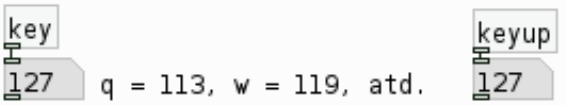
vstupy_a_interaktivita

Po teoreticky trochu náročnějších kapitolách si nyní odpočineme při prozkoumávání vstupních zařízení. Většina interakce, kterou jsme doposud v Pd provozovali, se týkala inicializace bangů, posuvníků nebo zpráv klikáním tlačítka myši. Klávesnice a myš jsou přitom pouze dva základní příklady z celé široké škály tzv. HID (Human Interface Devices) zařízení.

Přes USB jsou k počítači připojitelné různé MIDI kontrolery, tablety, joysticky, pady, WII, nebo třeba i senzory, které snímají srdeční tep a EEG. Pakliže jste zběhlí v základech elektrotechniky, můžete si s pomocí vývojové platformy Arduino postavit vstupní zařízení dle vlastního přání.

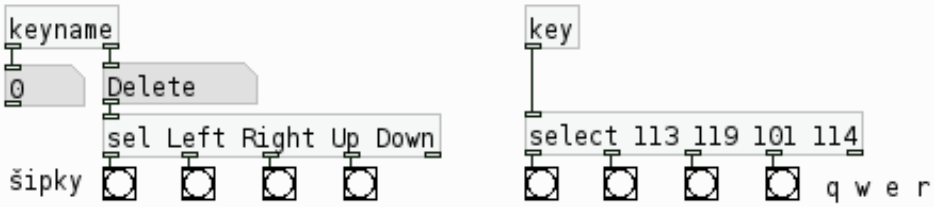
KLÁVESNICE

Objekt [key] má pouze jeden výstup, na který po zmáčknutí klávesy posílá číslo odpovídající její ASCII hodnotě v dekadickém zápisu. [keyup] dělá to samé, ale hodnotu posílá až po puštění klávesy.



Seznam všech hodnot najdete zde.

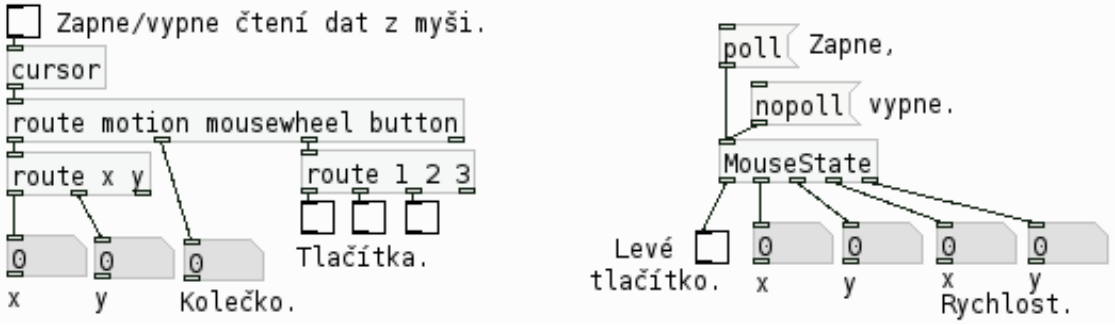
[keyname] má výstupy dva, na levý posílá 1/0 podle toho, zda klávesa je nebo není zmáčknuta, a na pravý výstup pak samotný znak (se selektorem symbol).



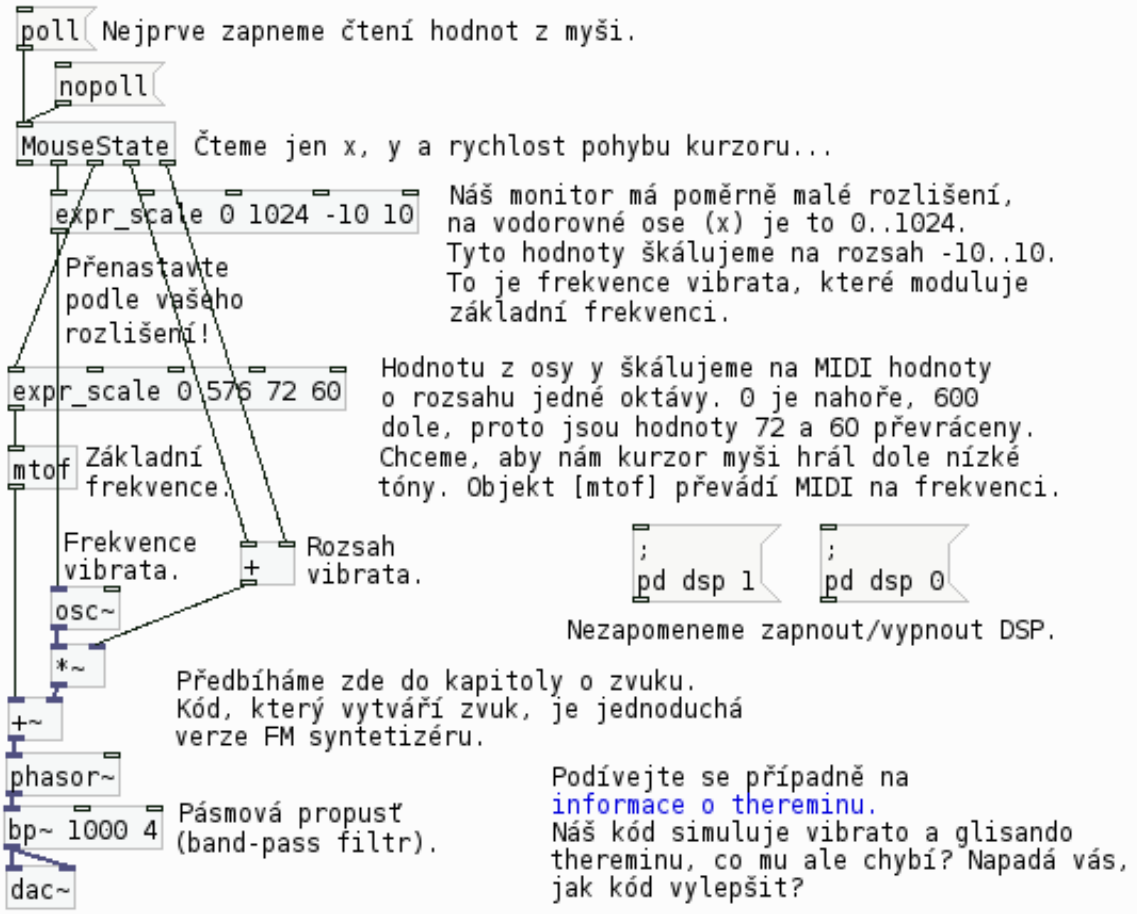
Objekty [key], [keyname] a [keyup] budeme nejčastěji kombinovat s objektem [select], jímž jednotlivé klávesy přiřadíme bangům, a dále pak nějakým dalším událostem v kódu, zahrání tónu, spuštění animace atd.

MYŠ

Informace týkající se polohy kurzoru a stavu tlačítek myši čteme v Pd buď objektem [cursor] nebo objektem [MouseState]. Nepoužíváme je zároveň, protože pak jsou některé jejich funkce navzájem v konfliktu (např. čtení stavu tlačítek). [MouseState] má oproti komplexnějšímu [cursoru] jedinou výhodu, a to je, že na posledních dvou výstupech nám dává aktuální rychlost kurzoru na ose X a Y.

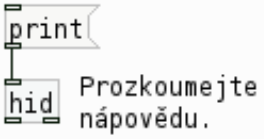


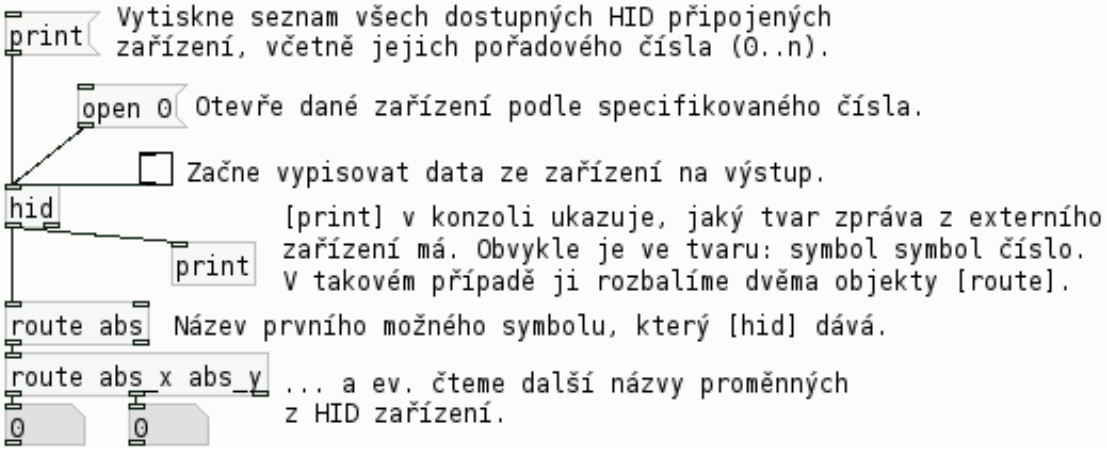
Jakou maximální hodnotu x a y budou mít se odvíjí od rozlišení monitoru. V následující malé případové studii si ukážeme, jak využít čtení hodnot z myši k ovládání hudebního nástroje - postavíme si primitivní "theremin".



HID_/_HIDIN

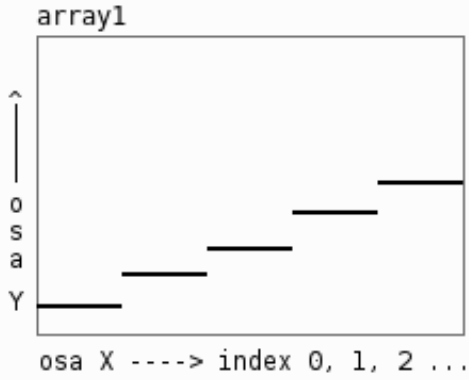
Ke zpracování dat z dalších HID zařízení kromě klávesnice a myši vám poslouží objekty [hid] (Linux a MacOS) a [hidin] (Windows). Zařízení k počítači připojujeme ještě před spuštěním Pd. Jinak totiž obvykle Pd zařízení nedetekují, nebo může dojít k jejich "zamrznutí". Další problém spočívá v ošetření práv k HID zařízením. Přímý přístup k nim má obvykle jen administrátor systému, proto pokud je chceme používat, je třeba Pd spustit s administrátorským oprávněním (nebo upravit práva umožňující uživateli číst z HID zařízení).



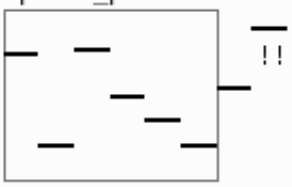


pole

Pole je datová struktura určitého počtu prvků, které jsou označeny indexem. Může v něm být zapsána notová partitura, zvukový soubor, či jakýkoliv jiný typ dat. Pole je v Pd zároveň grafický a interaktivní prvek. V performativním módu můžeme kliknout LTM na levý okraj libovolného prvku a měnit jeho polohu na ose Y. Do patche pole vložíme přes menu Vložit/Put -> Pole/Array. V nově otevřeném okně zadáváme unikátní jméno pole, počet jeho prvků, typ zobrazení a to, jestli si má pole pamatovat svůj obsah i po zavření patche.



Vedlejší pole má pět prvků, které se indexují od nuly. Hodnotu indexu představují sloupce (osa X) a číselná hodnota jednotlivých prvků se vynáší na osu Y. Indexy při takto nízkém počtu bezprostředně vidíme. Jakou mají prvky ale hodnotu na ose Y, s přesností určit nedokážeme. Když klikneme do pole PTM a vybereme Vlastnosti, otevřou se dvě okna: jedno, které již známe - s názvem "Vlastnosti pole" - v něm nastavujeme jméno atd. Druhé okno má titulek "Vlastnosti plátna" a nastavujeme v něm jednak grafický rozměr pole (položka Velikost) a dále rozsah, ve kterém se budou pohybovat hodnoty na ose X a Y. Pro osu Y je přednastaven rozsah -1..1. Pokud počet prvků pole, jenž jsme zadali v okně "Vlastnosti pole", není totožný s rozsahem pole na ose X v okně "Vlastnosti plátna", pak budou prvky buď z grafického ohraničení přetékat, nebo v něm bude "bílé místo". Rozsah na ose X by měl odpovídat počtu prvků.

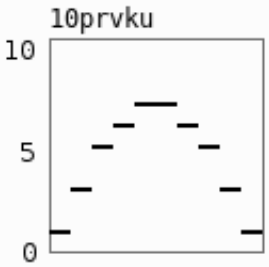


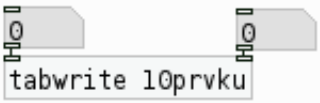
Bez pole bychom v Pd nemohli dost dobře pracovat se zvukovými vzorky, nepostavili bychom sampler ani tzv. wavetable syntetizér nebo "analogový" sekvencer.

Dodejme ještě, že v Pd musí mít každé pole jedinečný název! Pokud bychom náhodou otevřeli tento patch dvakrát, nevěděla by Pd, ze kterého pole jménem "array1" mají číst nebo zapisovat. V konzoli bychom na tuto chybu byli upozorněni hláškou: "warning: array1: multiply defined.

ČTENÍ_A_ZÁPIS_HODNOT

Pokud budeme do pole chtít zadat sekvenci prvků přísněji než myší, pak využijeme objektu [tabwrite]. Na pravý vstup zadáváme číslo indexu (poloha na ose X). První prvek má index 0.Na pravý vstup pak zadáváme samotnou hodnotu prvku. U [tabwrite] nesmíme zapomenout specifikovat argument, jímž je jméno pole, do kterého zapisujeme.

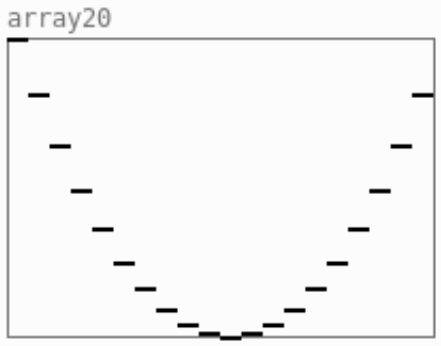




Ve vlastnostech číselných boxů jsme nastavili spodní a horní limit podle vlastností pole s názvem "10prvku".

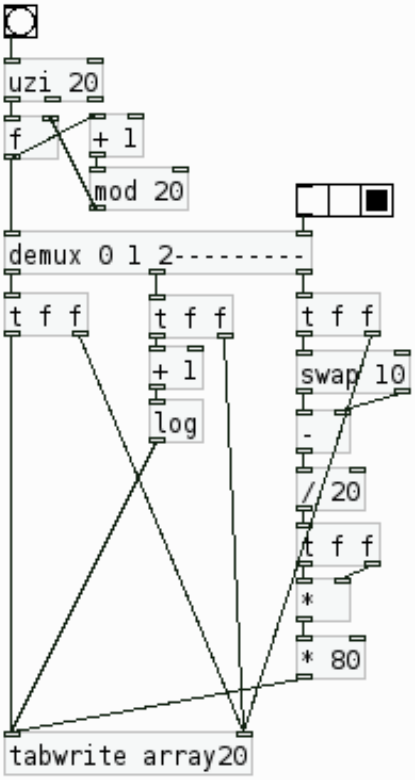
Nejprve nastavíme pasivní vstup na požadovaný index, a poté do aktivního vstupu pošleme hodnotu - prvek na indexu se na ni nastaví. Pole "10prvku" má, jak napovídá jeho název, 10 prvků. Na ose Y má zadán rozsah 10..0 a na ose X 0..10.

Jiným způsobem, jak pole vyplnit, je poslat do něj zprávu se seznamem. Zpráva musí mít následující formát: začíná středníkem, pak následuje název pole, do kterého seznam posíláme, poslední složkou zprávy je pak seznam čísel, jehož první hodnota označuje index, od kterého zapisujeme, a pak následují již samotné hodnoty.



Zpráva nahoře pošle do pole s názvem "array20" od indexu 0 seznam dvaceti hodnot začínajících 3 2 1 5 ...

A tato zpráva nastaví na všech indexech konstantu 0.

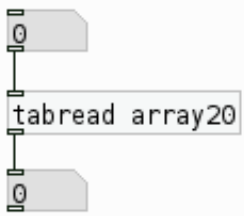


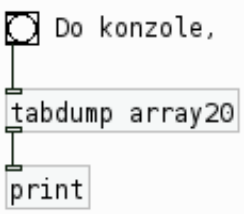
V řadě případů je jednodušší než zadávat hodnoty ručně si postavit patch, jenž za nás práci vyplnění pole provede. "array20" má 20 prvků, které můžou nabývat hodnotu v rozsahu 20..0 (podívejte se do vlastností tohoto pole).

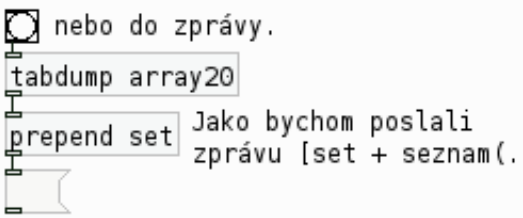
Počítadlem vytváříme hodnoty 0..20 a používáme je k indexování prvků pole. Podle hodnoty v přepínači napojeném na objekt [demux] se pak provede jedna ze tří větví kódu. Nejprve hodnotu posíláme objektem [t f f] do pasivního vstupu [tabwrite array20], takže nastavíme index prvku, a pak teprve jeho hodnotu na aktivním vstupu.

První větev patche do pole vykresluje lineární posloupnost, druhá logaritmickou a třetí parabolu.

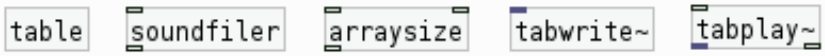
K získání hodnoty prvku na určitém indexu používáme objekt [tabread] s argumentem odpovídajícím názvu pole, ze kterého chceme číst. Na vstup [tebread] posíláme číslo indexu a na výstupu nám dá hodnotu daného prvku. Objekt [tabdump] zase, po obdržení bangu, vypíše všechny prvky jako seznam.







Problematiku polí bychom ještě mohli dalekosáhle prozkoumávat. Bude ale lepší, když se k ní vrátíme v souvislosti s nějakým konkrétním příkladem. Pokud byste chtěli provádět detailnější průzkum již nyní, podívejte se na nápovědu k následujícím objektům.



Nebo lze k samostudiu doporučit opět referenční soubory (Nápověda -> Pd Help Browser -> Pure Data -> 5.reference), tentokrát se jménem all_about_arrays.pd.

Až probereme látku týkající se vytváření abstrakcí, vrátíme se k polím a spojíme naše znalosti s tím, co víme identifikátoru $0-. Bude se nám to hodit při stavbě sampleru.

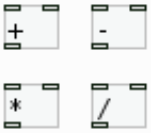
CVIČENÍ:

- pokuste se o sestrojení sekvenceru, který by četl údaje objektem [tabread] z pole
- pokuste se o sestrojení jednoduchého presetového systému, který by uchovával hodnoty v polích

trocha_matematiky

Se základními matematickými operacemi jsme se v Pd již seznámili. Nyní si znalosti z této oblasti trochu rozšíříme o ty, které v běžné programátorské praxi s velkou pravděpodobností budete potřebovat.

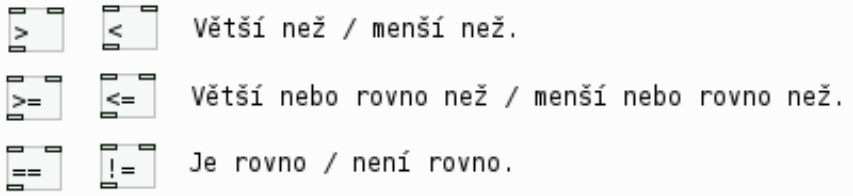
Nejprve něco málo k terminologii: zápisu "a + b" říkáme výraz. "a" a "b" jsou operandy a "+" je operátor - tedy to, co s operandy něco dělá. Operátory se dělí podle počtu operandů, se kterými pracují. Operátory [+], [-], [*], [/] a [mod] patří mezi tzv. binární operátory, protože pracují se dvěma operandy.



SROVNÁVÁME

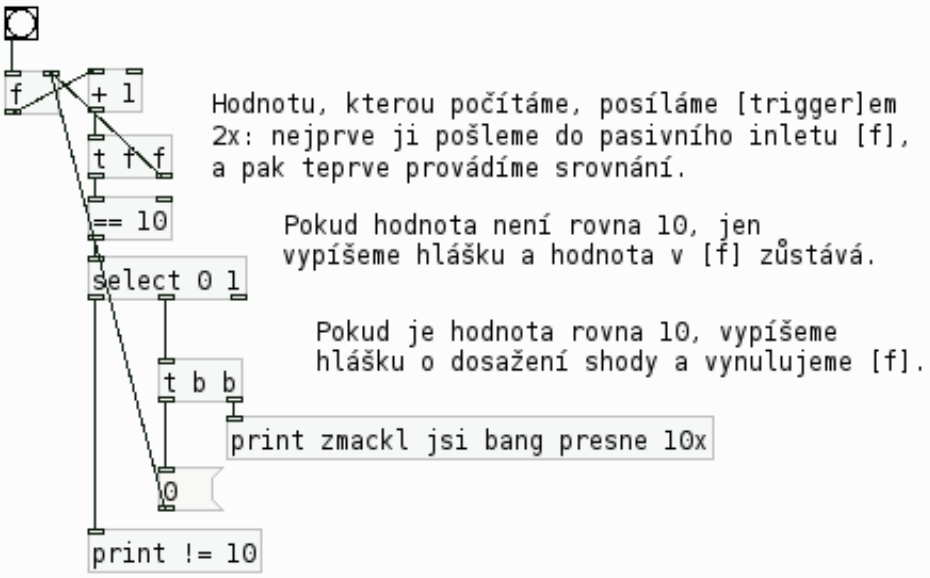
Mezi další binární operátory patří ty, se kterými můžeme dva operandy porovnat a zjistit, jaký je mezi nimi vztah. Na rozdíl od operátorů [+], [-] atd. jsou jejich výstupem pouze hodnoty 1 nebo 0, neboli "true" a "false". Ty označují, jestli je daný výraz pravdivý nebo nepravdivý.

Mezi srovnávací neboli relační operátory patří tyto:



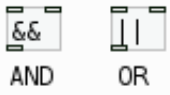
Srovnávací operátory využijeme v řadě situací: např. když budeme pořebovat zjistit, jestli bylo dané tlačítko zmáčknuto přesně 10x (k tomu můžeme využít i binární operátor [mod]), zda jsme myší "ujeli" vzdálenost větší než jeden kilometr, nebo jestli se dva body nacházejí na stejné souřadnici atd. Obecně jde o situace, nad kterými si můžeme položit otázku, na níž odpovídáme ano nebo ne.

Po vyhodnocení výrazu s binárním srovnávacím operátorem pak obvykle v kódu následuje objekt [select 0 1], kterým spouštíme další akce.



LOGICKÉ_OPERÁTORY

Logické operátory vrací, podobně jako srovnávací operátory, pouze hodnotu 1 nebo 0. Uvedeme si pouze dva základní: AND (logická konjunkce) a OR (logická disjunkce).



Výsledkem logické konjunkce je 1 pouze tehdy, když oba vstupní operandy jsou 1. Jinak je výsledek 0.

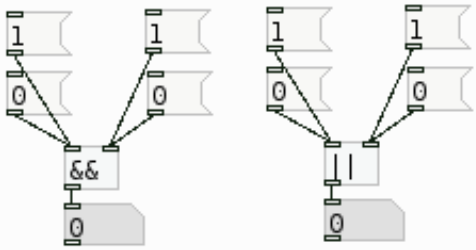
1 == pravda
0 == nepravda

Výsledkem logické disjunkce je 1 pouze tehdy, když je aspoň jeden z operandů 1. Jinak je výsledek 0.

Zjevnější nám výsledek těchto logických operací bude z následujících tabulek:

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

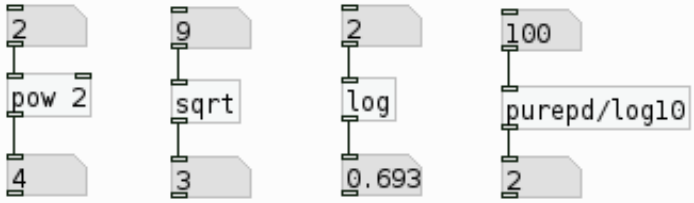


Mějme stále na paměti to, jak se chová aktivní a pasivní vstup! Výsledek je vždy vyhodnocen až po inicializaci levého, tj. aktivního vstupu.

Pokud do levého vstupu pošleme jakoukoliv hodnotu různou od nuly, pak ji logické operátory v Pd interpretují jako 1. Výsledek logické konjunkce bude tedy 1, i když do obou vstupů pošleme např. 10 a -7.

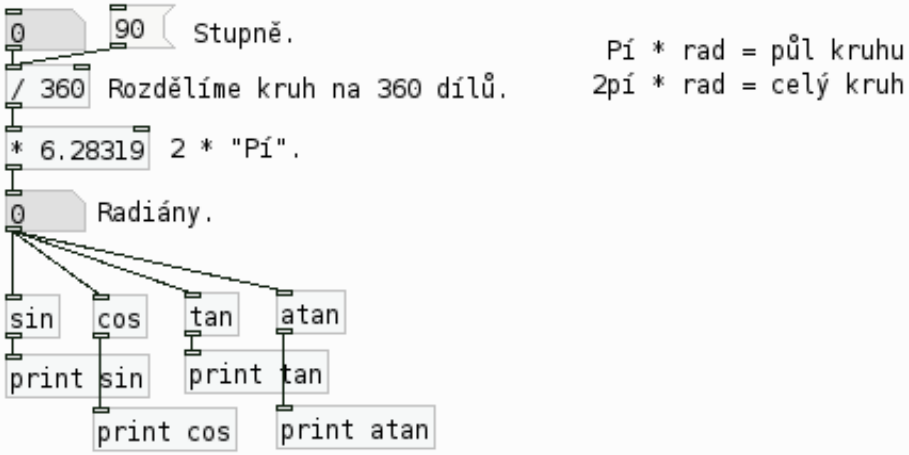
MOCNINA_ODMOCNINA_LOGARITMUS

Pokud budete potřebovat některé z těchto matematických operací, pak použijete objekty [pow] - mocnina, [sqrt] - druhá odmocnina a [log] nebo [purepd/log10], což je přirozený a dekadický logaritmus.



TRIGONOMETRIE

V Pd nechybí ani trigonometrické funkce. Na vstupu čekají na hodnotu v radiánech. [sin] a [cos] mají na výstupu hodnoty o rozsahu -1..1, [tan] 0..∞ a [atan] -pi/2..pi/2.



BITOVÉ_OPERÁTORY

Pozor, bitový operátor není binární operátor. Termín "bitový" zde znamená, že "v sobě" operace s čísly vyhodnocuje tak, jako kdyby byly zapsány ve dvojkové soustavě (číselná soustava, která používá pouze 0 a 1).

Přirozeně jsme zvyklí počítat v desítkové soustavě. Ta má základ 10.Číslo 256 v ní s pomocí tohoto základu dokážeme vyjádřit také takto:

$6 * 10^0 = 6 * 1 = 6$

$5 * 10^1 = 5 * 10 = 50$

$2 * 10^2 = 2 * 100 = 200$

Bereme tedy postupně jednotky, desítky, stovky atd. a násobíme je základem umocněným podle řádu, kde se číslo nachází. Výsledky pak sečteme a máme hodnotu 256.

Jak ale zjistit hodnotu čísla, které je zapsáno ve dvojkové soustavě? Jak převést binární číslo na dekadické? Podobně jako v předchozím případě, jen změníme základ z 10 na 2. Vezměme si např. binární číslo 1101.

$1 * 2^0 = 1 * 1 = 1$

$0 * 2^1 = 0 * 2 = 0$

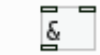
$1 * 2^2 = 1 * 4 = 4$

$1 * 2^3 = 1 * 8 = 8$

Součet jednotlivých výsledků nám dohromady dá 13.Hodnota binárního čísla 1101 je, vyjádřeno v dekadické soustavě, 13.

Dvojková soustava se základem 2 je v programování důležitá proto, že odpovídá reprezentaci čísel v počítači. Počítač "nevidí" číslo 13, ale 1101 a místo 9 "vidí" 1001.

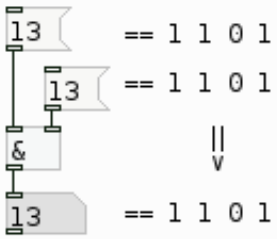
Uvedeme si pouze dva bitové operátory, které si snadno můžete splést s logickými operátory! Půjde o bitový AND a bitový OR. Oba dva se řídí podle stejných pravidel jako logické operátory (AND je 1, jen pokud jsou oba vstupy 1, OR je 1, pokud je aspoň jeden 1), jen "uvnitř" pracují s dvojkovým zápisem, takže výsledek bude jiný!

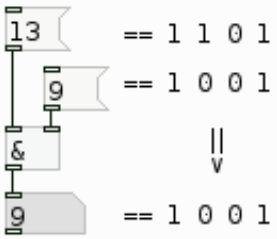


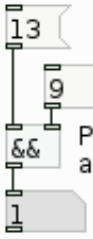
bitový AND

bitový OR

Bitový AND nám tedy dá tento výsledek:

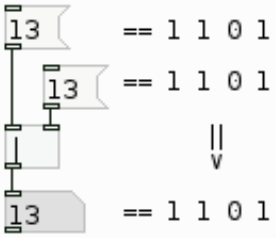


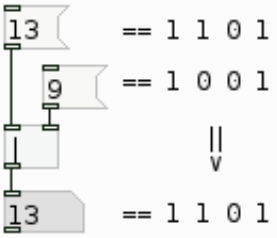




Pozor! Nejde o bitový, ale o logický operátor.

A bitový OR tento:





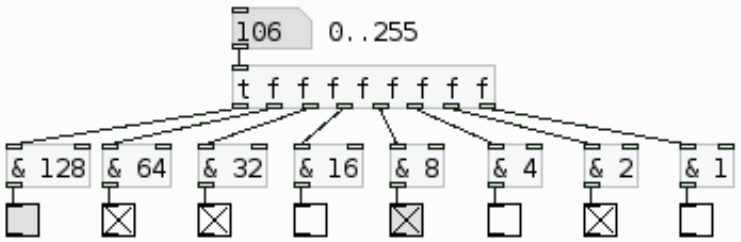
Čísla se v dvojkové soustavě zapisují obvykle v 8, 16, 32, 64, atd. -bitových číslech. Maximum, které na osmi bitech dokážeme vyjádřit je 1.11111e+07, což je dekadicky 255.

bit = nejmenší jednotka informace
1 / 0

$$1 * 2^0 + 1 * 2^1 + 1 * 2^2 + 1 * 2^3 + 1 * 2^4 + 1 * 2^5 + 1 * 2^6 + 1 * 2^7$$

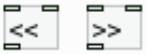
$$1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255$$

Když k bitovému operátoru [&] přidáme argument označující, jakou hodnotu má číslo dekadicky, když je jednička v daném řádu dvojkově zapsaného čísla, pak si můžeme postavit dekadicko-dvojkový převodník.



Nebo je to generátor všech možných pozic v jednostopém bicím automatu o osmi dobách?

Kromě bitového operátoru [&] a [|] jsou v Pd ještě operátory pro bitový posun: [<<] a [>>]. Vzhledem k tomu, že tato poslední podkapitola je ale spíš pro nadšence a zvědavce, necháme jejich průzkum na individuálním zvážení. Pokud by vás ale problematika zajímala hlouběji a tato kapitolka vám připadala málo podrobná a nesrozumitelná podívejte se na: V počítači jsou jen jedničky a nuly.



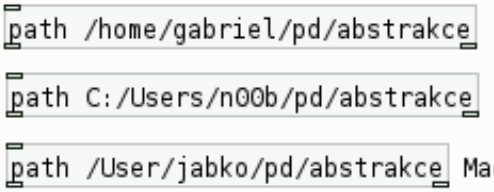
K základnímu programování v Pd bitové operátory nutně znát nepotřebujeme, přesto by ale měly patřit do výbavy každého skutečného programátora. Dají se s nimi totiž dělat moc pěkné triky: Jeden a druhý.

abstrakce

Až se v Pd pokusíte vytvořit několik menších projektů, jistě narazíte na to, že se vám některé části kódu budou opakovat. V takovém případě si můžete vytvořit sbírku subpatchů, které do svých nových projektů budete vkládat. Kdybychom si takto ale naprogramovali např. sampler, který by v sobě měl pole s názvem "sample_1" a pak bychom ho v patchi duplikovali, museli bychom pokaždé název pole změnit, aby pole nebyla navzájem v konfliktu. Neexistuje nějaká jednodušší cesta? Samozřejmě...

Samostatné části kódu, jež plní určitou funkci, nemusíme uzavírat jen do subpatchů, ale také do tzv. abstrakcí. Co to abstrakce je? Rozdíl mezi subpatchem a abstrakcí spočívá v tom, že subpatch je vždy součástí aktuálního otevřeného patche. Když provedeme změnu v jednom subpatchi, v dalších se neprojeví. Oproti tomu abstrakce je oddělený a samostatně uložený Pd patch, který do aktuálně otevřeného patche můžeme importovat. Pokud provedeme změnu v abstrakci a uložíme ji, projeví se ve všech ostatních abstrakcích.

Aby Pd "věděla", kde mají abstrakce hledat, musíme je mít uloženy ve stejném adresáři, v němž máme aktuálně otevřený patch, nebo na ně "ukážeme" pomocí objektu [path]. Ten má jeden parametr, a tím je cesta k adresáři s abstrakcemi nebo externími knihovnami. Zápis cesty se liší podle operačního systému, který máme na počítači nainstalovaný.



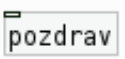
Pd mají v objektech problém s interpretací mezer, proto vždy ukládejte abstrakce do adresářů, jež v názvu nemají mezeru!

Existuje ještě třetí způsob, jak zadat cestu k abstrakcím: v menu Upravit vybereme Předvolby. Otevře se nové okno a v tom lze zadat cesty (tlačítko New), které Pd vždy prohledají v případě, že mají vytvořit abstrakci nebo objekt z externí knihovny.

PRVNÍ_ABSTRAKCE

Napište do nového patche (CTRL + n) vedlejší kód a uložte jej pod názvem "pozdrav.pd" do adresáře s abstrakcemi. Patch pozdrav.pd pak můžete s klidem zavřít. Dále nastavte v Pd jedním ze tří možných způsobů cestu k adresáři s abstrakcemi.

Pokud jste vše udělali správně, pak v jakémkoliv novém patchi, který otevřete a vytvoříte v něm objekt (CTRL + 1), do něhož napíšete "pozdrav", vytvoří abstrakci s tímto názvem. Měla by vypadat tak, jak vidíte vpravo:

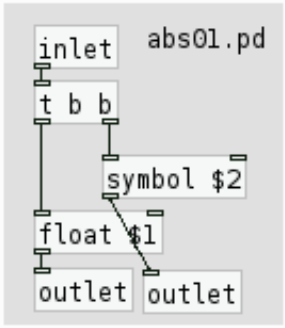


Na první pohled se vůbec nijak neliší od obyčejného Pd objektu. Kliknutím LTM jej ale můžeme otevřít a uvnitř uvidíme náš kód. V tomto se abstrakce podobá subpatchi, ale na rozdíl od něj nezačíná předponou "pd". Když do vstupu abstrakce zapojíme bang a klikneme na něj, pak abstrakce vypíše do konzole zprávu ahoj.

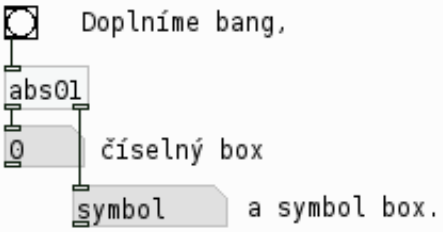
Výborně, máme za sebou programování naší první abstrakce.

DOLARY_V_ABSTRAKCI

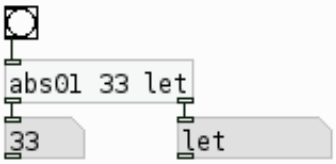
Až doposud jsme pracovali se znakem "$" převážně ve zprávách, kde "$1".. "$n" označují proměnné. Nyní se detailněji podíváme na to, jak s nimi můžeme pracovat, když je zapíšeme do objektu, a jakou funkci plní v abstrakcích. Přepište následující kód v rámci do samostatného patche a uložte ho pod názvem "abs01.pd" do adresáře s abstrakcemi.



A takto bude abstrakce vypadat,
---------->
když ji vytvoříme v hlavním patchi.

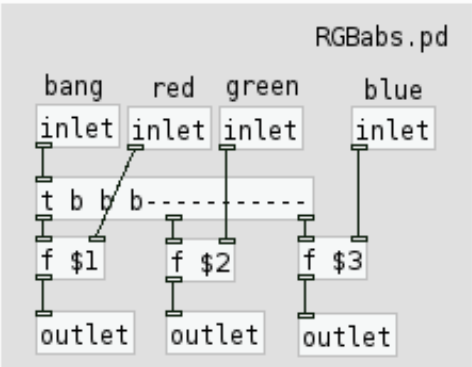


Když klikneme na bang vedoucí do abstrakce, nic se nestane, zkusme ale abstrakci doplnit o dva argumenty - první bude číslo a druhý bude symbol. Zapíšeme např: [abs01 33 let].



Když nyní klikneme na bang, v číselném a symbol boxu se objeví číslo 33 a symbol "let" - tedy agrumenty, které jsme předali abstrakci.

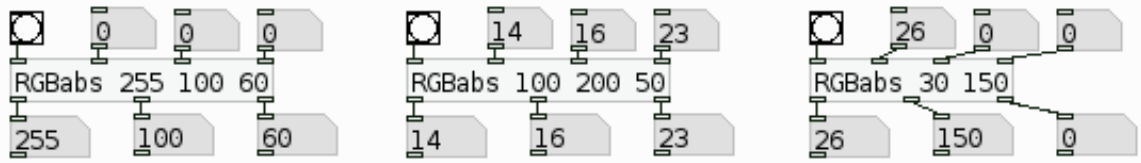
Pokud v abstrakci vytvoříme objekt typu float nebo symbol a doplníme je o $1, $2..$n, vytvoříme tak rozhraní abstrakce - pomyslný prostor pro předávání argumentů, se kterými pak abstrakce pracuje. "$1" odpovídá prvnímu argumentu abstrakce, "$2" druhému atd.



Podívejte se na vedlejší kód a zkuste popsat, jaké má abstrakce argumenty, vstupy, a jak bude asi fungovat.

Abstrakce má čtyři vstupy, ale jen tři argumenty typu float: [$f1], [$f2], [$f3]. Když do prvního vstupu pošleme bang (nebo i cokoliv jiného, protože [t b b b] konvertuje jakoukoliv zprávu na bang), z abstrakce "protlačíme" ven tři čísla, která jsme jí předali jako argumenty.

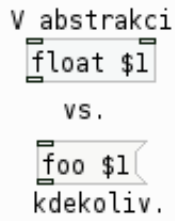
Druhý až čtvrtý vstup slouží pro vložení hodnoty z číselných boxů, které do vytvořené abstrakce zapojíme. Vidíme ale, že kabely z inletů vedou do pasivních vstupů objeků [$f1], [$f2] a [$f3]. Můžeme jimi posléze argumenty, které jsme abstrakci předali při vytváření, změnit. Na výstup se ale pošlou až poté, co do levého vstupu abstrakce pošleme bang.



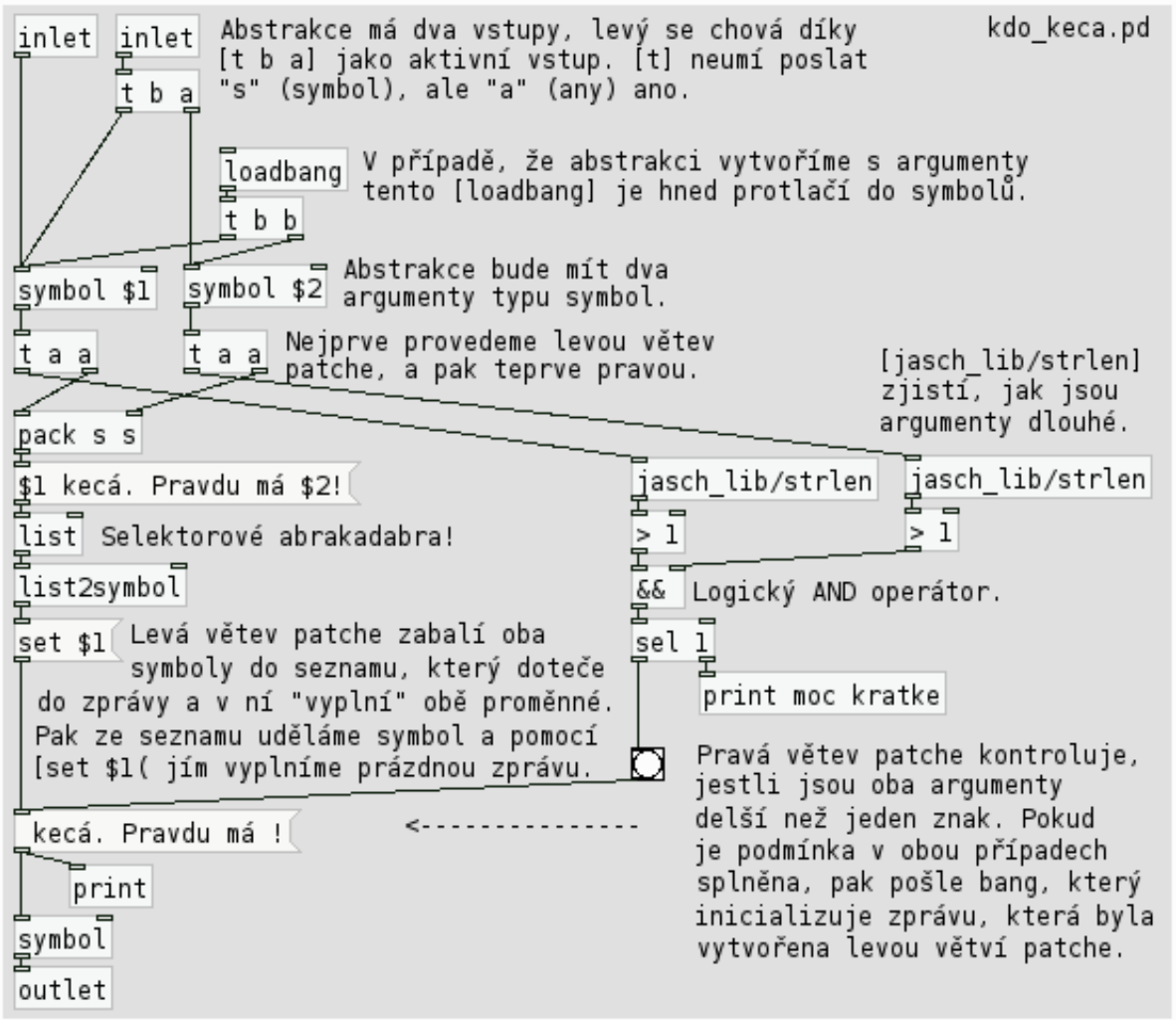
Na předchozí straně jsme vytvořili tři instance abstrakce [RGBabs], kterým jsme předali různé hodnoty, a každá z nich se chová individuálně a nezávisle. Ve třetím případě jsme [RGBabs] předali pouze dva argumenty, takže na třetí výstup pošle nulu. Stejně nezávisle by se chovala při předání jiných argumentů i abstrakce asb01.

Dodejme, že k dobré programátorské praxi patří to, že abstrakce uvnitř opatřujeme komentářem. Ten popisuje, jaké vstupy a argumenty abstrakce má, co dělá a jaké dává výstupy. Pojmenovat abstrakci při ukládání podle její funkce je též užitečné.

Ještě jednou si zopakujme a zapamatujme, že pokud znak "$1".."$n" zadáváme do objektu [float] nebo [symbol], který je součástí abstrakce, vytváříme tím argument, který abstrakce přijímá! Pokud znak "$1".."$n" uvidíme kdekoliv ve zprávě, pak jde o obyčejnou proměnnou - "mezeru", která ve zprávě bude nahrazena např. číslem, které do ní doteče z číselného boxu.

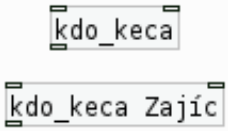


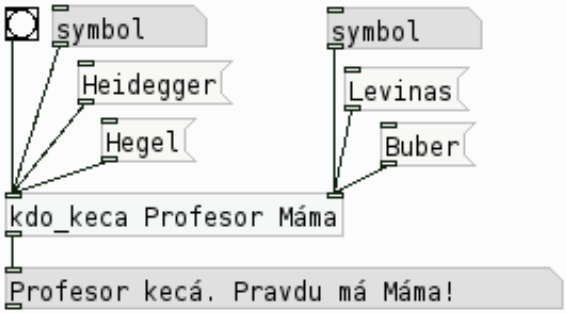
Na následujícím příkladu si ukážeme rozdílné zapsání znaků "$1" a "$2" do objektů a zprávy. Jde už o trošku "robustnější" abstrakci, která "ošetřuje vstupy": kontroluje u argumentů délku a pokud v obou případech není argument delší než jeden znak, upozorní nás na to v konzoli. Pokud jsou oba argumenty delší než jeden znak, vypíše zprávu do konzole i na výstup jako symbol.



Kód opět samostatně uložíme pod názvem "kdo_keca.pd" do adresáře s abstrakcemi, a pak zkusíme abstrakci vytvořit.

Když abstrakci při vytvoření nepředáme žádné, nebo krátké, vstupní argumenty, upozorní nás na to Pd v konzoli. Pokud jí zadáme dva argumenty delší než dva znaky, vypíše hlášku do konzole. Naši abstrakci navíc můžeme doplnit o vstupní zprávy/symbol boxy, bang a výstupní symbol box:





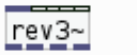
Bang funguje tak, že pokud byly vstupní podmínky pro délku argumentů splněny, zprávu znovu pošle do konzole i do symbol boxu.

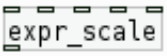
Abstrakci si rozklikněte LTM a podívejte se dovnitř, je v ní ještě trochu vylepšena pravá větev patche.

Uvedená abstrakce je sice prakticky nepoužitelná, ale jako případová studie na jednoduché ošetřování vstupů a vyhodnocování podmínek snad postačí. Použili jsme v ní nový objekt [jasch_lib/strlen], který umí zjistit délku symbolu, a ukázali jsme si v ní dvojí možný zápis znaku "$1" a "$2". V jednom případě to bylo při specifikování argumentů v abstrakci a ve druhém to byl zápis proměnných do zprávy. Pokud vám v patchi nebyla srozumitelná levá větev s popiskem "selektorové abrakadabra", doporučuji vrátit se ke studiu této problematiky.

Pokud si v programátorské praxi zvyknete vytvářet a používat abstrakce, hodně vám to ulehčí práci. Určitě je jednodušší si s postupem času založit sbírku jasně pojmenovaných abstrakcí a ty vkládat do projektu, na kterém pracujete, než dohledávat kousky kódů v patchích a subpatchích. Již teď bychom se mohli vrátit např. k počítadlům, BPM tempu a sekvenceru a vytvořit si z nich abstrakce.

Bez abstrakcí se neobejdeme v řešení některých situací. Nebo bychom místo nich museli použít mnohem méně elegantní a "tvrdé" řešení - vzpomeneme si na to při vytváření sampleru. S abstrakcemi jsme ostatně již také pracovali a Pd-extended jich obsahuje celou řadu. Vzpomeňte si např. na objekt [expr_scale] - to je také abstrakce.

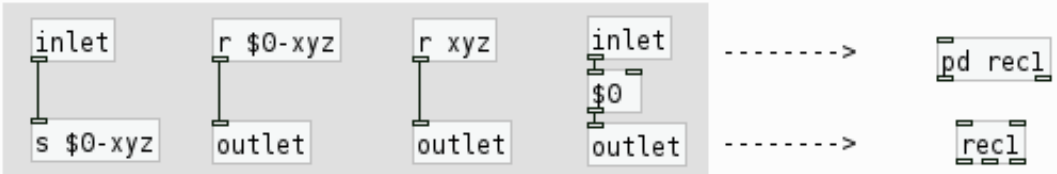




Podívejte se dovnitř kliknutím LTM.

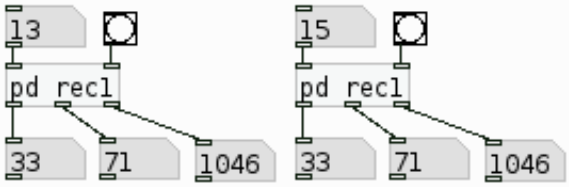
ABSTRAKCE_A_DOLAR_NULA

V kapitole o "bezdrátovém" posílání zpráv jsme se zmínili o indentifikátoru "$0-". Když bychom se ještě vrátili k analogii o mluvení v celém domě a jednom bytě, pak jsou abastrakce v kombinaci s identifikátorem "$0-" způsobem, jak "mluvit v jednom bytě", neboli prostředkem k zaručení lokální platnosti jmen proměnných. Vysvětlíme si toto enigmatické tvrzení na konkrétním případě.

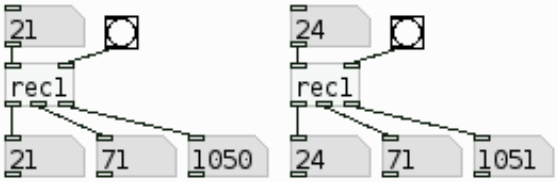


Kód z rámce na předchozí stránce jednou vložíme do subpatche [pd rec1] a jednou jej uložíme jako samostatnou abstrakci s názvem "rec1.pd". Pak subpatch jednou zduplikujeme a vytvoříme dvě instance abstrakce [rec1]. Subpatche i abstrakce doplníme o číselné boxy a jeden bang, takže výsledek bude vypadat takto:

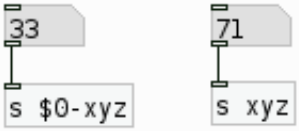
Subpatche:



Abstrakce:

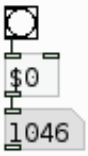


Jak v subpatchích, tak v abstrakcích máme stejný kód: první vstup posílá bezdrátově zprávu "$0-xyz". Na první výstup putuje bezdrátově přijatá zpráva "$0-xyz" a na druhý zpráva "xyz". Do třetího výstupu pak hodnota z identifikátoru "$0", kterou ven dostaneme bangem. Vložme do hlavního - rodičovského patche ještě následující "bezdrátové vysílače".



Pokuste se nyní sami provést "reverzní inženýrský" experiment a pochopit, jak se subpatche a abstrakce chovají. V čem je rozdíl?

Vidíme, že když posíláme zprávu do subpatche - ať už přímo do něj z číselného boxu, nebo bezdrátově z rodičovského patche, na levý výstup u obou vždy doteče daná hodnota. U abstrakcí se na výstupu neděje nic. Když inicializujeme bangem subpatche i kód s identifikátorem "$0" napravo, zjistíme, že hodnota, kterou identifikátor má, je stejná. Subpatche jsou součástí hlavního rodičovského patche a identifikátor je pro ně stejný.



Oproti tomu abstrakce je samostatný soubor a každá její instance má jedinečný identifikátor. Proto "přijímač" [$0-xyz] v abstrakcích "neslyší" to, co posíláme ze subpatchů nebo z rodičovského patche. Díky identifikátoru "$0" v něm názvy proměnných mají pouze lokální platnost.

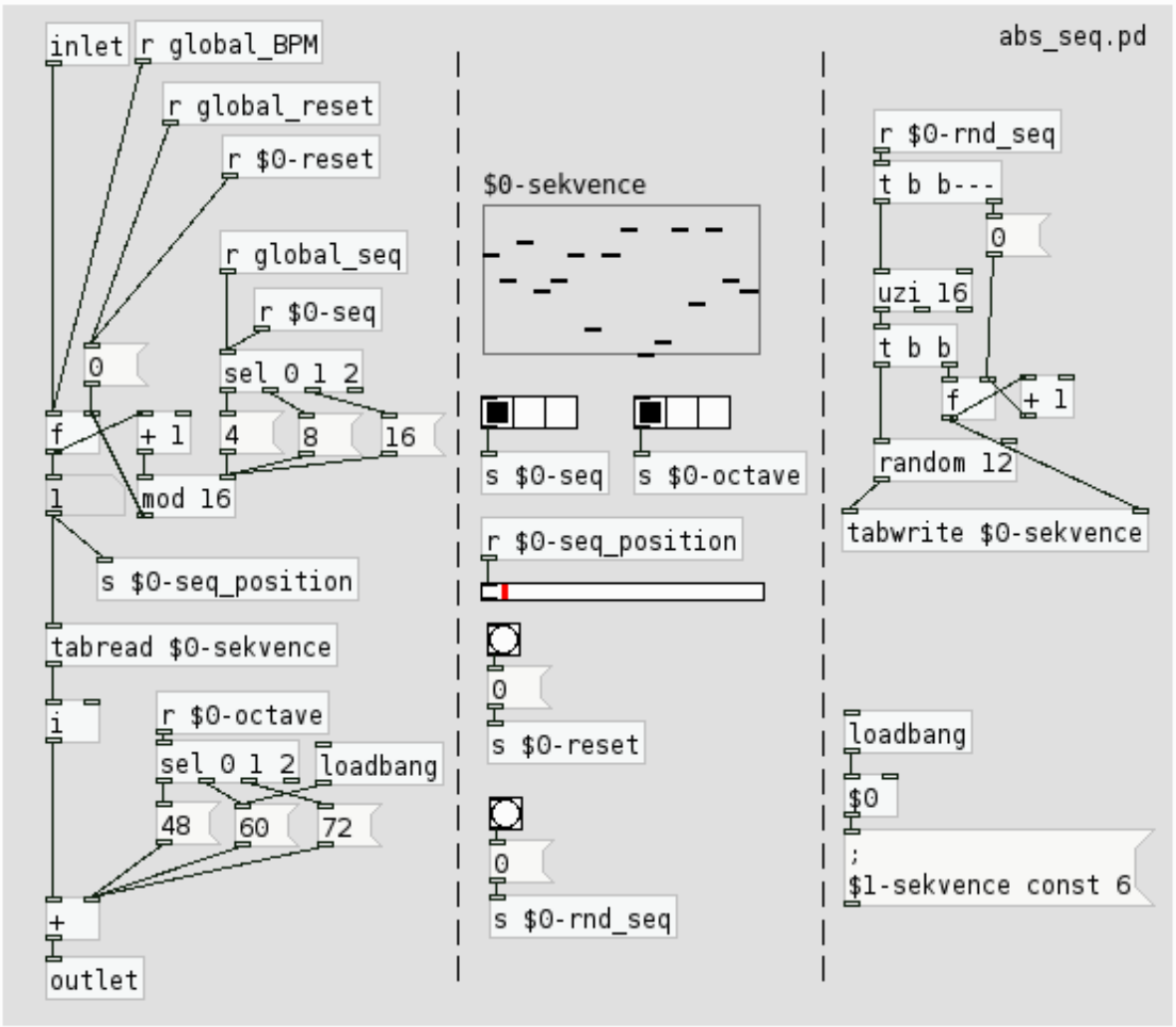
Zpráva "xyz" bez identifikátoru má globální platnost a bude doručena jak do subpatchů, tak do abstrakcí.

Na následující případové studii si ukážeme, jak skloubit to, co víme o identifikátoru "$0-", abstrakcích a polích. Takže se pro nás látka o abstrakcích snad stane méně abstraktní.

DOLAR_NULA_ABSTRAKCE_A_POLE

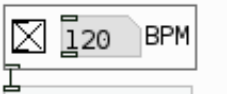
Postavíme si sekvencer, který ale nebude umístěný v subpatchi, jak tomu bylo, když jsme ho stavěli poprvé. Bude to samostatná abstrakce a místo přepínačů v něm k zadávání hodnot budeme využívat pole.

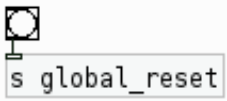
Nuže, kód abstrakce bude vypadat takto:

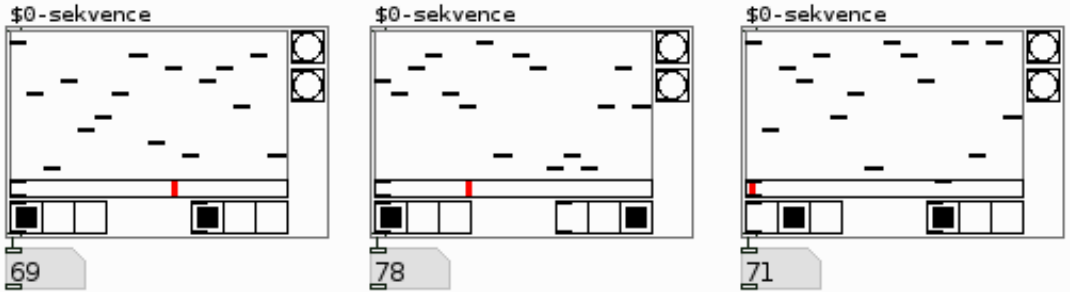


Uložíme ho jako "abs_seq.pd" do adresáře s abstrakcemi. Když pak vytvoříme v rodičovském patchi instanci, bude vypadat jako objekt [abs_seq] s jedním vstupem a jedním výstupem. To ale není všechno. Stejně jako v případě subpatchů, tak i abstrakce podporují tvorbu grafického uživatelského rozhraní. Vraťme se tedy k našim znalostem ohledně vytváření GOP:

Prostým kliknutím LTM otevřeme abstrakci a v jejím okně klikneme PTM do bílé plochy a z menu vybereme Vlastnosti, v novém okně nastavíme patřičné dimenze, které GOP má mít - v tomto případě je to 151x99, ale vy si uživatelské rozhraní samozřejmě můžete nadefinovat dle libosti. Nezapomeneme vyškrtnout položky "Graph-on-parent" a "Skrýt název objektu". Do červeně vyznačeného rámce přetáhneme pole a další grafické prvky. Abstrakci nezapomeneme znovu uložit! Zavřeme ji et voilà...







Nyní si naši abstrakci detailněji "přečteme". V levém bloku abstrakce vidíme kód z předchozí verze sekvenceru. Jde o počítadlo, jež je rozšířeno o řadu objeků [r], kterými do něj posíláme zprávy jak lokálního, tak globálního charakteru. Sekvencer bude číst data z globálního generátoru BMP, ale zároveň má jeden [inlet], kterým ho budeme moci řídit individuálně. Stejně tak ho budeme moci lokálně nebo globálně vynulovat a nastavovat délku sekvence. Rozdíl od první verze spočívá v tom, že sekvenci počítáme od nuly - odebíráme ji z objektu [f] a ne z [+ 1]. To kvůli indexování pole, které začíná právě indexem 0.

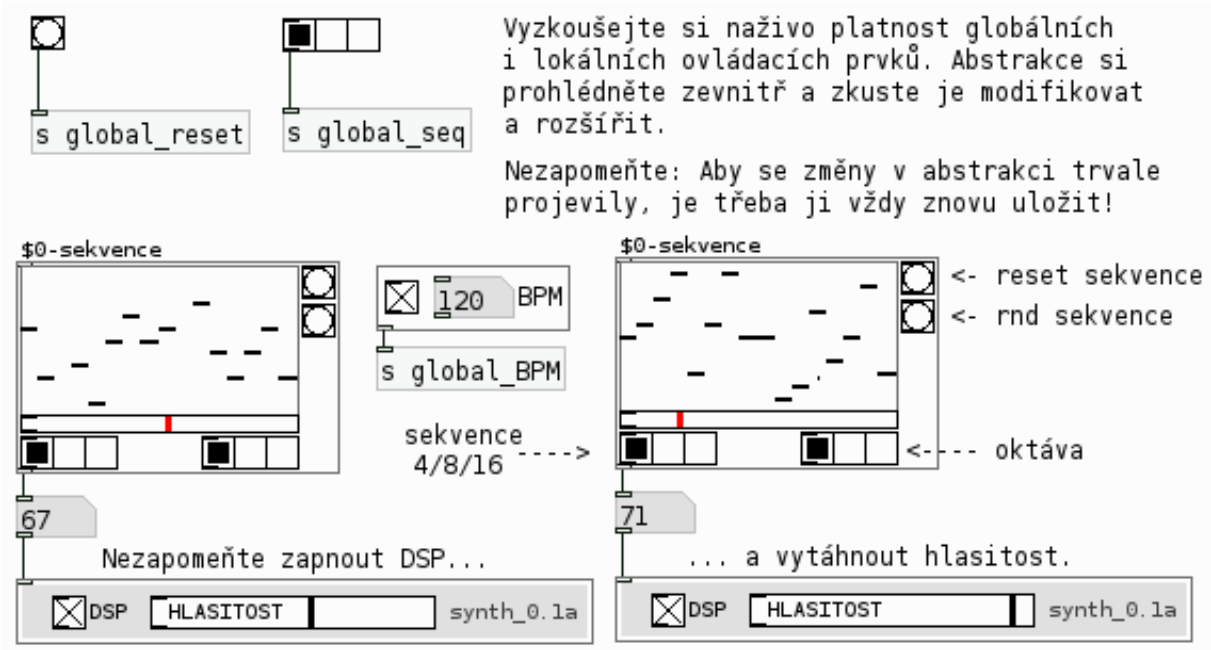
Lokálně číslo sekvence posíláme [s $0-seq_position] do posuvníku. Tomu jsme nastavili ve vlastnostech stejnou délku v pixelech, jakou má pole "$0-sekvence", a hodnoty vlevo: 0 a vpravo: 15.Posuvník tedy využijeme jako grafický prvek informující nás o tom, kde se právě v sekvenci nalézáme. Bylo by možné v jeho vlastnostech přímo zadat do kolonky "Přijmout symbol" $0-seq_position, ale v našem případě to kvůli názornosti a čitelnosti děláme s pomocí objektu [r $0-seq_position]. To je ostatně případ i dalších grafických prvků.

[tabread $0-sekvence] čte hodnoty z indexů pole s názvem "$0-sekvence". Díky předponě "$0-" je pole vytvořeno s jedinečným identifikátorem, a proto nebude v konfliktu s jinými poli, když vytvoříme více instancí naší abstrakce. Hodnotu z pole, která je typu float, zaokrouhlujeme objektem [i], abychom měli na výstupu pouze celá čísla, a ještě k ní přičítáme hodnoty, abychom se dostali do rozumné škály not v MIDI. [loadbang] vždy po vytvoření abstrakce nastaví tuto hodnotu na 60

V prostředním bloku vidíme prvky tvořící grafické uživatelské rozhraní. Pole se jmenuje "$0-sekvence". Má 16 prvků (16 dob), grafické rozměry 130x70 pixelů. Rozsah na ose X je 0..16 (doby), na ose Y pak 12..0 (noty). Bangy a přepínače posílají pouze lokálně platné zprávy, takže budou mít dopad pouze v hranicích jedné vytvořené abstrakce. "Mluvíme" jimi v jednom bytě, ne v celém domě.

V pravém bloku jsou pak již jen dva malé "kousky kódu". Jeden generuje 16 náhodných hodnot v rozsahu 0..11 a zapisuje je do pole "$0-sekvence". A konečně poslední [loadbang] vždy po vytvoření abstrakce nastaví všechny prvky v poli na hodnotu 6.Mohli jste se nad jeho podivností pozastavit: proč jsme nezapsali "$0-" rovnou do zprávy? Znak "$0" zapsaný ve zprávě je selektor zprávy (jde tedy buď o symbol, float, list, nebo any). "$0" zapsaný v objektu je identifikátor patche. Stačí, když si zapamatujeme, že "$0" v objektu je specifické číslo, jedinečné pro každý patch. Na "$0" ve zprávě můžeme zapomenout.

K tomu, abychom do zprávy nastavující pole "$0-sekvence" dostali toto specifické číslo, jsme použili malého triku: z objektu [$0] dostaneme námi požadovanou hodnotu identifikátoru a tu pak předáváme zprávě, která ji čte jako obyčejnou proměnnou - ve zprávě je tedy znak "$1".



Asi vám bude chvíli trvat, než používání abstrakcí dostanete "pod kůži", ale ze své vlastní praxe vám jejich používání můžu jen doporučit. Vaše projekty díky nim budou srozumitelnější a strukturovanější. Určitě je jednodušší používat sbírku abstrakcí, které si s postupem času vytvoříte, než "lepit" patch vždy od začátku.

Možná si také říkáte, že po takové spoustě probrané látky stále neumíte vytvořit nic, co by odpovídalo vašim představám. Jednak jde o potíž související s překonáním nesnadné fáze v učení se nového jazyka a jednak se již pomalu blížíme k "praktičtějším" kapitolám, kde se budeme konkrétně zabývat zvukem a obrazem.

Uvedená případová studie opravdu nereprezentuje nějaký velkolepý projekt, ale posloužila nám jako dobrý příklad k vysvětlení lokální a globální platnosti proměnných v rodičovském patchi a abstrakcích.

Mohli bychom nyní pokračovat opět problematikou implementace presetů, které by uměly spolupracovat s abstrakcemi. Šlo by ale už o poměrně robustní a detailní projekt, přesahující hranice našeho zájmu. Nicméně pro zvídavé a odolné bych ke samostudiu doporučil sbírku abstrakcí s názvem DIY2 vytvořených id hardoff. Najdete je na Pd Fóru, kde jsou po registraci volně ke stažení.

Balík abstrakcí k samostudiu je po registraci dostupný zde.

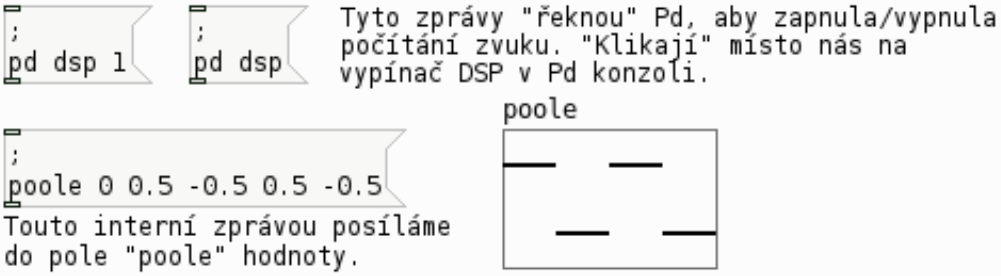
Pokud se na hardoffovy abstrakce chcete podívat a zjistit, jak problematiku ukládání stavu abstrakcí vyřešil, otevřete si v Pd soubor s názvem "readme-state-saving.pd". Jde v podstatě o modifikovanou verzi systému Sssad, o němž jsme se již zmiňovali. Pokud jste programátorsky trošku zběhlí, jistě vám nebude dělat problémy z jeho sbírky tento kód vypreparovat a ev. ho implementovat do vašeho projektu. Nebo se jím můžete inspirovat a vytvořit ještě lepší a jednodušší verzi.

K průzkumu lze doporučit i další hardoffovy patche, prohlídku lze začít otevřením souboru "000-introduction.pd".

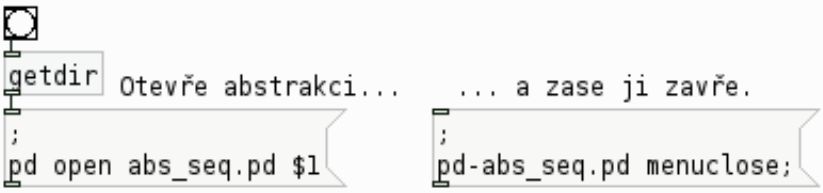
interní_zprávy

Naše seznámení s Pd by nebylo "kompletní", kdybychom neprobrali tuto, poněkud obskurní, kapitolu. Interní zprávy (internal messages) jsou totiž zvláštním typem zpráv, kterými můžeme "mluvit" přímo k Pd a "sahat" do kódu z vyšší úrovně. S jejich pomocí můžeme dynamicky měnit vlastnosti námi napsaného patche, nebo vytvářet kódy, které vytvářejí další kódy. V té souvislosti se o této programátorské praktice hovoří jako o metaprogramování.

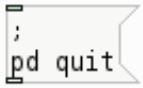
Na některé interní zprávy jsme již narazili, aniž bychom se jimi výslovně zabývali. Jistě si vzpomenete na následující:



Jsou zde ale i další, kterými můžeme např. otevřít Pd patch. Zpráva začíná středníkem, následovaným znaky "pd" - to znamená, že "říkáme" přímo něco Pd. "open" je příkaz, který po Pd chceme - tedy otevřít soubor. Dva argumenty označují jméno souboru a přesnou cestu k němu. V našem případě druhý argument "vyplňujeme" zprávou z objektu [getdir], jenž vrací cestu k aktuálnímu adresáři.

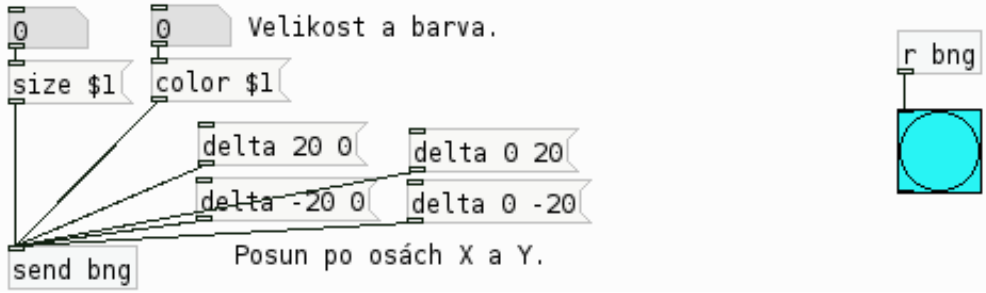


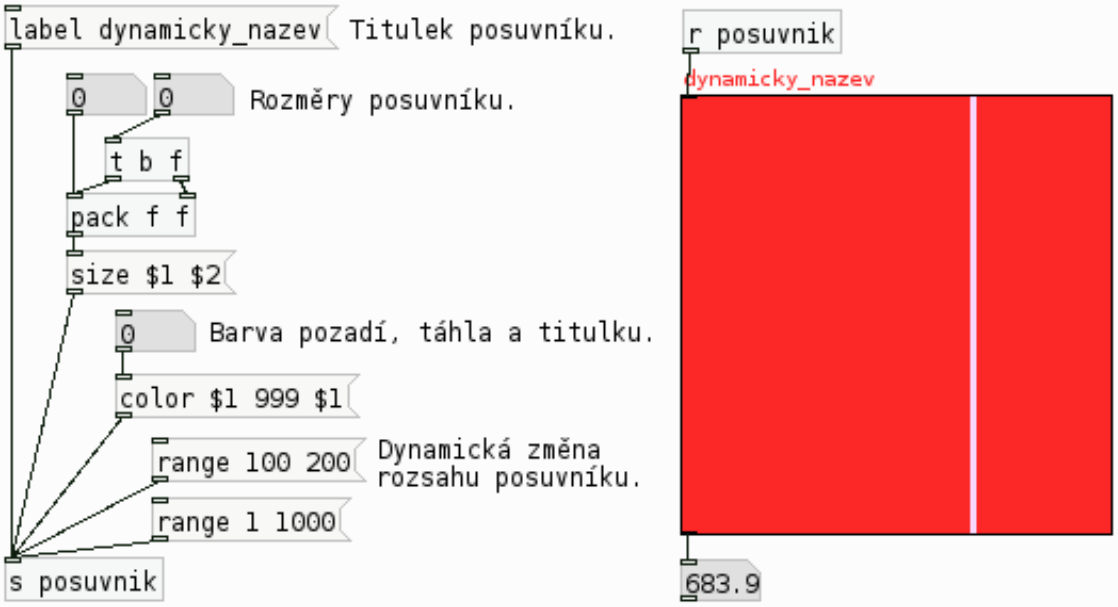
Ve druhé zprávě vidíte, že za "pd" následuje pomlčka a jméno souboru. To znamená, že zprávu posíláme do daného patche. V tomto případě posíláme patchi se jménem abs_seq.pd příkaz "menuclose".



GUI_DYNAMICKY

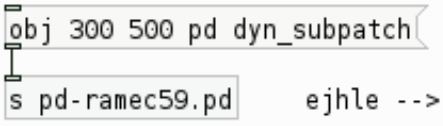
Do grafických uživatelských prvků jako bang, posuvník atd. lze posílat předdefinované interní zprávy a měnit tak jejich parametry.



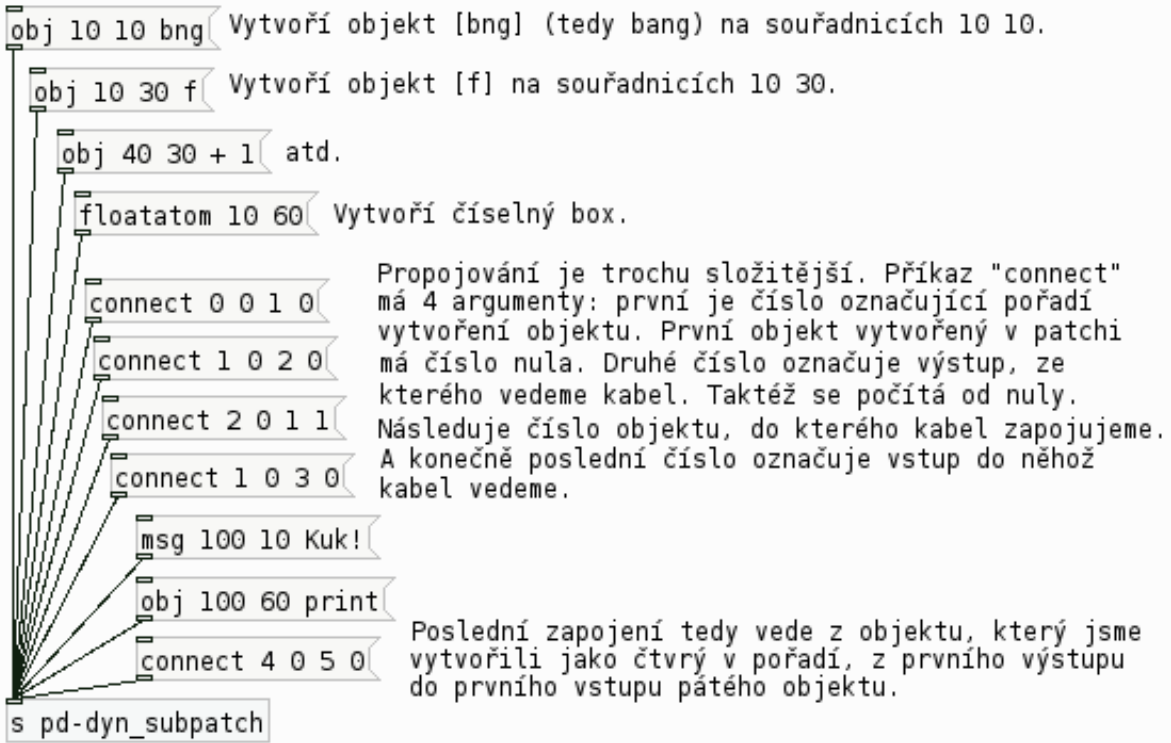


VYTVÁŘENÍ_KÓDU_KÓDEM

Tu pravou radost z metaprogramování si vychutnáme, když začneme vytvářet kód kódem. Nejprve si otevřeme nový subpatch tím, že do hlavního - rodičovského patche pošleme zprávu ve tvaru [obj X Y pd nazev_subpatche(.

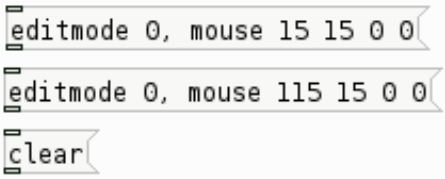


V hlavním patchi s názvem "ramec59.pd" jsme na souřadnicích 300 a 500 vytvořili subpatch s názvem "dyn_subpatch". A nyní do tohoto subpatche můžeme posílat další zprávy a vytvořit v něm funkční kód. Zprávy naklikávejte postupně odshora dolů a pozorujte, co se v subpatchi děje!



Při testování předchozí ukázky je důležité dodržet pořadí, ve kterém objekty v subpatchi vytváříme, a "naklikávat" zprávy postupně shora dolů, jinak propojení nevytvoří funkční patch. Též je třeba každou akci provést pouze jednou. Pokud bychom např. subatch nebo některá spojení vytvořili víckrát, může to vést ke zhroucení Pd.

Poté, co jste v subpatchi vytvořili interními zprávami funkční kód, zkuste do objektu [s pd-dyn_subpatch] napojit ještě následující zprávy a vyzkoušejte co dělají.

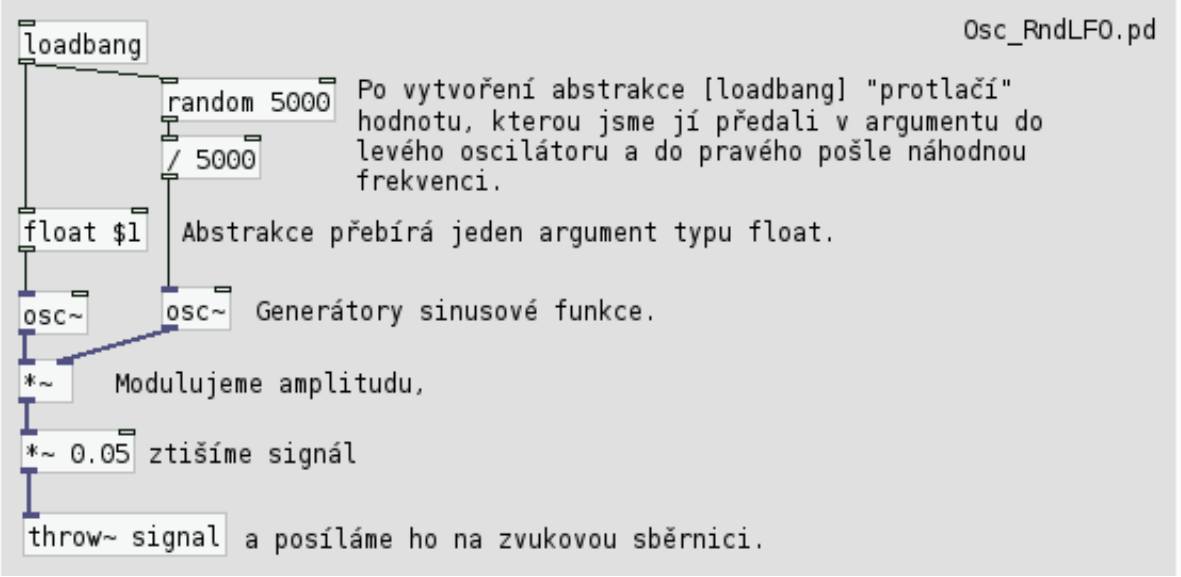


První dvě zprávy klikají do patche na danou souřadnici, takže pohání počítadlo, nebo se vytiskne zpráva do konzole. Zpráva [clear(subpatch vymaže.

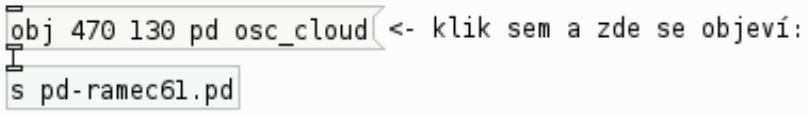
Přehled interních zpráv najdete přes menu Nápověda -> Pd Help Browser -> Manuals -> pd-msg. Dokumentace je ale v některých případech ne zrovna aktuální. Například od interní zprávy [click(, uvedené v této dokumentaci, bychom se kliknutí do subpatche nedočkali.

Praktické využití metaprogramování spočívá např. v automatizaci při vytváření rozsáhlých kódů, jejichž zápis by nám jinak trval velmi dlouho. Na následující případové studii si ukážeme, jak napsat kód, který vytvoří stovku instancí jedné zvukové abstrakce.

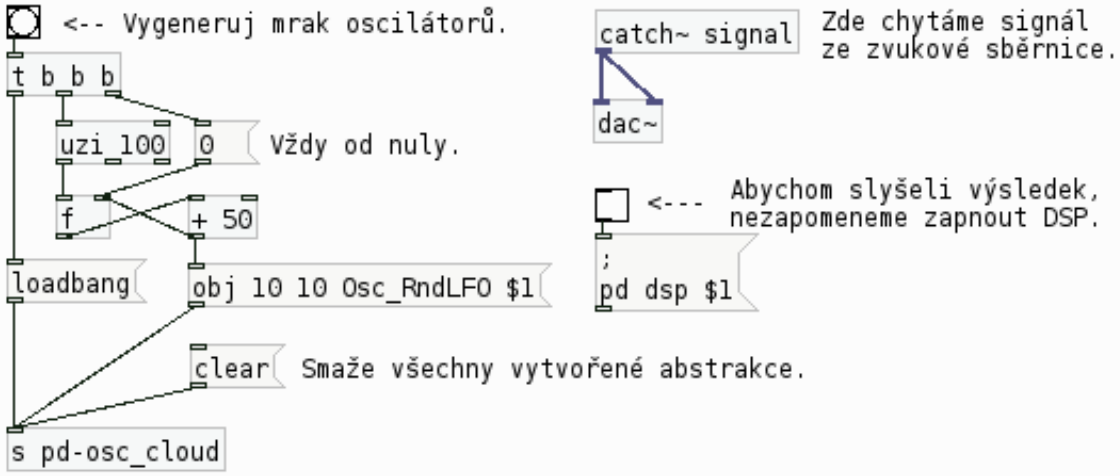
Z důvodu názornosti studie trochu předběhneme k problematice zvuku. Níže uvedená abstrakce obsahuje dva oscilátory, z nichž levý bude přebírat argument typu float při jejím vytvoření. Tento argument je frekvence, kterou bude oscilátor hrát. Funkce pravého oscilátoru spočívá v modulaci amplitudy levého oscilátoru - jinými slovy: bude modulovat jeho hlasitost, a to náhodně generovanými nízkými frekvencemi o rozsahu 0..5 Hz. Výsledný signál ještě musíme ztišit vynásobením hodnotou 0.05, protože abstrakci budeme mnohokrát duplikovat. Kdybychom signál neztišili, byl by výsledný zvuk značně zkreslený. Výstup posíláme bezdrátově na zvukovou sběrnici jménem signal. Patch opět samostatně uložíme pod názvem "Osc_RndLFO.pd" do adresáře s abstrakcemi.



V rodičovském patchi, jenž se v tomto případě jmenuje ramec61.pd, vytvoříme na souřadnici 470 130 interní zprávou subpatch [pd osc_cloud]:



A nyní do subpatche pošleme 100x zprávu, která vygeneruje 100 instancí předchozí zvukové abstrakce. Na výstupu počítadla z objektu [+ 50] dostaneme řadu čísel v rozsahu 50..5000 - ty posíláme jako proměnnou zprávě, jež v subpatchi [pd osc_cloud] vytváří jednotlivé instance abstrakce [Osc_RndLFO]. Proměnná je tedy zároveň argumentem abstrakce.



Když se podíváte dovnitř subpatche [pd osc_cloud], uvidíte jen jeden objekt s názvem [Osc_RndLFO 5000] - to je poslední vytvořená abstrakce a všech 99 ostatních se schovává pod ní. Příčinou překrývání jsou "na pevno" definované souřadnice 10 10 ve zprávě.

K tomu, aby se všechny abstrakce v subpatchi rozezněly, je třeba do něj, po jejich vygenerování, poslat ještě jeden loadbang - nestačí jen ten, který je v abstrakci. Řešení problému s inicializací abstrakcí vytvářených interními zprávami je k dohledání na Pd Mailing Listu, který je také velmi dobrým zdrojem informací. Link s touto problematikou najdete zde.

Po kliknutí na bang a zapnutí DSP by měla být slyšitelná ostrá a spektrálně bohatá plocha ze sta oscilátorů. Daný postup nemusí být aplikován jen k vytváření masivních zvukových ploch, ale lze ho využít i při generování rozlehlých vizuálních struktur - podívejte se na některé z příkladů ke knihovně pmpd (Nápověda -> Pd Help Browser -> pmpd -> examples -> 19_vertex.pd). Stejně dobře ho ale využijete při vytváření větších grafických uživatelských rozhraní.

CVIČENÍ:

- pokuste se patch vylepšit tak, aby abstrakce v subpatchi rozmisťoval do přehledné mřížky
- experimentujte s počtem vytvářných abstrakcí i s hodnotami, které jí předáváte - lze tak dosáhnout široké škály různých zvukových ploch

síťování

Poslední kapitola obecné části rukověti, kterou máme právě před sebou, se bude týkat komunikace patchů po síti. Problematika sítí dalece přesahuje možnosti našeho pojednání, proto si ukážeme pouze základní komunikaci typu server-klient. Pokud máte k dispozici dva počítače propojené v lokální počítačové síti (LAN), můžete si zde probranou látku naživo vyzkoušet. Jinak si budeme "po síti" posílat zprávy pouze v rámci místního hostitele (localhost).

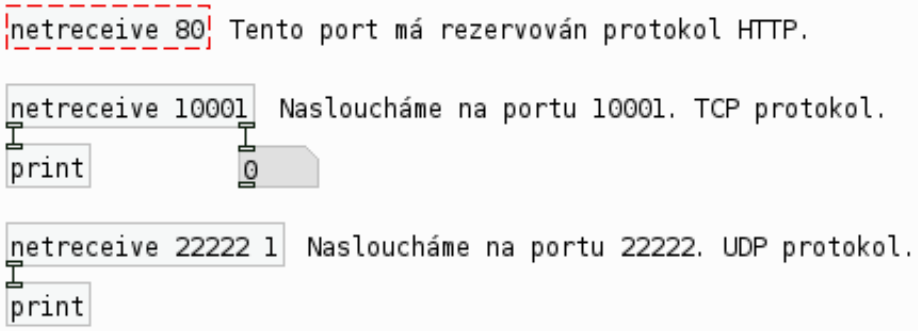
Pokud je pro vás problematika počítačových sítí naprosto neznámá, doporučuji vám aspoň zběžně z Wikipedie nastudovat hesla TCP/IP, ethernet, IP adresa, síťový protokol a síťový port.

NASLOUCHÁME_NA_PORTU

Dnes je v podstatě každý počítač vybavený síťovou kartou, takže může komunikovat v rámci protokolů TCP a UDP. Komunikace probíhá vždy přes nějaký port, což je číslo v rozsahu 0..65535. Například HTTP (Hyper Text Transfer Protocol) standardně využívá port 80 a FTP (File Transfer Protocol) zase port 21. Porty jsou pomyslné dveře do počítače, kterými data můžou přicházet i odcházet.

Porty v rozsahu 0..1024 jsou obvykle vyhrazeny systémovým službám. Různé síťové aplikace využívají různé síťové porty a aby nám provoz po síti fungoval i v Pd, je třeba se vždy ujistit, že je daný port volný. Obecně se dá říct, že od portu 10000 nahoru je "volno".

K naslouchání na portu je v Pd určen objekt [netreceive]. Má dva argumenty: prvním specifikujeme číslo portu a druhým protokol (0 = TCP, 1 = UDP). Pokud náhodou zvolíme port, jenž je již systémem nebo nějakou aplikací rezervován, jeho vytvoření se nepodaří.



Základní rozdíl mezi TCP a UDP spočívá v tom, že TCP ověřuje, zda byla data opravdu doručena příjemci. UDP toto neověřuje - datagramy se po cestě můžou ztratit.

Na první výstup [netreceive] posílá doručená síťová data. V případě užití protokolu TCP nás pravý výstup informuje o počtu připojených klientů na daný port. Pokud jsme tedy na jednom počítači otevřeli port, můžeme na něj z jiného počítače něco poslat.

Nikdy neotvírejte porty na serveru, který má veřejně dostupnou IP adresu, bez toho, aniž byste řádně zajistili jeho zabezpečení! Provoz na lokální síti nepřipojené do Internetu je samozřejmě bezpečnější.

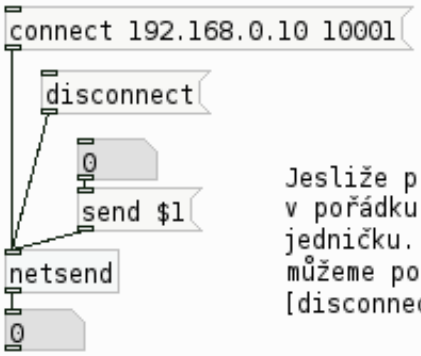
POSÍLÁME_DATA_NA_PORT

Abychom mohli zprávu na počítač, který v síti naslouchá, něco poslat, musíme znát jeho IP adresu a port. Ke zjištění IP adresy vám poslouží program ifconfig (ve Windows ipconfig), který se spouští z terminálu (pod Windows klikněte na tlačítko Start -> Spustit -> cmd). Jeho výstupem je řada informací o dostupných síťových rozhraních, mezi kterými najdete také číslo označující IP adresu daného počítače. IP adresa sestává ze čtyř čísel v rozsahu 0..255 oddělených tečkou. If/pconfig samozřejmě spouštíme na počítači, který naslouchá.

IP adresu si poznamenáme (v lokálních sítích má obvykle tvar 192.168.0.xxx), port již známe - v našem případě jde o číslo 10001. Nyní se můžeme přemístit k počítači, ze kterého budeme data posílat.

K prověření dostupnosti druhého počítače vám poslouží příkaz ping, který se taktéž spouští z terminálu. Když příkazu ping předáme v argumentu IP adresu (např. ping 192.168.0.10) a druhý počítač s touto IP je v síti dostupný, měli bychom v terminálu vidět hlášku jako "64 bytes from 192.168.0.10: icmp:req..." - ta nás informuje o rychlosti odezvy druhého počítače. Pokud ping dlouho žádnou hlášku nevypíše nebo zahlásí "connect: Network is unreachable", pak na sebe naše počítače v síti nevidí a je třeba jejich IP adresy správně nastavit. Problematikou konfigurace lokální sítě se zde ale nebudeme zabývat. Eventuelně prosím tuto látku dostudujte v patřičné odborné literatuře.

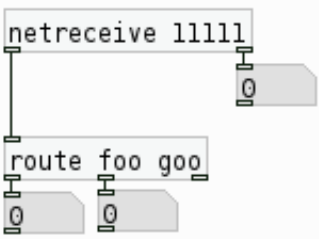
Pakliže na sebe počítače v síti "vidí", můžeme mezi nimi uskutečnit síťovou komunikaci. K zasílání zpráv po síti v Pd slouží objekt [netsend].



Do [netsend]u posíláme zprávu začínající příkazem "connect", následovanou IP adresou a portem počítače, ke kterému se připojujeme.

Jesliže připojení k druhému počítači proběhlo v pořádku, pak na výstupu [netsend]u uvidíme jedničku. Pak zprávou [send $1(na druhý počítač můžeme poslat data (číslo, nebo symbol). Zpráva [disconnect(spojení ukončí.

Pokud nemáte k dispozici síť, na které byste funkčnost objektů [netreceive] a [netsend] otestovali, vyzkoušejte aspoň následující lokální provoz.



Vidíme, že do [netsend]u lze posílat pojmenované proměnné a ty na druhé straně z objektu [netreceive] rozbalovat pomocí [route].

NETCAT_A_JINÉ_ALTERNATIVY

K posílání dat do Pd po síti lze použít i jiné programy. Ukážeme si, jak to provést s Netcatem (nc). Ten umí, podobně jako [netreceive] a [netsend], otevřít port, nebo na něj poslat data. Nejprve ho do systému musíme nainstalovat: v Linuxu je standardně součástí repozitářů, existuje ale ve verzi i pro Windows a MacOS.

Link na Netcat

Když po jeho nainstalování do terminálu zapíšeme příkaz: echo "100;" | nc 192.168.0.10 10001" a pokud druhý počítač má tuto IP adresu a naslouchá na portu 10001, bude mu doručena hodnota 100. Všimněte si, že hodnota je ukončena středníkem. Toto je třeba dodržet. Pd k zasílání a přijímání zpráv používají totiž tzv. FUDI protokol, ve kterém je definováno, že každá zpráva musí být oddělena středníkem.

FUDI protokol

Netcat je šikovný nástroj, s jehož pomocí můžeme přesměrovat síťový provoz do Pd. Stejně jednoduše to ale lze provést v jakémkoliv programovacím jazyku (PHP, Python, Java, atd.) - takže např. s pomocí kombinace HTML a PHP by bylo možné postavit systém, jehož uživatelské rozhraní je viditelné ve webovém prohlížeči. Jeden z projektů tohoto typu jménem WebPd, využívající jazyk Java, je právě ve vývoji.

WebPd na GitHubu

OPEN_SOUND_CONTROL

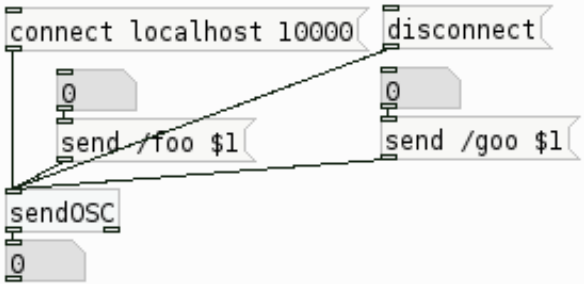
Open Sound Control (OSC) je další síťový protokol podporovaný Pd. Slouží k přenosu jednoduchých zpráv mezi různými multimediálními aplikacemi a má poměrně širokou podporu (Processing, Blender, Supercollider, vvvv, Ableton, Reaktor, Resloume, Quartz Composer - kompletní výčet najdete na Wikipedii). Jeho prostřednictvím lze tedy snadno vytvářet robustní multimediální systémy, ve kterých každý software obstarává oblast, pro kterou je určen, a jiným software pouze posílá zprávy ke zpracování.

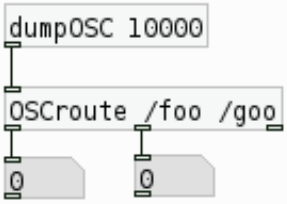
OSC na Wiki

Jeden příklad za všechny: V projektu Blendnik od Nicholase Porcaro má 3D software Blender na starosti vizuální a interaktivní složku a OSC protokolem posílá zprávy do Pd, které se starají o zvuk.

Blendnik

Objekty [dumpOSC] a [sendOSC] jsou podobné jako [netreceive] a [netsend], jen slouží k přenosu zpráv v rámci protokolu OSC. Rozdíl spočívá v tom, že jménům proměnných předchází znak "/". Objektem [OSCroute] zprávy rozbalujeme. Následující příklad demonstruje komunikaci pouze na lokálním hostiteli, snadno si jej ale upravíte tak, abyste ho mohli otestovat i v provozu na síti.





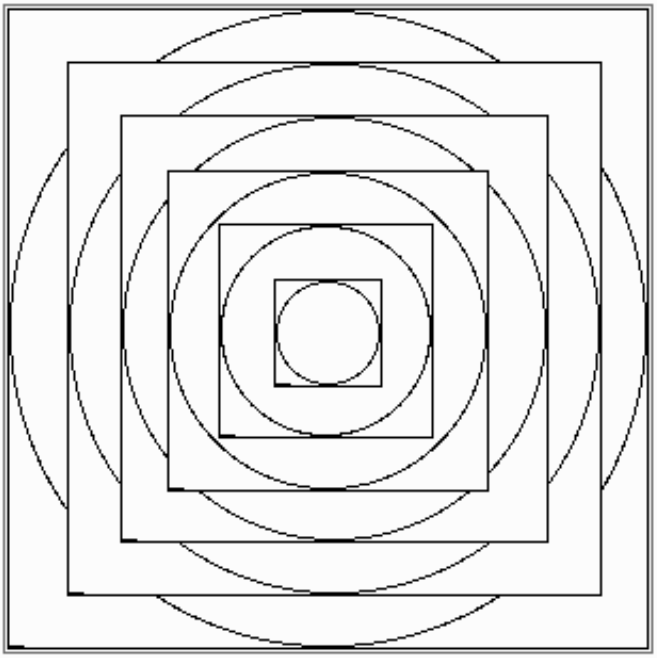
Komunikace patchů mezi sebou, ev. komunikace různých aplikací, v sobě nese řadu zajímavých přesahů. Samotná možnost poslat někomu zprávu po síti vypadá poněkud banálně, ale představte si, že se svými spoluhráči sdílíte stejné grafické rozhraní nástroje a všichni na něj v danou chvíli hrajete - zároveň v rozhraní vidíte akce vašich spoluhráčů. V případě reálného piána opravdu nejde o nic revolučního, ale pokud jde o hraní na software, tak zde posun je. Společného hraní se může účastnit velký počet hudebníků a můžou na určitý prvek nástroje sáhnout ve stejný moment - "přetahování" se o pozici posuvníku se tak může stát kompozičním prvkem.

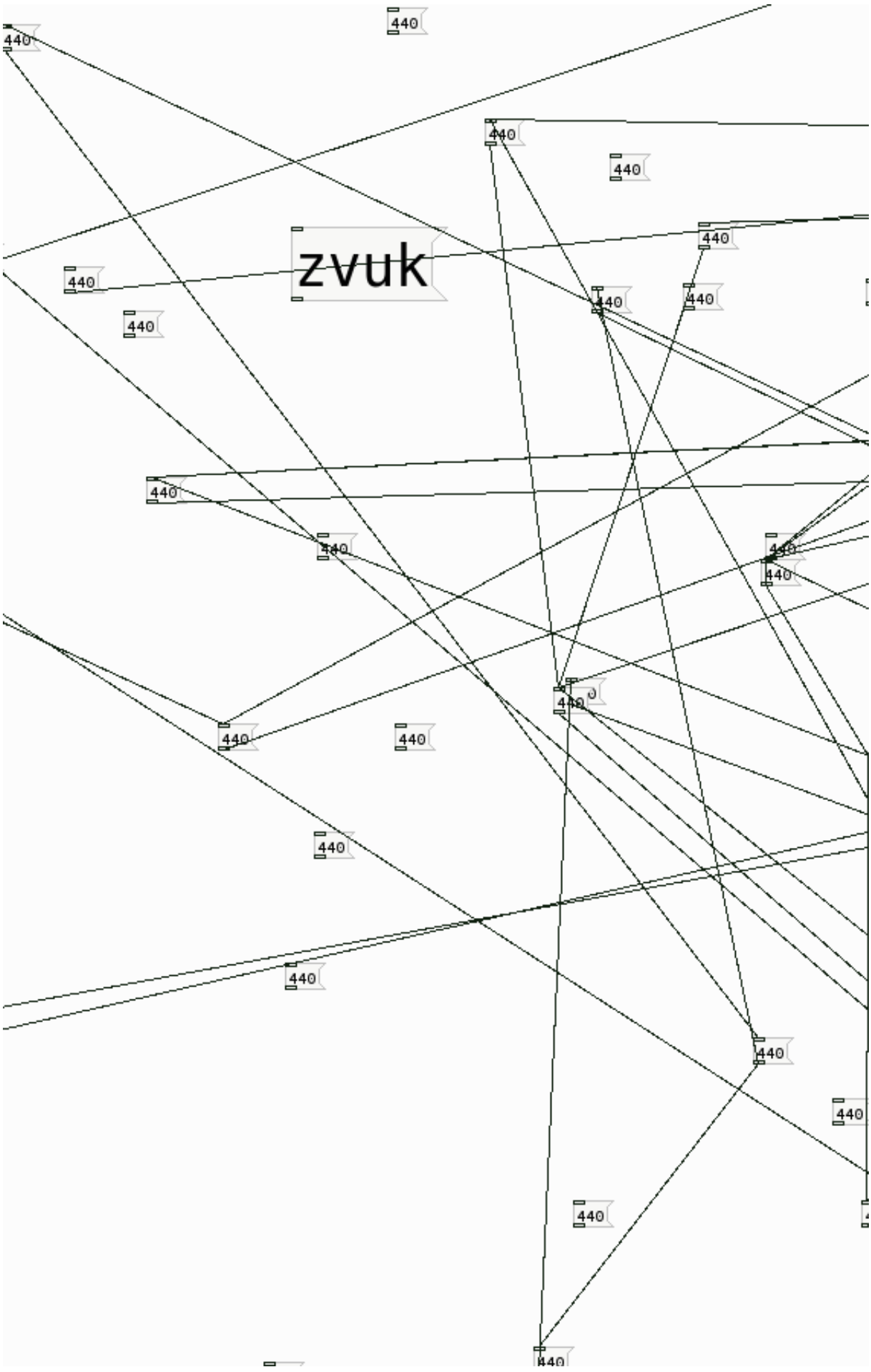
V tomto ohledu Pd a další software nástroje zprostředkovávají návaznost na hudební praxi takových skupin, jako byli např. The Hub.

Hezkým příkladem rozsáhleho CRNMME systému (Collaborative Realtime Networked Music Making Environment) sloužícího pro hudební improvizaci je projekt netpd. V jednodušší verzi jsem podobných principů využil při konstrukci nástroje pro hudební těleso Ty & Ra. O obou projektech se ještě zmíníme.

Link na netpd

V tuto chvíli za sebou máme průzkum základní "gramatiky" a "slovní zásoby" vizuálního programovacího jazyka Pure Data. Doufám, že se vám podařilo probíraná témata rámcově uchopit. Pokud vám ve své obecnosti přišla látka nejasná, nebo obtížně aplikovatelná, pak vězte, že před sebou máme kapitoly týkající se zvuku a obrazu, na kterých obecnost předchozí kapitoly "usadíme" kontaktem s konkrétními problémy.




zvuk
440

zvuk

Po průzkumu obecných základů Pd se nyní ponoříme do studia problematiky zvuku. Pokud hovoříme o vytváření a manipulaci signálu, ke kterému dochází v počítači, nebo na jiné digitální platformě, dotýkáme se oblasti označované zkratkou DSP (Digital Signal Processing), která v sobě zahrnuje velmi širokou škálu témat. Obecně můžeme říci, že DSP se zabývá problematikou zacházení se signálem, který je reprezentovatelný posloupností čísel.

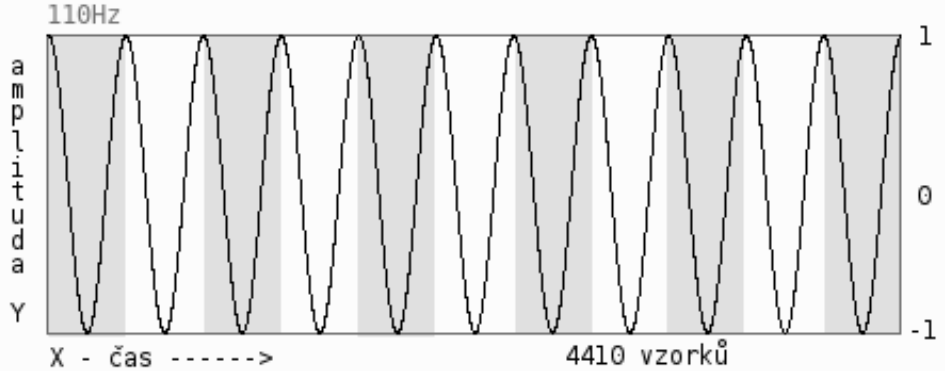
V rámci naší rukověti do problematiky DSP nevstoupíme "naplno", ale spíš lehce "zaťukáme na dveře". Seznámíme se se základy reprezentace signálu v počítači, probereme některé z přístupů v syntéze zvuku a práce se vzorky. Pokročilejší problematiku jako Fourierova analýza, práce se spektrem, fázový vokóder, "pich-shifting" atd. ponecháme stranou. Je možné, že se těmto tématům budeme detailně věnovat v rozšířené verzi s pořadovým číslem v. > 1.

Poučené čtenáře poprosím o shovívavost ke zjednodušením, k nimž jsem se v rámci srozumitelnosti a přístupnosti této rukověti uchýlil. Zvídavé čtenáře, toužící po přísných a detailních základech, pak odkazuji ke studiu původních pramenů v bibliografii. Spolehlivě vystačí na vyplnění jedné knihovny a několik let usilovného samostudia. Po nastudování této části rukověti byste nicméně měli být dostatečne vybaveni ke stavbě vlastních hudebních nástrojů a zvukových systémů.

ZVUK_A_FYZIKA

Z úhlu pohledu fyzikální akustiky můžeme o zvuku hovořit jako o kmitavém pohybu hmoty v pevném, kapalném a plynném prostředí, které vyvolává sluchový vjem. Když rozezníme úderem ladičku, uslyšíme tón, jenž je výsledkem rychlých změn tlaku vzduchu v našem uchu. Kmitání ladičky, stejně jako jakýkoliv jiný zdroj zvuku, způsobí to, že se molekuly vzduchu stlačují a rozvolňují. Pakliže toto kmitání doputuje s patřičnou intenzitou až k našemu bubínku a náš mozek je "v pořádku", slyšíme daný zvuk.

Pokud je kmitání pravidelné, interpretujeme ho jako tón s určitou výškou, pakliže jde o nepravidelné kmitání, hovoříme o hluku. K zobrazení zvuku se v akustice a DSP používá dvourozměrný graf (v Pd objekt pole). Jeho obdoby ostatně najdete vždy, když spustíte jakýkoliv zvukový editor a načtete v něm patřičný soubor. Vodorovná osa X reprezentuje čas a svislá osa Y pak amplitudu - krajní výchylku kmitající soustavy.



Na předchozí straně vidíme graf, ve kterém je na ose X vyneseno celkem 4410 hodnot, což představuje 1/10 vteřiny při standardní vzorkovací frekvenci (44100 vzorků za vteřinu). Zobrazený signál je pravidelné harmonické kmitání o frekvenci 110 Hz.

Počet kmitů neboli period za jednu vteřinu se nazývá frekvence. Jednotkou frekvence je Hz (hertz). V případě našeho grafu jsme periody označili střídáním šedých a bílých částí, takže snadněji vidíme, že do 1/10 vteřiny se jich při frekvenci 110 Hz vejde právě jedenáct.

V případě slyšitelného signálu, co se dispozice lidského ucha týče, se pohybujeme v rozmezí přibližně od 20 Hz do 20 kHz. Netopýři slyší ve škále 3 kHz až 120 kHz a delfíni dokonce 1 kHz až 130 kHz. S věkem se rozsah zmenšuje.

Jednoduchou analogií k modelovému zobrazení zvuku v grafu je kmitání membrány reproduktoru. Když bychom do reproduktoru ze zesilovače poslali signál o frekvenci 110 Hz, pak by jeho membrána za 1/10 vteřiny kmitala nahoru a dolů přesně tak, jak je zaznamenáno v grafu. Osa Y, neboli amplituda, zobrazuje míru vychýlení membrány v daném čase - ta souvisí s hlasitostí signálu.

DIGITÁL_JE_DISKRÉTNÍ

Název této podkapitoly neodkazuje na to, že by byl digitální signál nějak ohleduplný, ale na to, že jde o signál nespojitý. Dostáváme se tak k zásadnímu tématu reprezentace signálů v počítači a potažmo též k zajímavé filozofické otázce.

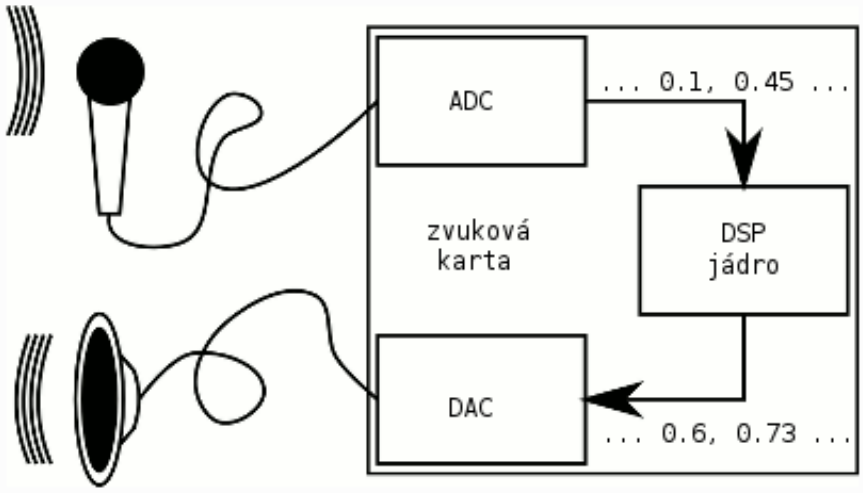
Prostor a čas se nám jeví jako kontinuum, jako cosi nepřetržitého a spojitého. Otázkou, zda to tak opravdu je, se zde nebudeme zabývat. V digitálním počítači dokážeme ale reprezentovat pouze omezené množství diskrétních, tj. počitatelných hodnot. Jak pak tedy něco takového jako spojitý pohyb v prostoru a čase (např. kmitání membrány) převést do počítače?

Víme, že jsme bytosti, které se nechají snadno ošálit. K tomu, abychom začali vnímat statický obraz jako pohyb, nám stačí rychle vystřídat 24 nepohyblivých obrázků do vteřiny, a k tomu, abychom uslyšeli zvuk, který může obsahovat všechny frekvence odpovídající možnostem našeho ucha, potřebujeme na jednu vteřinu 44100 hodnot neboli vzorků. Toto číslo má své odůvodnění a vychází z Nyquist-Shannonova teorému, ze kterého vyplývá, že vzorkovací frekvence musí být 2x větší než maximální frekvence, již chceme reprezentovat. 100 navíc souvisí s tzv. aliasingem a nedokonalostí filtrů, ale to bychom zabíhali již příliš do detailů.

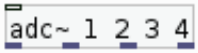
Pro naši potřebu postačí, když pojmu digitalizace porozumíme jako převodu spojitého (analogového) signálu do nespojité posloupnosti čísel. Jako bychom 44100x za vteřinu změřili vychýlení membrány reproduktoru a pak tyto hodnoty nanesli na osu X do našeho grafu.

Digitální signál je tedy "pouze" aproximací analogového signálu - digitalizací vždy dochází ke ztrátě informace. Míra ztráty ale může být téměř libovolně malá.

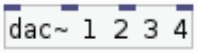
V počítači se o převod analogového signálu na digitální a vice versa starají dva typy konvertorů, které jsou součástí zvukové karty. První se označuje jako ADC (Analog Digital Convertor) a zajišťuje převod analogových signálů na digitální. DAC (Digital Analog Convertor) se, překvapivě, stará o opačný proces. Schematicky si funkci obou převodníků můžeme znázornit tako:



V Pd existují objekty [adc~] a [dac~]. Nejsou to přímo převodníky, ale zprostředkovávají nám vstupy a výstupy, které máme k dispozici díky zvukové kartě. Pokud je naše karta vícekanálová, pak tyto objekty můžeme specifikovat argumenty, jež označují jednotlivé vstupy a výstupy.



Vícekanálová zvuková karta se čtyřmi vstupy



a čtyřmi výstupy.

Když bychom se rozhodli v Pd naprogramovat systém, jehož funkcí by byla změna vstupního signálu v reálném čase (např. kytarový efekt), pak by v zásadě patch měl tuto podobu:

Z objektu [adc~] čteme vstupní digitalizovaný signál (hlas, kytara) a posíláme ho do subpatche, kde dochází k samotné manipulaci signálu.

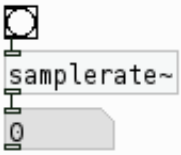
A posíláme ho zpět zvukové kartě, která digitální signál převede zpátky na analogový.

VZORKOVACÍ_FREKVENCE

O vzorkovací frekvenci jsme se již zmínili. Je to číslo informující nás o tom, kolik "měření" při převádění analogového signálu na digitální za jednu vteřinu pořizujeme. Standardně je to 44100 vzorků za vteřinu. Od hodnoty vzorkovací frekvence se odvíjí maximální reprezentovatelná frekvence v daném vzorku. V případě 44100 je to 22050 Hz, což dostatečně pokrývá námi slyšitelnou škálu.

Pokud vaše zvuková karta zvládá i vyšší vzorkovací frekvenci, můžete ji přes nastavení zvuku v menu Media změnit na 96000.Pak je ale proces vzorkování pro počítač samozřejmě náročnější.

Když budete později v Pd nahrávat zvukové vzorky do pole, mějte na paměti, že v jedné vteřině zvuku je standardně na ose X vyneseno 44100 údajů o amplitudě. Pocítíte to, když se v editačním módu pole s delším zvukem pokusíte přemístit. Tato akce Pd obvykle hodně zaměstná a na slabších počítačích může trvat až několik vteřin. Pole se zvukovými vzorky je vhodné schovávat do subpatchů, takže Pd nejsou zaměstnána jejich vykreslováním.



Objekt [samplerate~] po obdržení bangu na vstup vypíše vzorkovací frekvenci, kterou spuštěná Pd aktuálně používají.

BITOVÁ_HLOUBKA

Vzorkovací frekvence souvisí s maximální reprezentovatelnou frekvencí a v grafu s osou X. Nyní se zaměříme na osu Y. Na ní jsou vyneseny hodnoty týkající se intenzity signálu.

Jak jemnou škálu na ose Y dokážeme vyjádřit souvisí s počtem bitů, které vyhradíme číslu, jež pro tento účel chceme použít. Osmibitovým číslem vyjádříme maximálně hodnoty 0..255, šestnáctibitovým 0..65535, třicetidvoubitovým 0..4294967295. Vidíme, že se vzrůstajícím počtem bitů obdržíme širší číselný rozsah.

Na dvou bitech vyjádříme čtyři hodnoty.

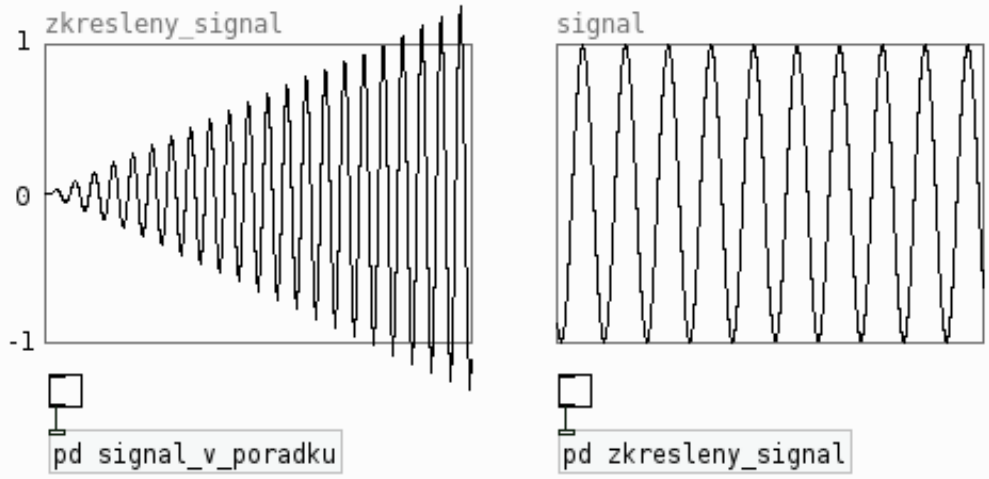
0 0
0 1
1 0
1 1

Jemnost číselné škály na ose Y ve zvuku souvisí s dynamickým rozsahem - tj. rozdílem mezi nejtišší a nejhlasitější úrovní daného signálu. Čím je bitová hloubka vyšší, tím jemnější škálu v dynamice signálu dokážeme popsat. Pro zvukový signál jsou v Pd vyhrazena 32 bitová čísla, což odpovídá dynamickému rozsahu 192 dB. Práh slyšitelnosti je 0 dB a start rakety saturn V měl údajně 140 dB (práh bolesti). Dynamický rozsah, který díky 32 bitové hloubce Pd nabízí, je dalece za hranicí praktické využitelnosti. Reálně se dynamický rozsah odvíjí od zvukové karty, která je obvykle na výstupu limitována hloubkou 16 nebo 24 bitů.

V DSP je normou, že hranice amplitudy jsou stanoveny na interval -1..1, který dokážeme popsat s jemností odpovídající přibližně 4.3 biliónu různých hodnot.

Prakticky si pod hodnotou na ose Y můžete představit výkyv membrány reproduktoru v daném čase. Nula je na ose Y uprostřed a označuje klidový stav reproduktoru, hodnoty -1 a 1 pak maximální výkyv membrány.

V levém poli, které má 44100 prvků, vidíme signál o frekvenci 50Hz. S postupem času se zesiluje až do té míry, že jeho intenzita překročí interval -1..1. V tu chvíli dojde ke zkreslení signálu.

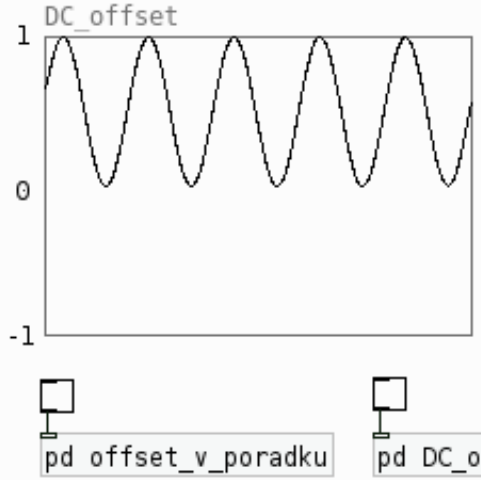


Absolvujte malé sluchové cvičení, které vám pomůže rozpoznat zkreslený signál. Když zapnete subpatch [pd signal_v_poradku], uslyšíte (a v poli signal i uvidíte) tón o frekvenci 440 Hz s amplitudou v rozsahu -1..1. Subpatch vypněte a zapněte druhý, ten přehrává tón o stejné frekvenci, ale jeho amplituda "vyskakuje" z pole. Reproduktor tento přesah již neumí interpretovat a uslyšíme zkreslení. Subpatche případně prozkoumejte. Dokážete identifikovat objekt, který způsobuje zkreslení?

Zkreslení můžeme v některých případech využít jako efekt, jinak je ale budeme považovat za nežádoucí prvek. Spíš se vždy budeme snažit "ukočírovat" signál tak, aby hranice pole nepřekročil.

DC_OFFSET

Další nežádoucí jevy v signálu souvisí s jeho posunem od nuly (tzv. DC offset). Signál by měl vždy rovnoměrně kmitat kolem nuly. Opět si ukážeme názorný příklad.

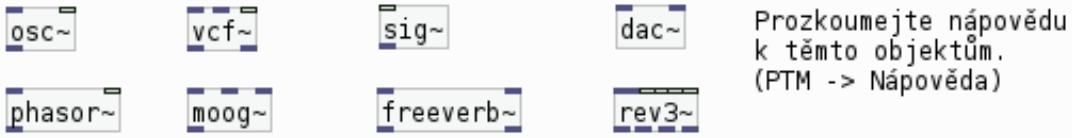


V prvním případě slyšíme signál o frekvenci 220 Hz s poloviční amplitudou, který kmitá rovnoměrně kolem nuly. Ve druhém případě uslyšíme stejný signál, ale vidíme, že je posunut nahoru. Membrána reproduktoru tedy kmitá pouze v horní polovině a neprochází klidovým stavem. DC offset má nežádoucí vliv na dynamiku signálu a může být též příčinou zkreslení.

Na tomto příkladu je hezky vidět, že ne všechny chyby v signálu jsou slyšet. Je dobré vždy signál i přehlédnout.

základy

Doposud jsme používali převážně objekty, které na vstupech a výstupech pracují se zprávami typu číslo, symbol a seznam. Zvukové objekty se od těchto liší tím, že nepracují se zprávami, ale se zvukovým signálem. Poznáme je snadno podle toho, že jsou v názvu doplněny vlnovkou. Jsou to např. tyto:



Prozkoumejte nápovědu k těmto objektům. (PTM -> Nápověda)

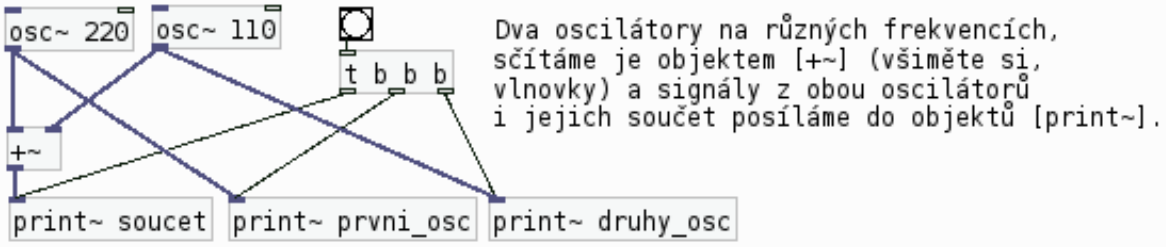
Objekty pracující se zprávami a zvukové objekty se liší také charakterem kabelu - zvuk "teče" po těch silnějších. V Pd-extended od verze 0.44 a výš najdeme v menu Upravit nový nástroj Magic Glass, kterým můžeme sledovat, jaký signál nebo zpráva po kabelech zrovna putuje. Když zapneme DSP i Magic Glass (CTRL + ALT + g), přepneme se do editačního módu a pak najedeme kurzorem na zvukový kabel, uvidíme aktuální amplitudu signálu. V menu Upravit můžete zapnout také nástroj Automatické Tipy (CTRL + ALt + g). Ten nás zase v editačním módu informuje o objektech a charakteru jejich vstupů a výstupů. Informace se zobrazují v levém dolním rohu patche.

Pokud byste se pokusili zapojit kabel z výstupu objektu [osc~ 2] do vstupu objektu [print], tak se vám to nepodaří. V Pd konzoli by se objevila hláška: "can't connect signal outlet to control inlet". Objekt [print] totiž slouží ke zpracování zpráv a ne signálů. Signál budeme posílat pouze objektům, které si s ním umí poradit. Že by [print~]?

BLOKY

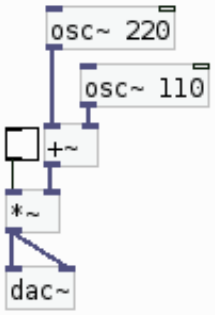
Již jednou jsme se zmínili o tom, že název Pure Data odkazuje k tomu, že vše, s čím v počítači zacházíme, je reprezentovatelné nějakým číslem - platí to samozřejmě i pro zvukový signál. Ten je ve skutečnosti velmi rychlým tokem vzorků (tj. číselných hodnot), se kterými lze provádět různé matematické operace.

Když zapneme DSP, začnou Pd "počítat zvuk". Víme již, že za jednu vteřinu Pd zvládnou standardně zpracovat 44100 vzorků. Nedělají to "po jednom", ale seskupují vzorky do tzv. bloků. Podívejme se podrobně, jak to vypadá, když např. chceme sečíst dva audiosignály.



Dva oscilátory na různých frekvencích, sčítáme je objektem [+~] (všiměte si, vlnovky) a signály z obou oscilátorů i jejich součet posíláme do objektů [print~].

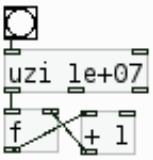
Když kliknete na bang, objeví se v konzoli spousta čísel (případně si konzoli rozšiřte tak, abyste viděli výstup z obou oscilátorů i součet). Objekt [print~], na rozdíl od objektu [print], umí přijmout signál a vytisknout jeden blok vzorků do konzole. Jeden blok standardně obsahuje 64 vzorků a vskutku - v konzoli uvidíme 3 x 64 různých hodnot, které odpovídají amplitudám jednotlivých oscilátorů a jejich součtu. Zkuste si sečíst hodnoty z oscilátorů ve stejném sloupci a řadě - výsledek by měl odpovídat hodnotě, kterou najdete v konzoli v položce součtet. Součet oscilátorů si také poslechněte. ---------->



Zvukové objekty zpracovávají signál po blocích o 64 vzorcích, což při vzorkovací frekvenci 44100 vzorků za vteřinu odpovídá zpracování jednoho bloku za cca. 1.45 milisekundy. Ve srovnání s objekty, které operují se zprávami, toho mají "víc na práci".

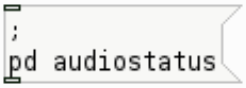
Objekty, které pracují se zprávami, zaměstnávají procesor počítače pouze tehdy, když je po nich vyžadována nějaká akce, oproti tomu zvukové objekty začnou počítat proud vzorků ve chvíli, kdy zapneme DSP. Procesor zatěžují i objekty, které zrovna "hrají" jen samé nuly.

Pd fungují tak, že vždy nejprve spočítají jeden blok ve zvukových objektech, a pak se ev. věnují výpočtům v objektech, které pracují se zprávami. Během počítání jednoho bloku (cca 1.45 ms) nemůže být proveden žádný výpočet související se zprávami - ty se odhrávají "mezi" bloky. Pokud bychom provedli nějakou extrémní akci, např. inicializovali bang, jenž pošle deset miliónů bangů do počítadla, tato operace bude trvat déle, než je spočtení jednoho bloku v audio objektech a ve zvukovém bufferu zůstane "viset" déle jeden blok - uslyšíme pak charakteristickou chybu.



Zapněte DSP a příklad s dvěma oscilátory vpravo nahoře. Když pak klikneme na bang vlevo, vyvoláme tím akci, která na pár milisekund "zmrazí" zvuk. V konzoli uvidíme hlášku informující nás o chybě zvukového I/O.

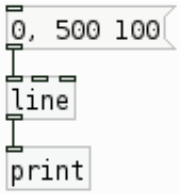
O poslední chybě, související s počítáním zvuku v reálném čase, se můžete informovat následující interní zprávou.



Vypíše do konzole typ chyby a čas uplynulý od jejího zaznamenání.

Délku bloku můžeme v některých případech změnit příkazem [block~]. Podívejte se do subpatchů napravo a klikněte v nich na bang. Uvidíte, že do konzole se vypisuje různý počet hodnot.

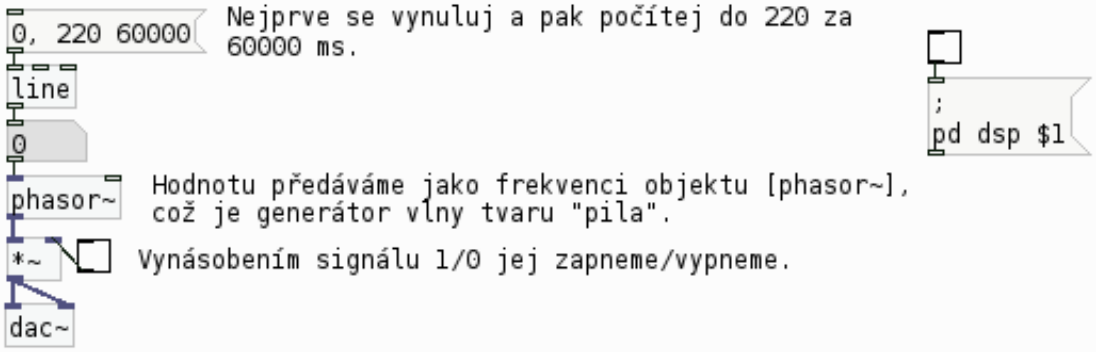
V jak dlouhých intervalech probíhají výpočty ve zvukových objektech se odvíjí od délky bloku a vzorkovací frekvence. Standardně tedy zvukové objekty "tikají" rychlostí 1.45 ms. Objekty pracující se zprávami "tikají" pomaleji. Např. objekt [line] má přednastavenu rychlost 20 ms.



Do objektu [line] posíláme dvě zprávy, nejprve ho nastavíme na nulu a pak mu řekneme, ať od této hodnoty počítá do 500 za 100 ms. V konzoli pak uvidíme hodnoty 0, 100, 200..500, což odpovídá rychlosti "tiku" 20 ms.

FREKVENCE

Frekvence, jakou nějaká soustava kmitá, má vztah k tónové výšce. Kmitá-li např. oscilátor rychlostí 440 Hz, odpovídá to tónu komorního A, podle kterého se ladí orchestr. Tón C2 odpovídá frekvenci 523 Hz. Rychlejší frekvence odpovídá vyšším tónům. Provedeme nyní půlminutový poslechový experiment, jehož předmětem bude přechod z infrazvukové oblasti, kdy soustava kmitá pomaleji než 20 Hz, do slyšitelné části zvukového spektra.



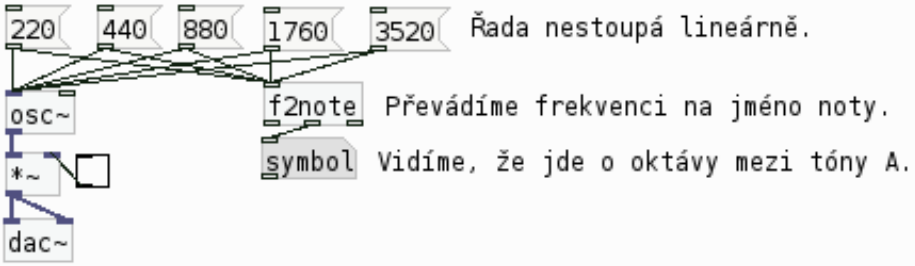
Nejprve se vynuluj a pak počítej do 220 za 60000 ms.

Hodnotu předáváme jako frekvenci objektu [phasor~], což je generátor vlny tvaru "pila".

Vynásobením signálu 1/0 jej zapneme/vypneme.

Ve fázi, kdy objekt [line] počítá hodnoty od 0 do 20, slyšíme pouze rytmus, při vyšších frekvencích se ale rytmus promění v tónovou výšku.

Zvláštností našeho ucha je to, že není "vyladěno" lineárně, ale logaritmicky. V následujícím příkladu uslyšíme frekvence, které odpovídají intervalu oktávy. Další oktáva je vždy dvojnásobkem předchozí hodnoty.

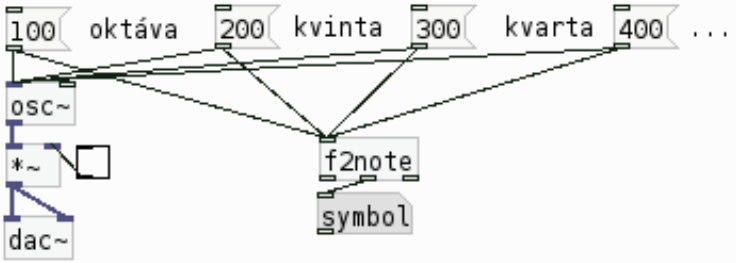


Řada nestoupá lineárně.

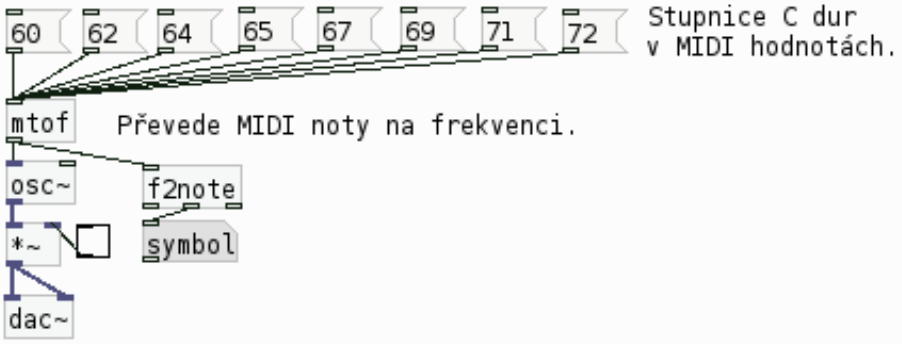
Převádíme frekvenci na jméno noty.

Vidíme, že jde o oktávy mezi tóny A.

Když do oscilátoru budeme posílat hodnoty v lineární posloupnosti, dostaneme tím další intervaly: kvintu, kvartu atd. Řada stoupá lineárně, ale naše ucho interpretuje rozdíly mezi jednotlivými frekvencemi jako čím dál menší.

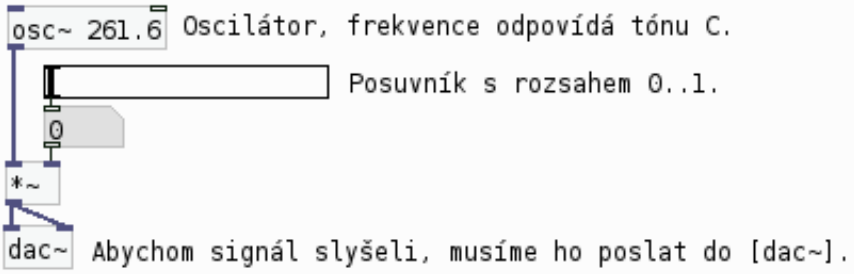


Pokud budete v Pd chtít používat klasickou tónovou řadu, nemusíte si pamatovat jednotlivé frekvence tónů, ale můžete využít objekt [mtof], který převádí hodnoty MIDI not na frekvenci. Pd MIDI normu samozřejmě podporují, takže syntetizéry postavené v Pd lze řídit MIDI kontrolerem nebo hardwarovým MIDI sekvencerem.

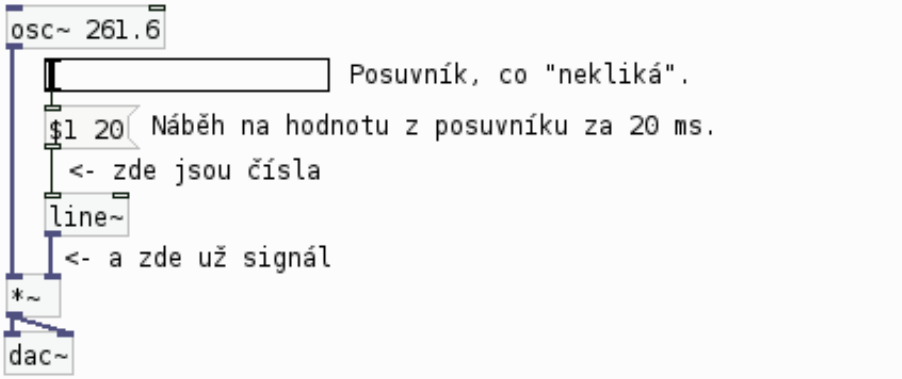


HLASITOST

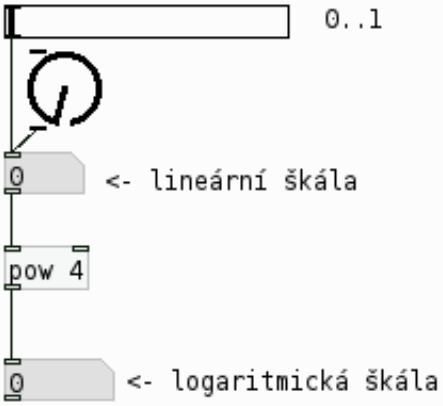
Hlasitost zvuku, což je subjektivní veličina, souvisí s amplitudou, která je objektivně měřitelná. V předchozích příkladech je vždy před objektem [dac~] předřazen objekt [*~], jímž signál násobíme. Když signál vynásobíme nulou, bude jeho amplituda nulová a nic neuslyšíme. Když jej vynásobíme jedničkou, bude jeho amplituda mít rozsah -1..1. Pokud bychom jej násobili vyšší hodnotou, amplituda "vyskočí" nad hranici -1..1 a uslyšíme zkreslení. Vypínač posílá hodnoty 0 a 1, takže signál vždy úplně ztiší nebo zesílí. V následujících příkladech nahradíme vypínač posuvníkem.

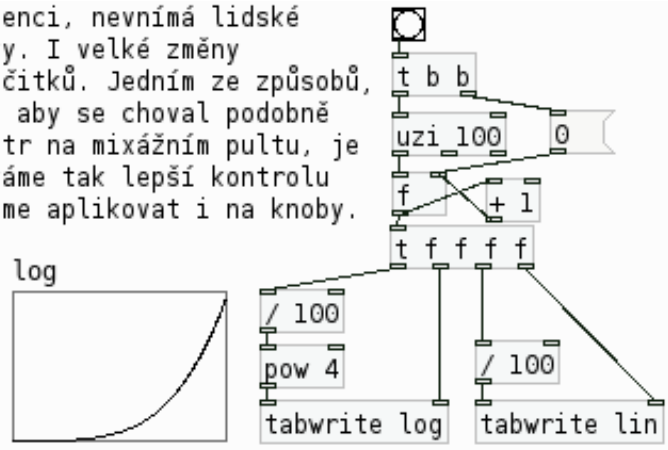


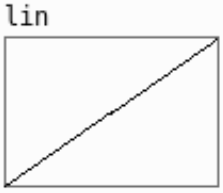
Signál bychom ale neměli míchat s číselnými zprávami, i když to Pd umožňují. Když posuvníkem totiž manipulujeme příliš rychle, uslyšíme charakteristické a nežádoucí kliknutí. Abychom se této chybě vyhnuli, budeme amplitudu ovlivňovat objektem [line~]. Ten se chová podobně jako [line], jen jeho výstupem není číslo, ale signál. Podrobněji si ho popíšeme v kapitole o obálkách.



Hlasitost zvuku, podobně jako frekvenci, nevnímá lidské ucho lineárně, ale také logaritmicky. I velké změny v podnětech způsobují malé změny počitků. Jedním ze způsobů, jak přimět posuvník o rozsahu 0..1, aby se choval podobně jako logaritmický tahový potenciometr na mixážním pultu, je "prohnat" ho objektem [pow 4]. Získáme tak lepší kontrolu nad hlasitostí. Podobný postup můžeme aplikovat i na knoby.



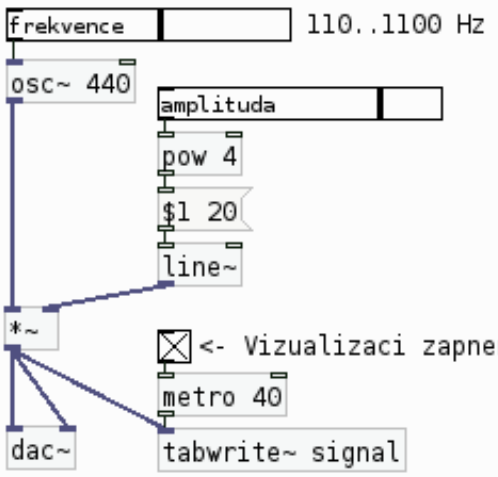


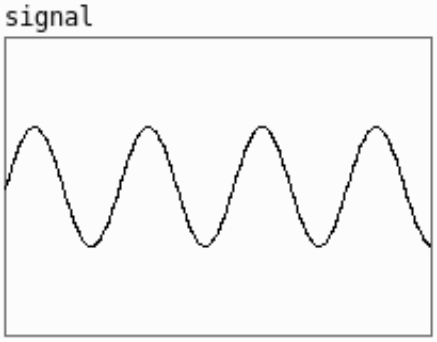


Patch, který vykresluje rozdíl mezi lineárním a logaritmickým průběhem funkce do pole.

VIZUALIZACE_SIGNÁLU

Ještě před tím, než se začneme věnovat podrobněji jednotlivým generátorům signálů, ukážeme si, jak jednoduše signál zobrazit v poli. Slouží k tomu objekt [tabwrite~], jehož argumentem je název pole, do kterého signál zapisujeme. V tomto případě je to pole s názvem "signal". Signál se do pole zapíše pouze když do [tabwrite~] posíláme signál a inicializujeme ho bangem. Abychom vykreslování viděli kontinuálně, do [tabwrite~] posíláme bangy intervalu 40 ms z objektu [metro].





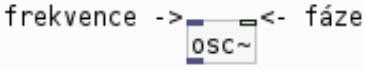
Pole o 256 prvcích. Vidíme v něm průběh amplitudy signálu v čase.

Vizualizace signálu je dobrým pomocníkem při odladňování chyb a normalizaci. Poslouží nám ale také k názornému pochopení probrané látky týkající se frekvence a hlasitosti. Když budeme zvyšovat posuvníkem frekvenci, perioda kmitání se bude zkracovat - oscilátor kmitá rychleji a uslyšíme vyšší frekvenci. Stejně když budeme manipulovat s posuvníkem ovládajícím amplitudu, bezprostředně uvidíme, jak signál na ose Y roste nebo se zmenšuje. Tuto techniku není vhodné využívat při živém hraní nebo když potřebujeme plný výkon počítače. Vykreslování signálu do pole totiž procesor poměrně zatěžuje. Zkuste si z uvedeného patche vytvořit abstrakci.

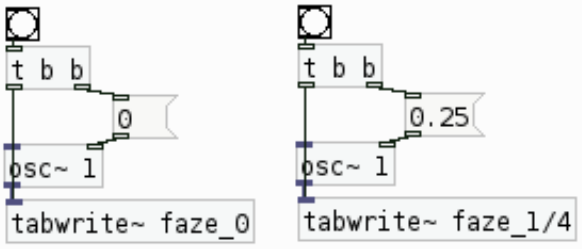
generátory_signálů

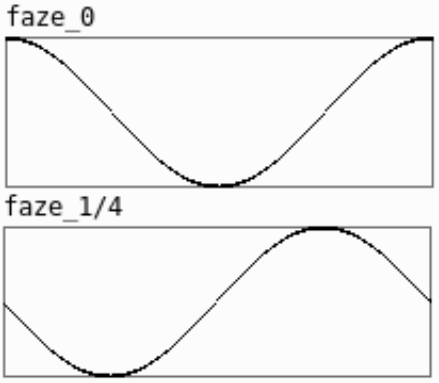
Předpokladem umění syntézy zvuku, slučování různých signálů, je seznámení se s "materiálem". V této kapitole prozkoumáme několik základních generátorů signálů, jejichž kombinací lze dosáhnout ve zvuku bohatších barev. Známe již dva generátory signálu: objekty [osc~] a [phasor]. Než se jim budeme věnovat jednotlivě, řekneme si ještě něco málo o fázi a spektru.

FÁZE



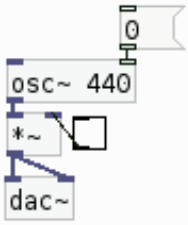
K nastavení fáze u pravidelně oscilujících generátorů slouží pravý vstup, kterému posíláme hodnotu v rozsahu 0..1. Co nastavení fáze znamená bude zjevnější z následujícího příkladu:





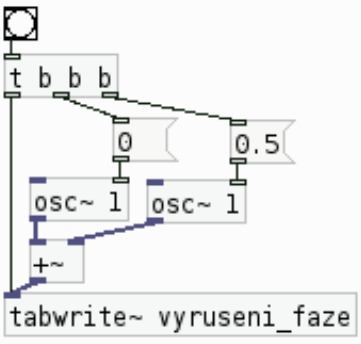
Prvním bangem objektu [osc~] specifikujeme fázi a druhým inicializujeme zápis do pole.

Objekt [osc~] "hraje" goniometrickou funkci cosinus, jejíž jednu periodu vidíme v poli s názvem "faze_0". První prvek pole na ose X má hodnotu 1. Ve druhém poli pak vidíme tutéž funkci, ale s posunem fáze o 1/4 doprava, takže začíná na nule. Když se vrátíme k analogii s reproduktorem, s pomocí fáze můžeme reproduktoru "říct, kam má skočit". Fáze nemá v tomto případě vliv na frekvenci ani hlasitost. Pokud ji ale nastavujeme "za běhu", snadno se stane, že dojde k charakteristickému kliknutí.



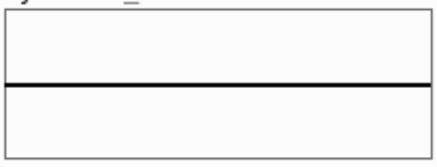
<- Fázový posun "na živo" způsobí kliknutí. To se ozve, když je zrovna reproduktor vychýlen do záporné poloroviny a my jej vynulováním fáze přinutíme prudce změnit polohu.

Přesnější práci s fází využijete později zejména v souvislosti s nízkofrekvenčními generátory, které se používají např. ke kontrole filtrů a parametrů různých efektů.



Když posuneme fázi stejné frekvence o jednu polovinu, takže mají navzájem protichůdný průběh, a poté je sečteme, navzájem se vyruší.

vyruseni_faze



SPEKTRUM

Doposud jsme se nezmínili ještě o jedné důležité složce zvuku, kterou je třeba uvést, a tou je jeho barva (témbr), neboli spektrální charakteristika.

Klavír i klarinet mohou hrát stejný tón o stejné frekvenci, a přeci je dokážeme jasně rozlišit - právě díky různé charakteristice jejich barvy. Když rozezníme strunu klavíru, výsledný tón není jen jedna frekvence, ale součet celé řady dalších, tzv. vyšších harmonických frekvencí. Poměr amplitud těchto vyšších harmonických frekvencí je pro každý nástroj jiný.

Kdybychom se na signál z klavíru a klarinetu podívali s pomocí našeho "osciloskopu", který jsme si postavili v předchozí kapitole, viděli bychom, že kmitají stejně rychle, ale průběh vlny mají odlišný. Tvar průběhu vlny určuje zabarvení daného tónu a na následujících stranách uvidíme i uslyšíme, že různé generátory signálu vytvářejí různé tvary vln a mají různý charakter.

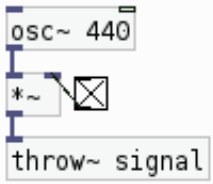
Z důvodů názornosti budeme používat rozšířenou verzi "osciloskopu", který kromě průběhu vlny bude v druhém poli zobrazovat i zvukové spektrum. Na osu X se vynáší frekvence a pokrývá rozsah 20 Hz až 22 kHz. Na osu Y se pak vynáší amplituda jednotlivých složek. Abstrakce se jmenuje [analyzer] a je součástí zdrojových kódů k rukověti. Její funkce je spíš ilustrativní. Čtenářům, kteří by chtěli spektrum prozkoumávat v detailnějším rozlišení, bych doporučil multiplatformní open-source aplikaci SonicVisualiser nebo aplikaci Baudline (pouze Linux a MacOS) pro analýzu v reálném čase.

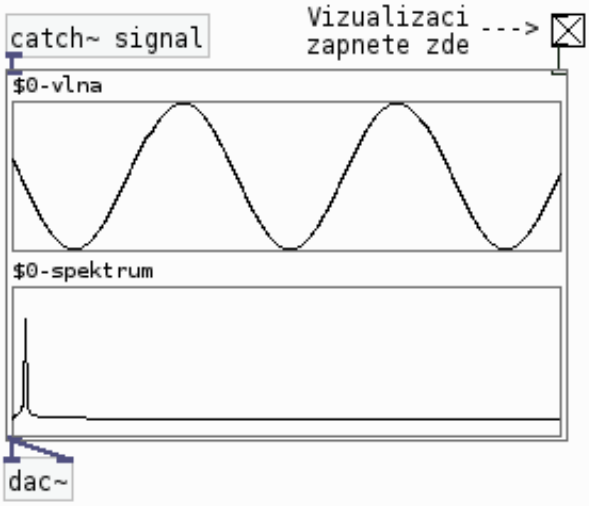
Sonic Visualiser

Baudline

[OSC~]

Základním zvukovým generátorem v Pd je objekt [osc~]. Jeho výstupem je vlna ve tvaru kosinusoidy. Má "měkký" charakter. Ve spektru, kromě základní frekvence, nevytváří žádné další harmonické složky. Levý vstup slouží pro předání frekvence a pravý pro nastavení fáze - tak je tomu ostatně i u řady ostatních periodických generátorů.



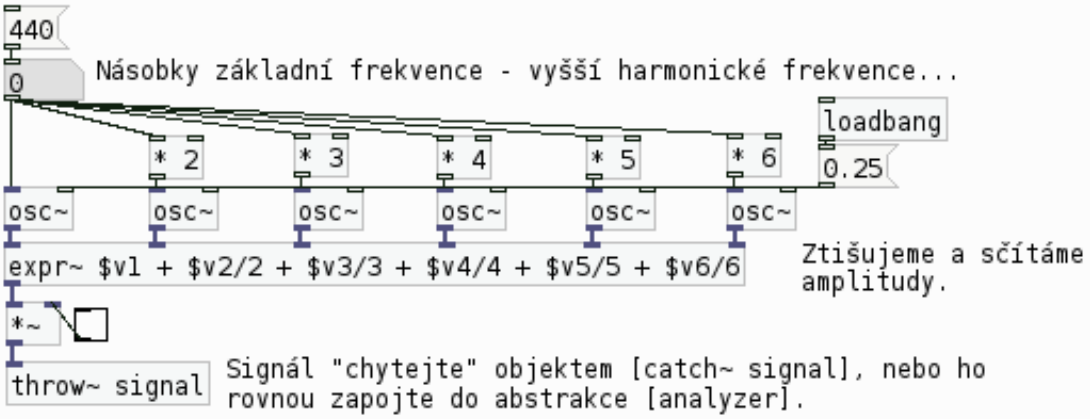


Poslechněte si oscilátor a zároveň zapněte vizualizaci. Povšimněte si tvaru vlny a charakteru spektra. Ve spektru uvidíte pouze jeden vrchol, který odpovídá frekvenci 440 Hz. Žádné jiné složky v něm nebudou.

Abstrakce [analyzer] má dva vstupy: levý je pro signál a pravý zapíná nebo vypíná vizualizaci. Výstup je stejný jako vstupní signál a můžeme ho zapojit do [dac~]. V tomto případě do abstrakce posíláme signál "bezdrátově" objektem [throw~] a "chytáme" ho objektem [catch~].

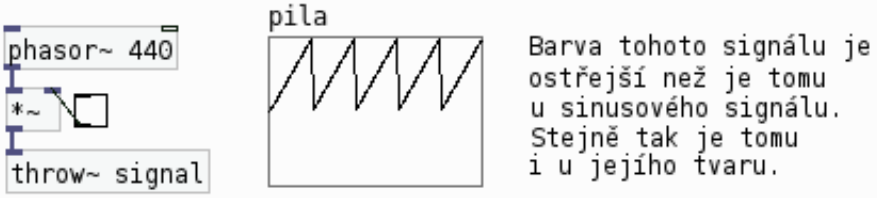
Součtem různých počtů tohoto jednoduchého signálu o různých frekvencích a amplitudách lze vytvořit jakýkoliv jiný složitější signál a naopak: složitý signál je možné rozložit do jednotlivých sinusových kmitů. Matematický aparát pro tyto operace vytvořil francouzský fyzik a matematik J. B. Fourier. Obecně se dá říct, že čím je tvar vlny signálu komplexnější, tím více oscilátorů je potřeba k jeho konstrukci.

Vlnu, která se přibližuje tvaru "pily", např. vytvoříme, když budeme sčítat násobky vyšších harmonických frekvencí, jejichž amplitudy se zmenšují vůči amplitudě základního kmitočtu v poměru shodném s pořadím harmonické. Tzn. že druhá harmonická má amplitudu dvakrát menší než základní frekvence, třetí harmonická má amplitudu třikrát menší atd...



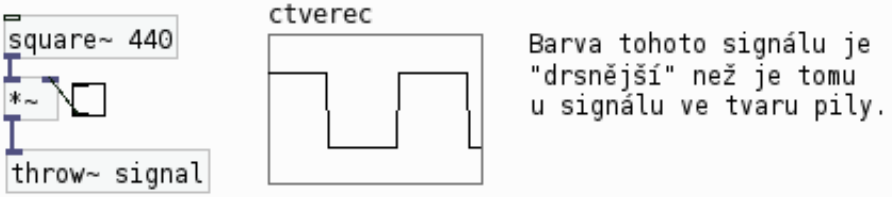
[PHASOR~]

Bylo by poněkud nepraktické, kdybychom museli všechny ostatní tvary vln vytvářet sčítáním řady oscilátorů. Proto jsou v Pd pro tyto případy dostupné odpovídající objekty. [phasor~] je tzv. unipolární generátor - to znamená, že osciluje pouze v intervalu 0..1 a vytváří vlnu, které se říká "pila". Z předchozího příkladu víme, že ve spektru obsahuje všechny násobky základní frekvence. Prozkoumejte charakteristiku [phasor~]u opět s pomocí abstrakce [analyzer].



[SQUARE~]

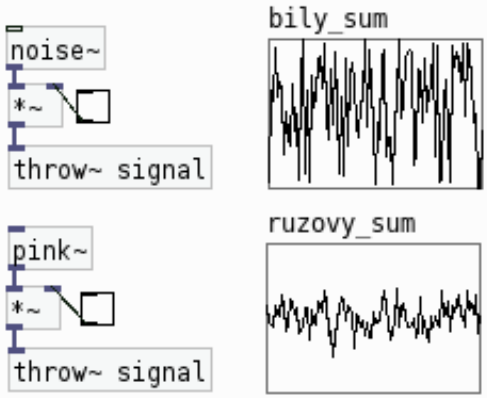
Samotný název objektu [square~] napovídá, že generuje vlnu ve tvaru čtverce. Ve spektru obsahuje pouze liché násobky základní frekvence.



[NOISE~]_A_[PINK~]

Co se týče neperiodických signálů a generátorů šumů, jsou v Pd k dispozici objekty [noise~] a [pink~]. [noise~] je generátor tzv. bílého šumu, což je náhodný signál s nepravidelnou vlnou. V jeho spektru jsou zastoupeny všechny frekvence.

Předchozí tvary vln, které jsme si ukázali, uplatníme např. v tzv. aditivní syntéze, kdy jednotlivé tvary vln sčítáme a tím vytváříme nové kvality v barvě signálu. Bílý šum pak uplatníme v tzv. subtraktivní syntéze, kdy z jeho plnosti budeme odčítat s pomocí filtrů určitá pásma.

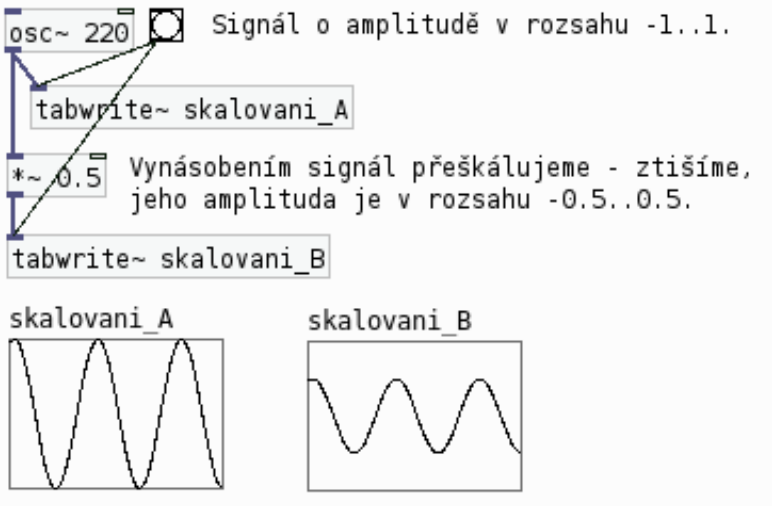


Když srovnáte v poslechu i ve spektru bílý a růžový šum, zjistíte, že růžový šum má potlačeny vyšší složky. Je "vyladěn" na charakteristiku lidského ucha, které nevnímá všechny slyšitelné frekvence se stejnou citlivostí.

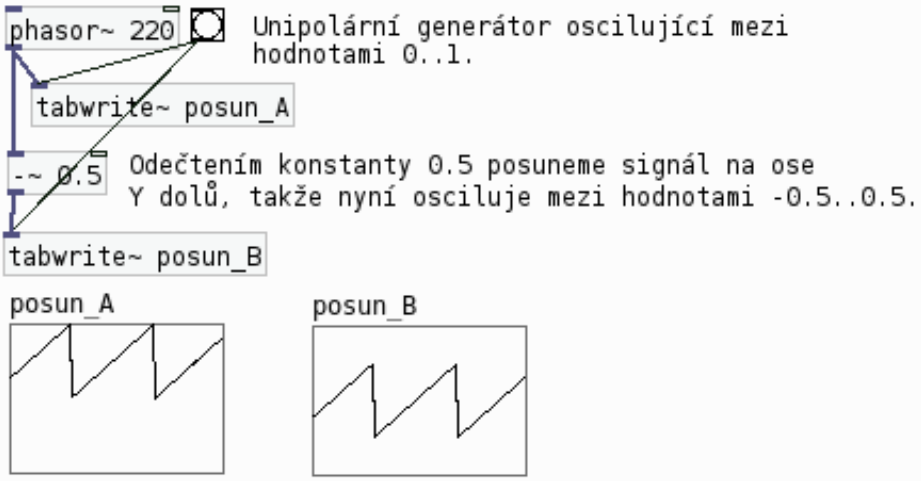
tvarování_vln

Tvarování vln (wave shaping) je praxe vytváření nových vln za účelem obdržení nových témbrů. Touto kapitolou se vám otevírá prostor pro vlastní experimentování s vlnovou "alchymií". Za výchozí "materiál" nám dobře poslouží vlny, se kterými jsme se seznámili. Tvarovat je budeme s pomocí několika málo jednoduchých operací. Půjde o škálování, posun, inverzi, ořez a doplněk signálu.

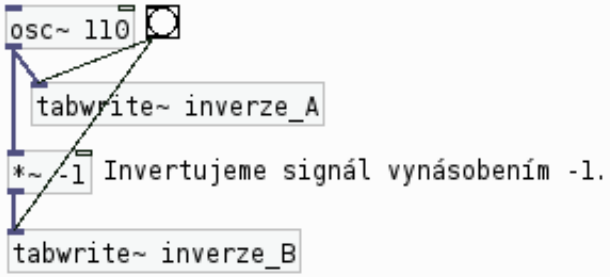
Škálování provádíme objektem [*~]. Pokud signál vynásobíme nějakou číselnou konstantou, ovlivní to jeho amplitudu. Obvykle jde o rozsah 0..1.

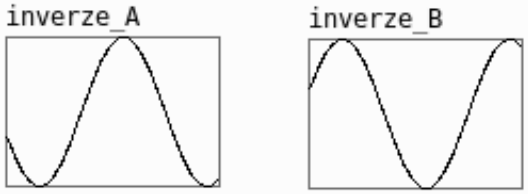


Posunem máme na mysli změnu polohy signálu na ose Y. Poslouží nám k tomu objekty [+~] a [-~]. Obvykle signál budeme chtít posunout v rozmezí intervalu -1..1.

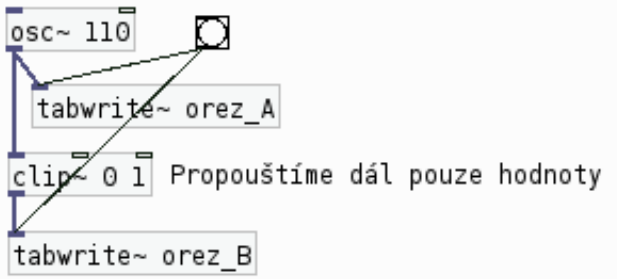


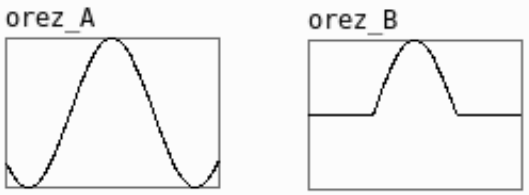
Inverze je obrácením průběhu signálu a provedeme ji prostým vynásobením signálu hodnotou -1. Invertovaný signál prochází na ose Y bodem 0 ve stejném okamžiku jako neinvertovaný signál, ale jeho směr je opačný. Inverze signálu posune jeho fázi o hodnotu 0.5.



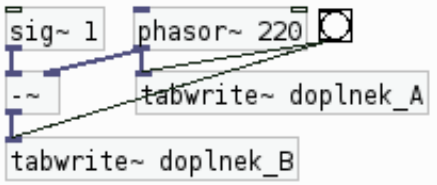


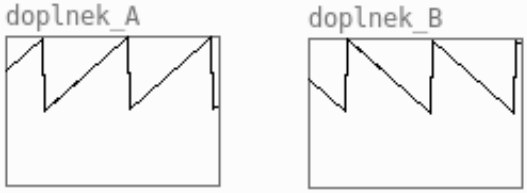
Oříznout sigál můžeme objektem [clip~]. Jeho argumenty jsou dvě hodnoty, kterými určujeme, jaký rozsah z osy Y propustíme dál.





Konečně doplňkem signálu, jehož amplituda se pohybuje v rozsahu 0..1, budeme rozumět odečtení tohoto signálu od konstantního signálu o hodnotě 1. Ten si vygenerujeme s pomocí objektu [sig~ 1]. Když původní signál stoupá, jeho doplněk zrcadlově klesá. Tato operace se odlišuje od inverze, i když podobně jako ona obrací směr. Doplněk signálu nemění znaménko a je definován pouze pro pozitivní hodnoty.

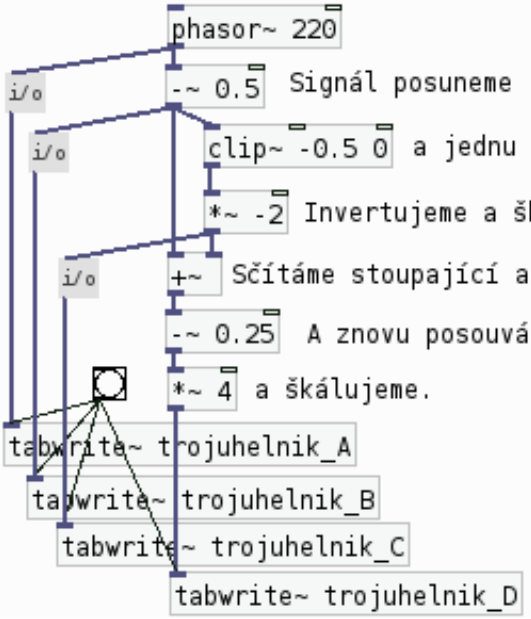


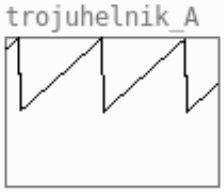


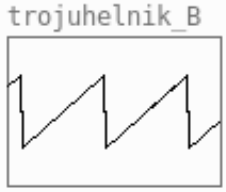
TROJÚHELNÍK

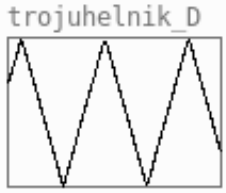
Na následujícím příkladu si ukážeme, jak předchozí operace skloubit a využít je ve tvarování vlny - bude jí vlna ve tvaru trojúhelníku. Vychází částečně z tvaru "pily", ale poté co dosáhne svého vrcholu, nepadá prudce k nule, ale pozvolna klesá. Ve výsledku vypadá tak, jak ukazuje pole "trojuhelnik_D".

Začneme [phasor~]em a posuneme jeho signál o -0.5 dolů (viz pole trojuhelnik_B). Jednu část signálu, která v amplitudě stoupá postupně od od 0 do 0.5, již máme. Teď si musíme nějak vytvořit i tu klesající: když ze signálu izolujeme "spodní část" ([clip~ -0.5 0]) a pak ji invertujeme, dostaneme klesající část vlny. Musíme ale ještě zdvojnásobit její amplitudu (násobíme signál hodnotou -1 a pak 2, tedy -2). Pakliže tuto část přičteme jen k posunuté části signálu, obdržíme trojúhelníkový průběh vlny. Jen ji ještě jednou musíme posunout dolů a škálovat.





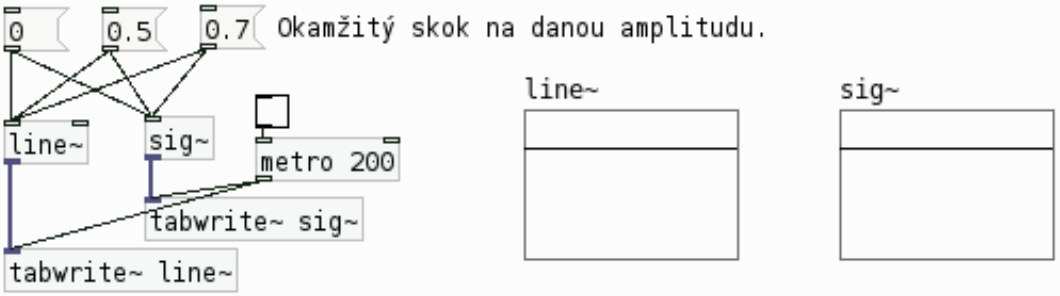




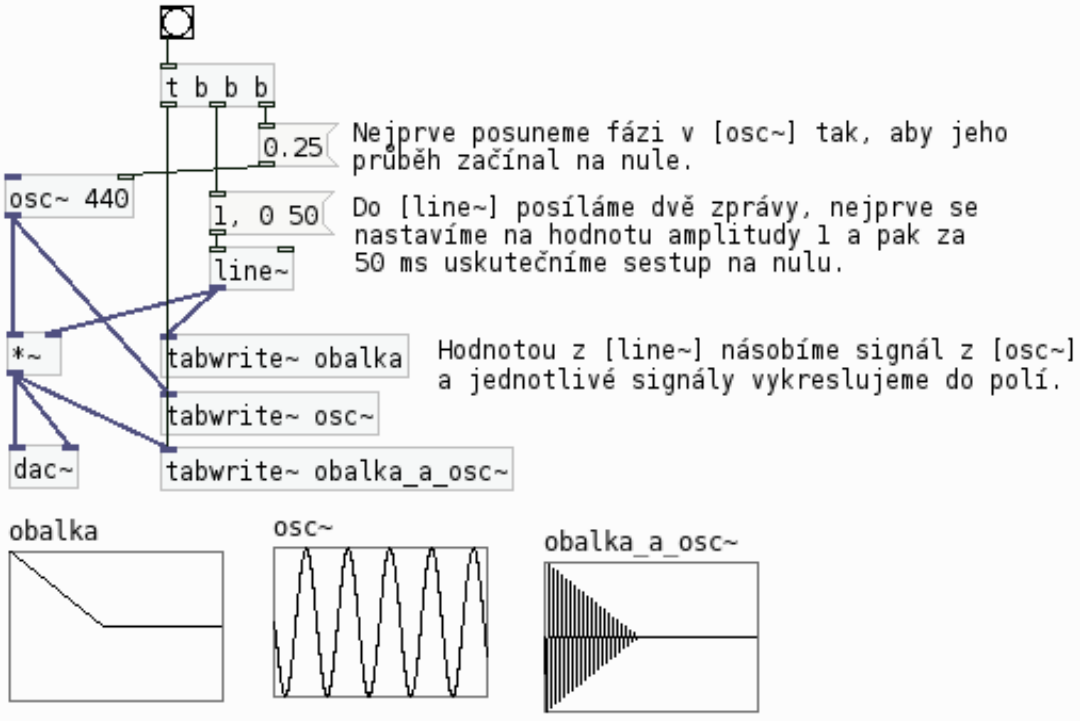
obálka

Doposud generované signály se od zvuků, které vytvářejí tradiční hudební nástroje, lišily v jednom podstatném aspektu a tím je průběh jejich hlasitosti v čase. Tón zahraný na klavír má v intenzitě nějaký náběh, trvá a odeznívá. V této kapitole se podrobněji budeme zabývat technikami, jak v Pd vytvořit tzv. obálku, s jejíž pomocí budeme ovlivňovat amplitudu signálu.

Dobře nám v tom poslouží objekt [line~]. Když do něj zprávou pošleme jednu hodnotu, pak jeho výstupem bude konstantní signál o dané amplitudě. Podobně bychom to provedli s objektem [sig~].

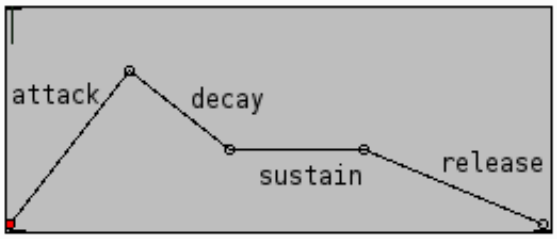


Když do [line~] pošleme zprávu se dvěma hodnotami, přičemž první číslo označuje hodnotu amplitudy a druhé čas v milisekundách, pak výstupem bude lineární náběh, nebo sestup, který bude trvat přesně tuto dobu. Jak výsledný signál vypadá, bude zjevnější z následující ukázky:



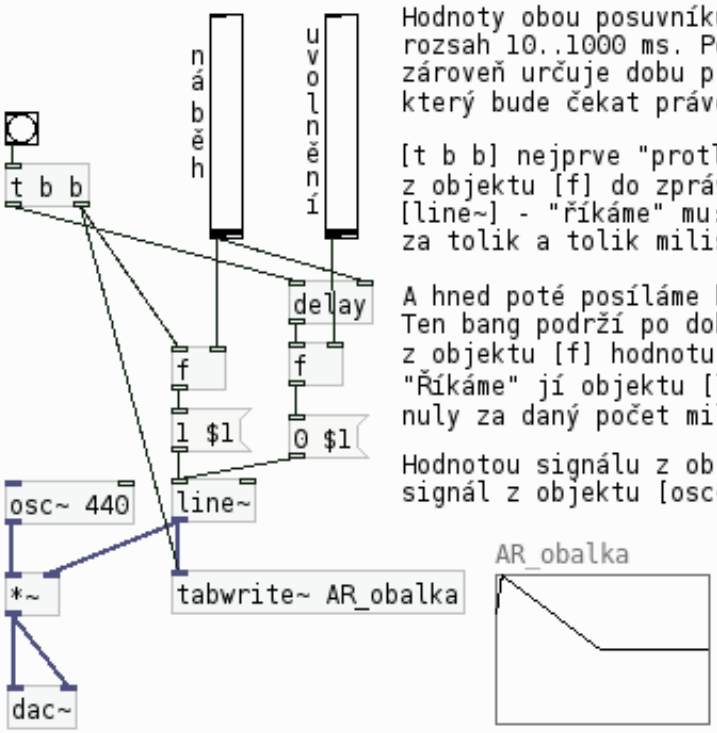
Trochu s uvedeným patchem zaexperimentujte: změňte zprávu pro [line~] na [1, 0 200(, takže dozvuk bude delší, nebo ji naopak zkraťte na 10 ms. Jste při tak krátkém dozvuku schopni rozlišit frekvenci signálu? Zvyšte hodnotu frekvence v [osc~] na 4400 a nechte 10 ms na dozvuk. Je nyní výška frekvence postřehnutelnější? Vyzkoušejte různá nastavení.

V DSP a u syntetizérů se obvykle setkáte s tzv. ADSR obálkou: je v ní popsán náběh (attack), útlum (decay), podržení (sustain) a uvolnění (release). Tyto čtyři parametry stačí k jednoduchému popisu dynamiky tónu.



ATTACK-RELEASE_OBÁLKA

Nyní si postavíme obálku, ve které budeme moci specifikovat délku náběhu a uvolnění - půjde tedy o jednoduchou AR obálku. Vytvoříme ji tak, abychom mohli hodnoty dynamicky měnit posuvníkem.



Hodnoty obou posuvníků jsme nastavili na rozsah 10..1000 ms. Posuvník pro náběh zároveň určuje dobu prodlevy objektu [delay], který bude čekat právě tak dlouho, jak trvá náběh.

[t b b] nejprve "protlačí" hodnotu z objektu [f] do zprávy [l $1(pro objekt [line~] - "říkáme" mu: napočítej do jedné za tolik a tolik milisekund.

A hned poté posíláme bang do objektu [delay]. Ten bang podrží po dobu náběhu a pak "protlačí" z objektu [f] hodnotu do zprávy [0 $1(. "Říkáme" jí objektu [line~], aby napočítal do nuly za daný počet milisekund.

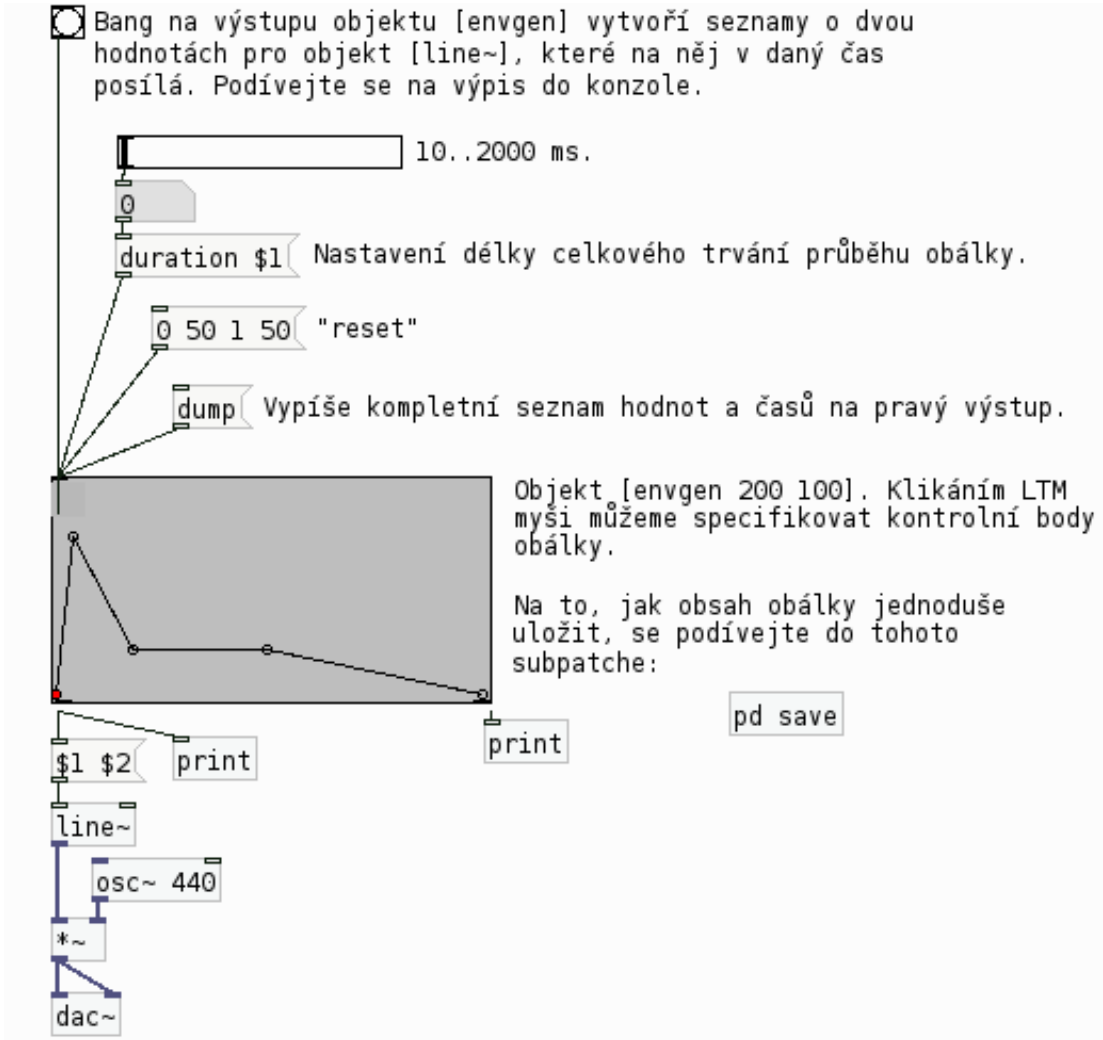
Hodnotou signálu z objektu [line~] opět násobíme signál z objektu [osc~].

Uvedená obálka je velmi jednoduchá, ale i pouze se dvěma parametry dokážeme popsat poměrně širokou škálu gest. Zkuste s hodnotami v posuvnících opět experimentovat, inicializací bangu si výsledek vždy poslechněte a zároveň sledujte, jak se charakter obálky mění v poli "AR_obalka".

SLOŽITĚJŠÍ_OBÁLKY_S_[ENVGEN]

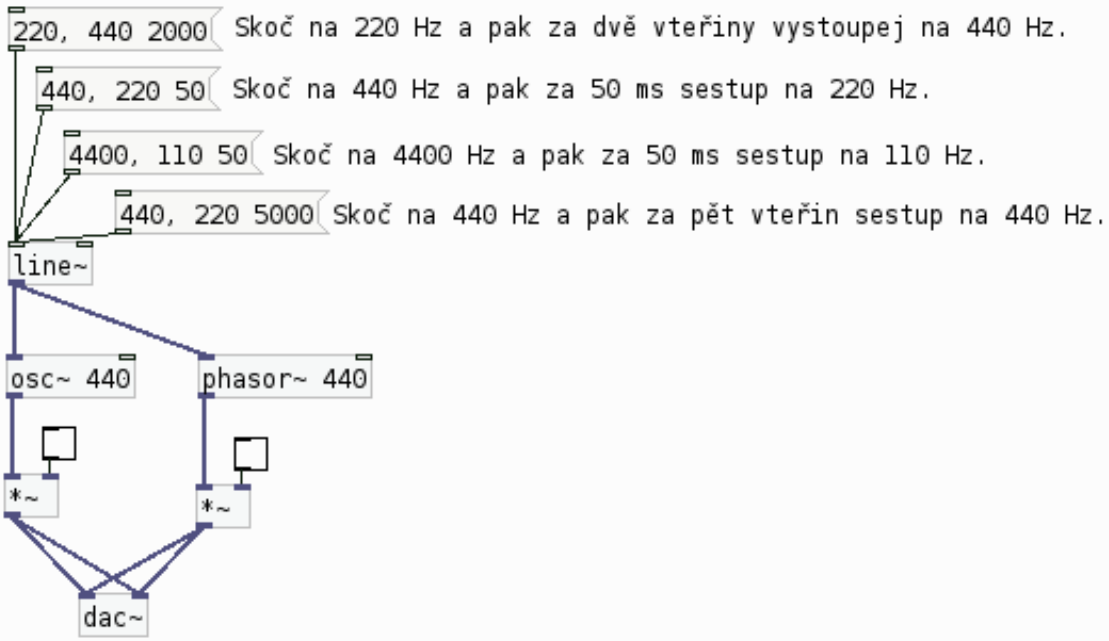
Mohli bychom se pokusit i o konstrukci složitější obálky, místo toho ale využijeme objekt jménem [envgen], který vytvoří grafické uživatelské rozhraní, v němž lze průběh obálky pohodlně naklikat myší.

Grafický rozměr v případě tohoto objektu nenastavujeme v jeho vlastnostech, ale zadáváme ho bezprostředně při jeho vytváření. Takže když napíšeme [envgen 200 100], bude mít rozměry 200x100 pixelů.



[LINE~]_A_OVLÁDÁNÍ_FREKVENCE

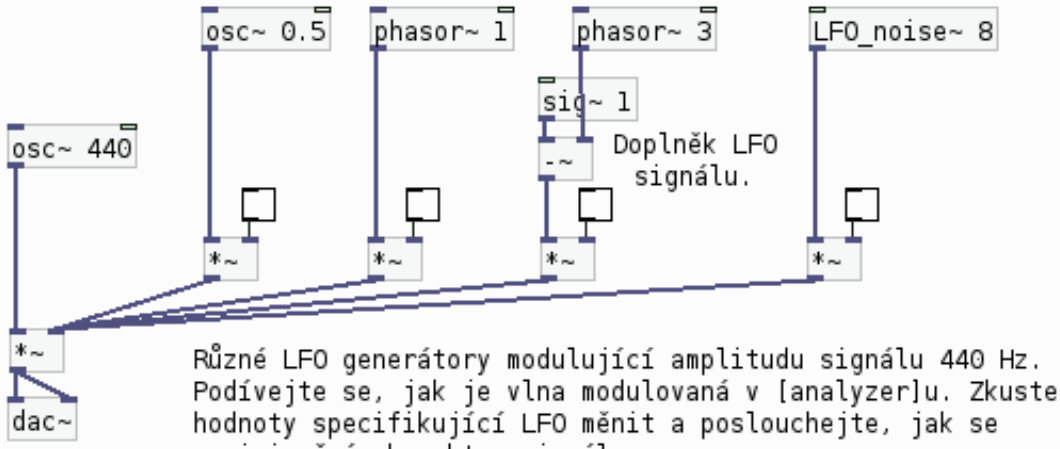
Objekt [line~] nemusí sloužit pouze k ovládání amplitudy, ale můžeme jím řídit i frekvenci v některých generátorech signálu. Lze tak vytvářet lineárně stoupající nebo klesající glisanda.



MODULACE_AMPLITUDY_S_LFO

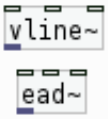
LFO je anglická zkratka, která znamená Low-frequency oscillation. Generátory s tímto označením vytvářejí signál oscilující obvykle v rozsahu do 20 Hz. Jde tedy o neslyšitelné frekvence, jež jsou ale dobře využitelné pro modulaci amplitudy nebo řízení jiných DSP prvků jako filtry a efekty.

LFO generátor vytvoříme prostě tak, že objektům, které již známe, předáme jako argument nízkou frekvenci. Objekt LFO_noise~ je součástí knihovny iemlib a generuje nízkofrekvenční šum.



Různé LFO generátory modulující amplitudu signálu 440 Hz. Podívejte se, jak je vlna modulovaná v [analyzer]u. Zkuste hodnoty specifikující LFO měnit a poslouchejte, jak se s nimi mění charakter signálu.

CVIČENÍ:



- prozkoumejte Nápovědu k objektu [vline~] a [ead~]; čím se liší od objektu [line~]?

- prozkoumejte Nápovědu k [fade~], který je součástí knihovny iemlib; v čem se liší jeho charakteristika od objeků [line~] a [vline~]?

- a do třetice: prozkoumejte Nápovědu k signálové variantě objektu [expr] - tedy [expr~]; pokuste se v něm zapsat polynom $-18x^3 + 23x^2 - 5x$; jaký má vlna tvar?

- zkuste pomocí jednoduchých operací (škálování, inverze, ořez, doplněk) sestrojit vlastní vlny nebo pro ten účel použijte právě objekt [expr~]

- vytvořte si z uvedených příkladů obálek abstrakce

- pokuste se sestrojit jednoduchý monofonní (tzn. jednohlasý) syntetizér s obálkou; výšku frekvence zadávejte z klávesnice - využijte objekty [key] a [select]; zmáčknutou klávesou ovládejte zároveň obálku

- a konečně: pokuste se váš syntetizér propojit s šestnáctidobým krokovým sekvencerem, jehož konstrukcí jsme se zabývali v první části rukověti

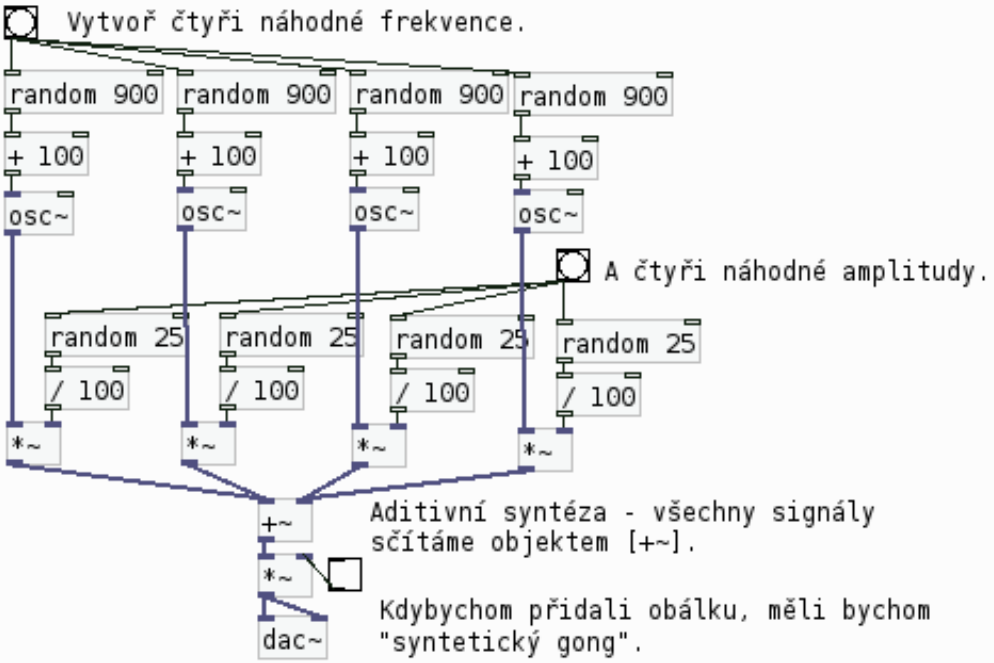
Známe zatím jen pár základních kamenů z celé široké oblasti DSP problematiky - a to ještě v poměrně zjednodušujícím podání. I tak málo ale již stačí k tomu, abyste podnikli svá vlastní sonická dobrodružství a pokusili se o konstrukci prvních jednoduchých nástrojů.

syntetizujeme

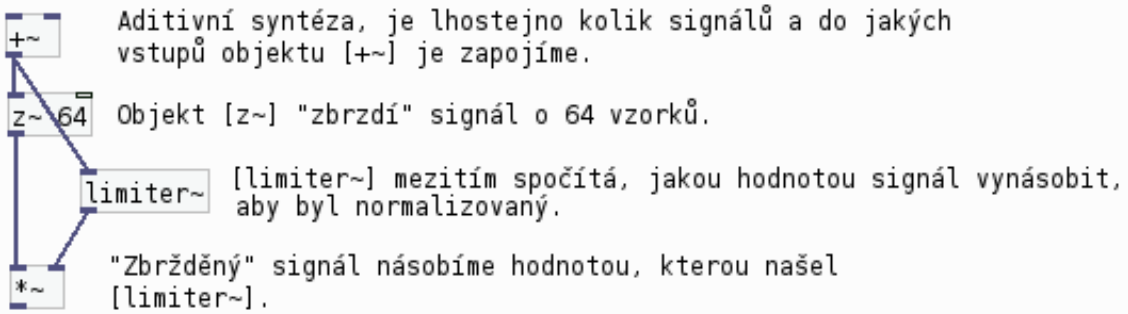
Řecké slovo synthesis, od kterého se odvozuje slovo syntéza, označuje proces slučování - skládání různých částí do jednoho celku. V DSP se syntézou míní různé techniky vytváření zvuku. V této kapitole si v základech představíme aditivní, subtraktivní, tabulkovou (wavetable) a frekvenčně modulovanou syntézu.

ADITIVNÍ_SYNTÉZA

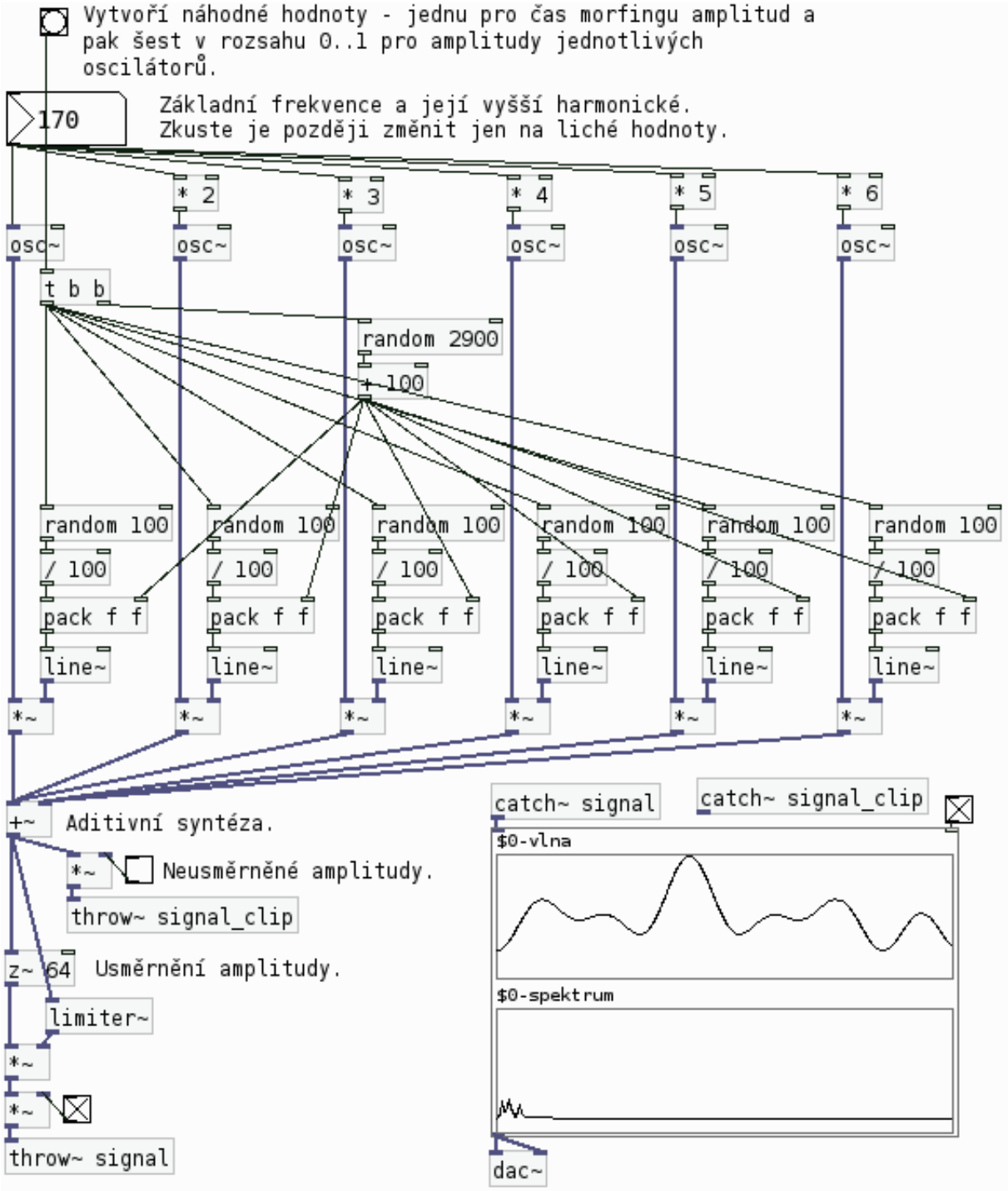
Aditivní syntéza je v Pd realizovatelná prostým sčítáním různých signálů s pomocí objektu [+~]. Když sečteme několik signálů z objektu [osc~] o různých frekvencích a amplitudách, je výsledná vlna komplexnější a barva zvuku plnější. Vzpomeňte si na kapitolu o interních zprávách - v ní jsme vlastně sestrojili masivní aditivní syntetizér.



Když budete sčítat více různých signálů o různých amplitudách, snadno se přihodí, že amplituda "vyskočí" nad limit -1..1 a dojde ke zkreslení signálu. Předcházet tomuto nežádoucímu jevu můžeme objekty [limiter~] a [z~], které jsou součástí knihovny zexy. Segment patche usměrňující součet amplitud do patřičného limitu vypadá takto:



Aditivní syntézou se nám otevírají možnosti sonických průzkumů. Na následujícím jednoduchém příkladu si ukážeme, jak např. provést morfing jednoho témbru do jiného.



Doba morfingu amplitud je náhodná, ale pro všechny oscilátory stejná - je v rozsahu 100..3000 ms. Velikost amplitudy je pro každý oscilátor jedinečná. Obě hodnoty zabalíme do seznamu objektem [pack] a posíláme je rovnou do objektu [line~]. Nemusíme používat zprávu [$1 $2(. Signál usměrňujeme [limiter~]em. Prozkoumejte v [analyzer]u, co se s vlnou během morfingu děje a jak vypadá signál před a po usměrnění. Kdybychom patch ještě doplnili o obálku a schovali do abstrakce, máme hezký monofonní aditivní syntetizér - pokuste se o to.

Aditivní syntéza hrála důležitou roli na úsvitu elektronické hudby. Zkuste si např. poslechnout kompozici Studie II od Karlheinze Stockhausena a dohledejte si k ní informace.

SUBTRAKTIVNÍ_SYNTÉZA

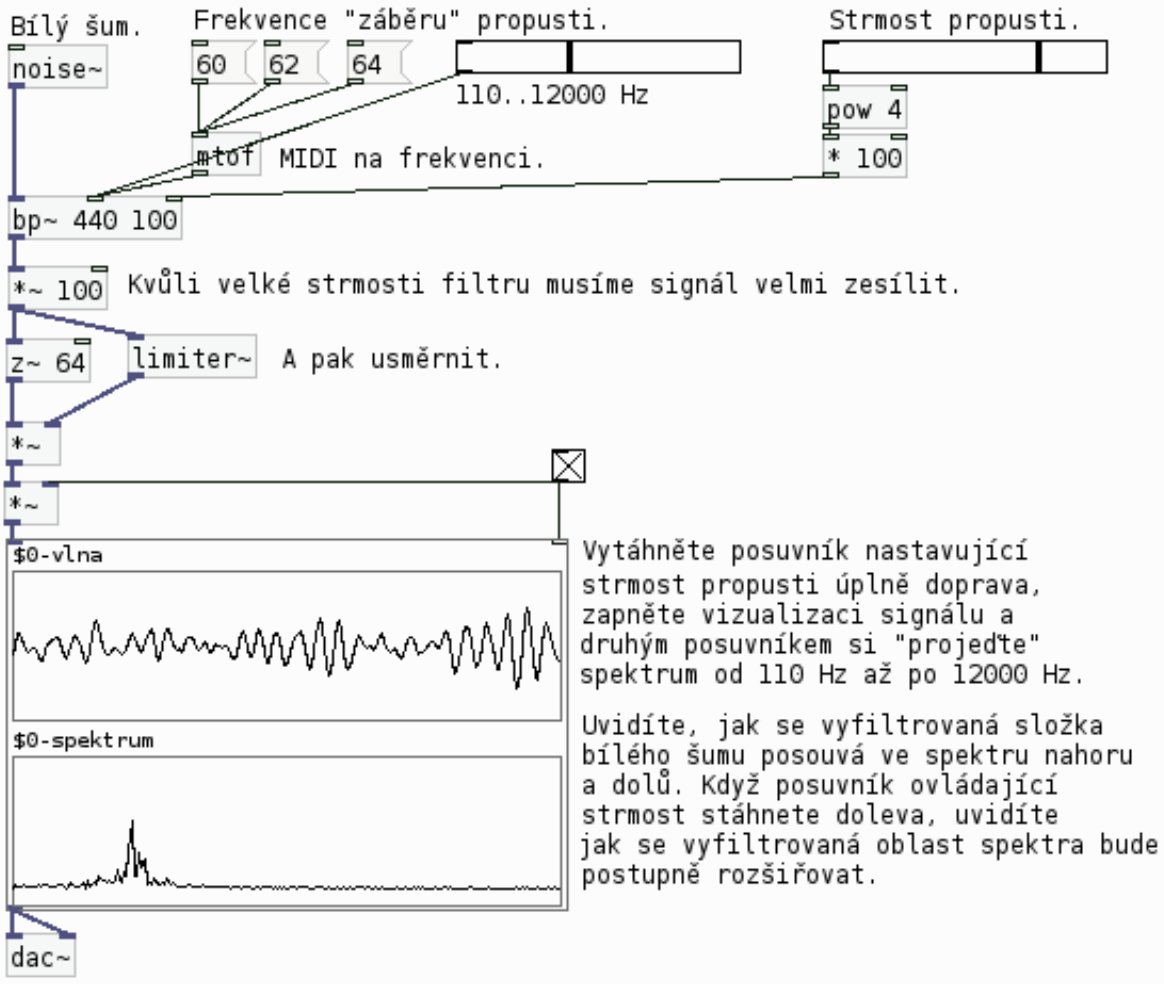
V aditivní syntéze signály slučujeme a tím vytváříme nové kvality ve spektru. U subtraktivní syntézy je proces opačný. Výchozím "materiálem" nám bude spektrálně bohatý signál a my z něj budeme odečítat složky. S pomocí objektu [bp~], což je pásmová propust (band-pass filter), můžeme z bílého šumu vyjmout určité frekvence.

[bp~] má tři vstupy. První je pro signál, druhým specifikujeme frekvenci, kolem které filtr "zabírá". Pásmová propust - jak napovídá název - propouští pouze určité pásmo v okolí dané frekvence. V Pd máme k dispozici i horní a spodní propust (filtry [hip~] a [lop~]), s [bp~] ale dané pásmo odfiltrujeme snáz. Třetím vstupem u [bp~] specifikujeme strmost filtru - tj. jaký rozsah z "okolí" frekvence záběru ještě propustí. Když je strmost filtru malá, je tento rozsah větší a vyfiltrovaný signál je více zašumělý a vice versa.

bp~

hip~

lop~



Vytáhněte posuvník nastavující strmost propusti úplně doprava, zapněte vizualizaci signálu a druhým posuvníkem si "projeďte" spektrum od 110 Hz až po 12000 Hz.

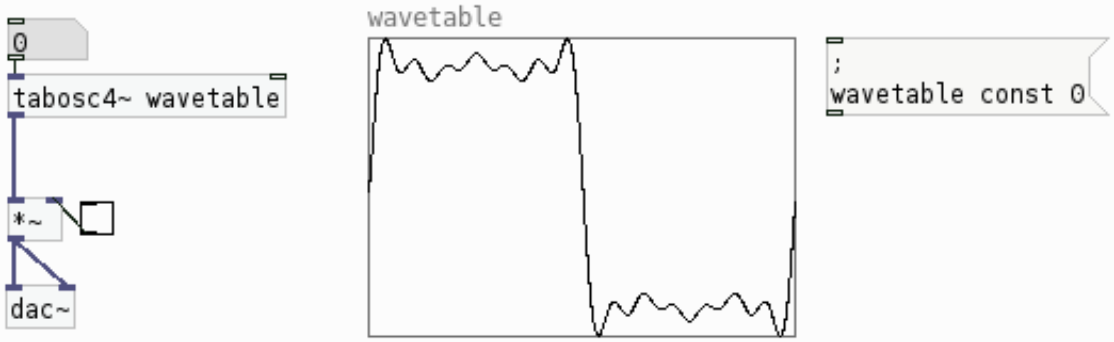
Uvidíte, jak se vyfiltrovaná složka bílého šumu posouvá ve spektru nahoru a dolů. Když posuvník ovládající strmost stáhnete doleva, uvidíte jak se vyfiltrovaná oblast spektra bude postupně rozšiřovat.

Zkuste experimentovat s mírou strmosti filtru, ev. prozkoumejte, jak se v kontaktu s bílým šumem chovají objekty [hip~] a [lop~]. Subtraktivní syntéza je vhodná k vytváření nástrojů, které mají "vzdušný" charakter - připomíná různé flétny a píšťaly.

V syntetizérech se často používají různé typy a kombinace syntéz - již teď se můžete pustit do vlastních průzkumů a zkusit zkombinovat aditivní a subtraktivní syntézu v různých poměrech a s různými obálkami.

TABULKOVÁ_SYNTÉZA

Tabulková (wavetable) syntéza je založena na možnosti uložit krátký vzorek, se kterým se pak cyklicky operuje. V Pd nám k uložení vzorku poslouží pole, do kterého si vlastní průběh jedné periody dané vlny můžeme nakreslit myší. Pro periodické přehrávání vlny v poli slouží objekt [tabosc4~], kterému se jako parametr předává název pole se vzorkem. Levý vstup je pro číslo nebo signál udávající frekvenci a pravý je pro nastavení fáze.



Když se na signál podíváte abstrakcí [analyzer], zjistíte, že tvar vlny odpovídá nakreslenému tvaru v poli. Pole musí mít délku, která odpovídá $2^x + 3$ (tedy např. 64+3=67, 128+3=131, 256+3=259). Pokud tato podmínka není splněna, [tabosc4~] nebude fungovat a v Pd konzoli uvidíme chybovou hlášku.

Kromě volného kreslení myší lze pole se vzorkem vyplňovat také řízeněji. Vzpomeňte si na příklad, kdy jsme si ukazovali, jak součtem více sinusovek vytvořit vlnu, která se podobá tvaru "pily". Sčítali jsme postupně všechny harmonické frekvence a zmenšovali jejich amplitudu podle toho, o kterou harmonickou frekvenci šlo (1/h. frekvence). Podobně jako s objekty [osc~] to můžeme provést pomocí zprávy sinesum, kterou posíláme do daného pole.

```
;
wavetable sinesum 131 1
```
Pošle do pole s názvem "wavetable", jehož délku nastaví na 131 prvků, součet amplitud sinusovek jednotlivých harmonických frekvencí.

```
;
wavetable sinesum 131 0.5
```
První harmonická o poloviční amplitudě.

```
;
wavetable sinesum 2051 1 0.5 0.3 0.25, normalize
```
Součet čtyř harmonických s klesající amplitudou a normalizovaných.

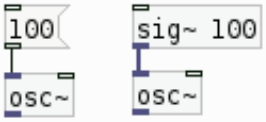
```
;
wavetable sinesum 2051 1 0.5 0.3333 0.25 0.2 0.1667 0.1429
0.125 0.1111 0.1 0.0909 0.0833 0.0769, normalize
```
Součet všech harmonických s amplitudou 1/h vytvoří "pilu".

```
;
wavetable sinesum 2051 1 0 0.3333 0 0.25 0 0.1429 0 0.1111
0 0.0909 0 0.0769, normalize
```
Součet pouze lichých harmonických vytvoří "čtverec".

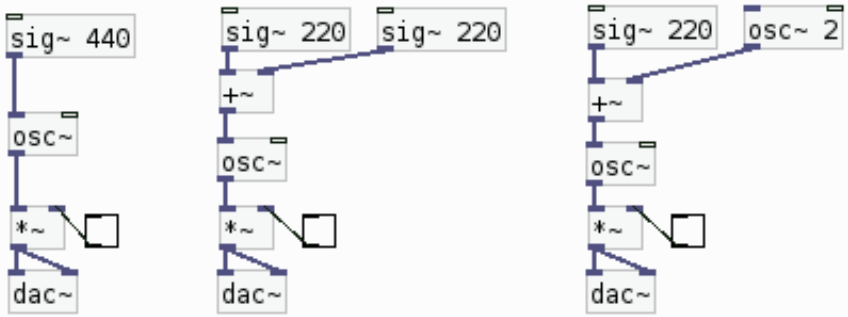
FM_SYNTÉZA

Vynález frekvenčně modulované syntézy zvuku Johnem Chowningem, jenž byla patentována v roce 1975, spolu s její implementací v syntetizéru Yamaha DX7, znamenal určitý průlom na poli elektronické hudby. Tato technika syntézy totiž umožnila vytvářet nové zvukové barvy a řada autorů začala její dispozice ve velkém využívat.

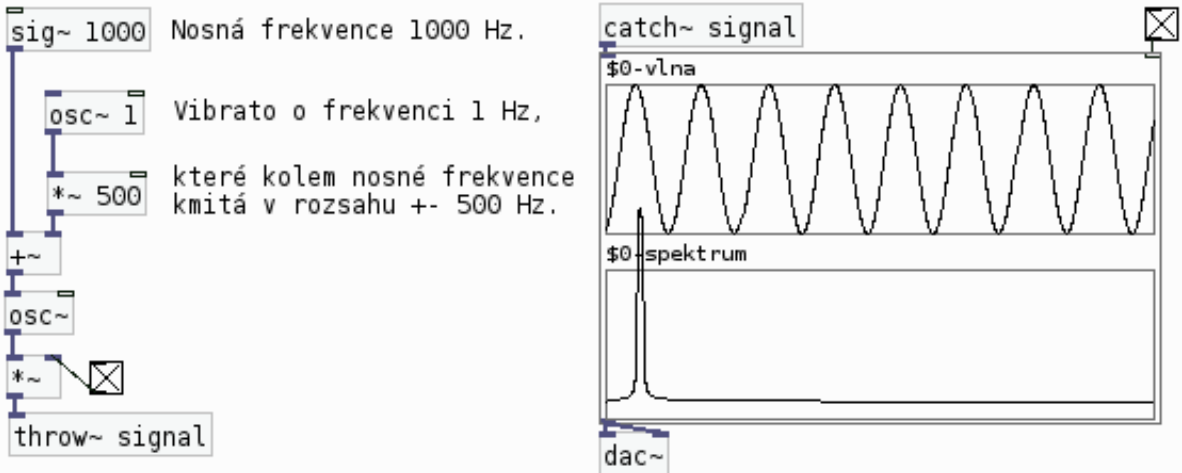
Ukážeme si v několika stupních, jak jádro FM syntetizéru v Pd postavit. Do levého vstupu objektu [osc~] lze poslat buď číslo nebo signál - obojí určuje jeho frekvenci, budeme ji nazývat nosnou frekvencí. Když mezi objekt [sig~] a [osc~] vložíme objekt [+~], můžeme po cestě ještě k signálu přičítat. Když sečteme výstupy ze dvou [sig~ 220] a výsledný signál pošleme do [osc~], uslyšíme frekvenci 440 Hz. Co kdybychom ale místo konstantního [sig~ 100] použili nějaký periodicky oscilující generátor?



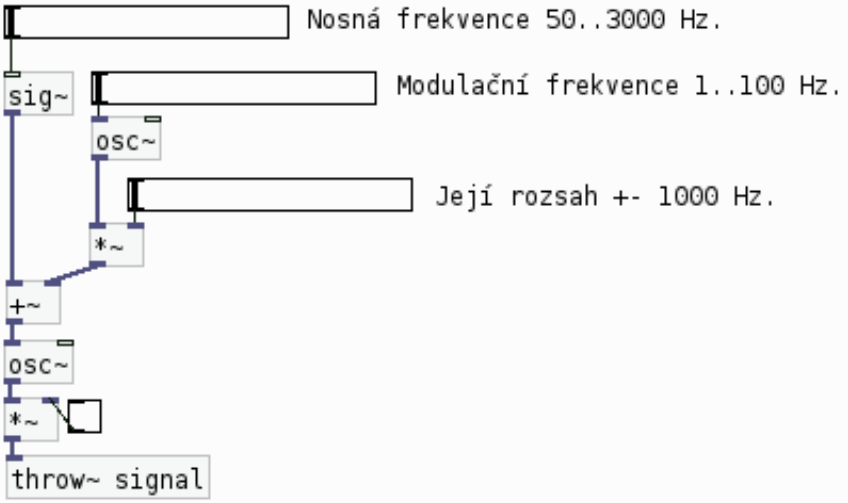
Záměnou [sig~ 220] za [osc~ 2] vytvoříme již jednoduchý FM syntetizér. Co se se signálem děje? Ke konstantní hodnotě 220 permanentně přičítáme hodnoty z [osc~], které jsou v rozsahu -1..1. Ve zvuku při 220 Hz skoro žádnou změnu neuslyšíme, ale pokud hodnotu v [sig~ 220] snížíme na 80, pak bychom měli slyšet velmi jemné vibrato mezi 79 a 81 Hz. Ve vibratu jde o modulaci frekvence - v našem případě má vibrato frekvenci 2 Hz a kolem nosné frekvence osciluje nahoru a dolů v rozsahu -1..1, což je dáno amplitudou [osc~].



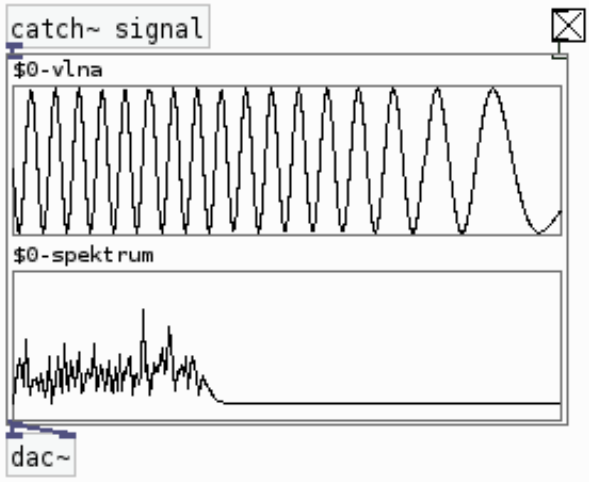
Nyní jde o to, jak amplitudu z modulačního oscilátoru rozšířit tak, aby byl záběr vibrata větší. Poslouží nám k tomu dobře objekt [*~], kterým velikost amplitudy modulačního oscilátoru zvětšíme. Z ilustrativních důvodů nyní hodnoty pozměníme tak, aby bylo v abstrakci [analyzer] dobře viditelné, co se se signálem při FM syntéze děje.

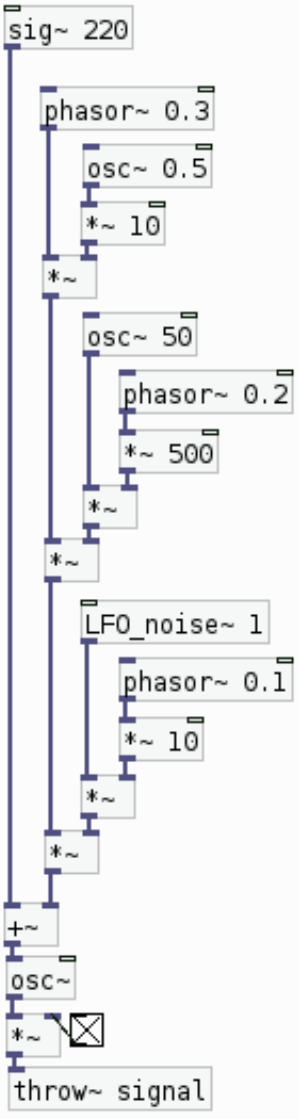


Poslechněte si poslední příklad a podívejte se na něj v abstrakci [analyzer]. Ve spektru uvidíte, jak se nosná frekvence pohybuje v rozsahu 500..1500 Hz. Jednotlivé parametry můžeme samozřejmě dynamicky ovládat. Vyzkoušejte různá nastavení, poslouchejte a pozorujte, jak se charakter signálu mění.



Větev patche, která se stará o frekvenční modulaci, může být samozřejmě mnohem složitější a lze v ní kombinovat různé typy generátorů signálu. Zkuste k některým objektům z vedlejšího příkladu připojit ovládací prvky a proměnit ho tak na experimentální FM nástroj. Každou z částí patche prozkoumejte v [analyzer]u zvlášť, pak vám snad bude jasnější, co která dělá.





Jsme na konci kapitoly, která se týkala základních typů syntéz. I když byl náš výklad zjednodušující a nešel až k základům problematiky DSP, poskytl snad dostatečnou škálu možností pro započetí vlastních experimentů a konstrukci zvukových syntetizérů. Určité oblasti z průzkumu zvuku budeme s to aplikovat dobře i v části zabývající se obrazem - např. obálku.

Zkuste tedy nějaký čas věnovat experimentování s různými typy syntéz. Díky jejich kombinaci, tvarování vln a užívání odlišných typů generátorů signálu se vám otevírá poměrně široké pole pro sonická dobrodružství.

vzorky_neboli_samply

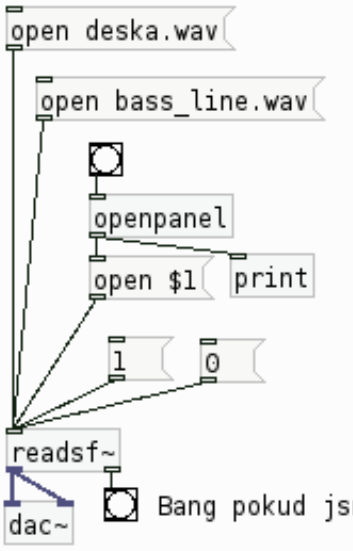
Jestliže by předchozí látku a praxi, kterým jsme se v Pd věnovali, bylo možné přirovnat k práci na vodním díle, a jednoduché aritmetické operace se signálem k usměrňování proudu, touto kapitolou, věnující se vzorkům, otvíráme téma "jak zacházet s kusy ledu".

Víme již, že na digitální platformě je zvuk reprezentován vzorky, kterých je standardně v jedné vteřině 44100, a pokud je jeden vzorek reprezentován 32bitovým číslem, může nést hodnotu v rozsahu vyjádřitelné číslem 2^{32}.

Vzorky zvuku jsou uloženy v souborech, k tomuto účelu určených. Obvykle se setkáme s formáty WAV (Waveform audio file format) nebo AIFF (Audio Interchange File Format), které používají tzv. bezztrátovou kompresi dat. Soubory s koncovkou MP3, OGG obsahují zvuková data, jež jsou komprimována ztrátovou kompresí dat. Pd zprostředkovávají objekty, které umí pracovat s různými formáty. Vzhledem k výpočetní nenáročnosti a kvalitě signálu je ale doporučitelné používat formáty s bezztrátovou kompresí.

PŘEHRÁVÁME_SOUBORY_Z_DISKU

K jednoduchému přehrávání souborů z disku slouží objekt [readsf~] (read sound file), který umí načítat soubory typu WAV. Ve zprávě mu předáváme příkaz open a cestu k souboru. Zprávy [1(a [0(spustí, nebo zastaví jeho přehrávání. Levý výstup je audiosignál a na pravý výstup objekt [readsf~] posílá bang, pokud byl soubor přehrán do konce. Je možné ho tedy využít k reinicializaci přehrávání ve smyčce.

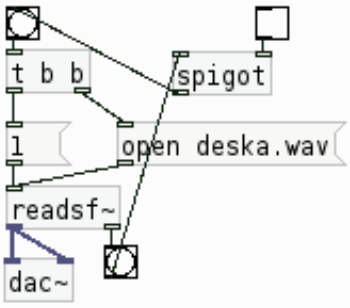


Pokud máme soubory se zvukem přímo v adresáři, kde je i patch, jenž je má načítat, stačí předat jméno souboru. Jinak musíme zadat absolutní cestu k souboru.

Objekt [openpanel] otevře okno, v němž můžeme procházet adresáře a vybrat daný zvukový soubor. Po kliknutí na tlačítko "Otevřít" pak [openpanel] předá absolutní cestu k souboru jako symbol zprávě [open $1(.

Po načtení souboru spustíme / zastavíme jeho přehrávání.

Bang pokud jsme soubor přehráli do konce.



"Problém" objektu [readsf~] spočívá v tom, že před spuštěním přehrávání je vždy třeba soubor z disku počítače načíst. Zde to řešíme objektem [t b b], který nejprve načte soubor, a pak teprve spustí přehrávání. Přepínač zapojený do objektu [spigot] pak zapíná/vypíná přehrávání ve smyčce.

Dalším "problémem" je, že [readsf~] neumí zvukový vzorek přetáčet dopředu nebo dozadu.

NAČÍTÁME_VZORKY_DO_POLE

Pro velmi základní práci se zvukovými vzorky postačí objekt [readsf~]. Z předchozího příkladu ale snad bylo zřejmé, že kromě spuštění a zastavení přehrávání nám nic víc nenabízí. Pro práci se zvukem v reálném čase také není vhodné načítat data stále z disku, ale pracovat s nimi v paměti, protože k paměti má počítač rychlejší přístup.

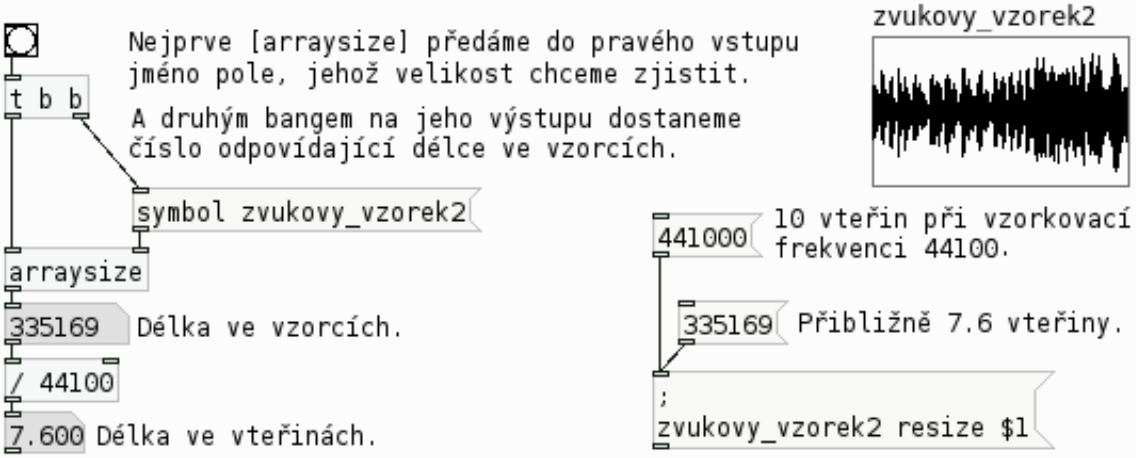
K načítání zvukových vzorků do pole slouží objekt [soundfiler]. Zprávou mu předáváme příkaz read, následuje jméno zvukového souboru a poslední parametr je název pole, do něhož chceme zvukový vzorek načíst. Tedy např:

```
read deska.wav zvukovy_vzorek
```

Před názvem souboru ale ještě může předcházet zpráva "-resize", která zajistí to, že se velikost pole nastaví přesně podle celkového počtu vzorků ve zvukovém souboru. Jak velký soubor Pd dokážou načíst se odvíjí od hardwarových dispozic počítače. Na tom mém se mi do pole nepodařilo nahrát soubor delší než 15 minut. Pokud jsou dispozice počítače překonány, upozorní vás na to v konzoli. Mně se objevila hláška: "soundfiler_read: truncated to 4e+07 elements". 20ti minutových záznam ve formátu WAV, který jsem se [soundfiler]em pokusil do pole načíst, byl redukován na 40 miliónů vzorků, což odpovídá při vzorkovací frekvenci 44100 přibližně 15ti minutám.

Pokud byste chtěli do pole přesto načíst delší soubor, pak musíte před název soubouru připsat ještě "-maxsize" a délku uvedenou ve vzorcích. V případě 20minutového souboru to tedy dělá 20 * 60 * 44100 (minuty * vteřiny * počet vzorků ve vteřině), což vychází na 5.292e+07 vzorků. V tom případě ale hrozí, že při větším množství takto velkých souborů se Pd zhroutí. S polem, jež obsahuje větší množství vzorků, se v editačním módu také velmi obtížně manipuluje. Proto se pole v případě velkých souborů schovávají do subpatchů, aby Pd nebyla zaměstnána jejich vykreslováním.

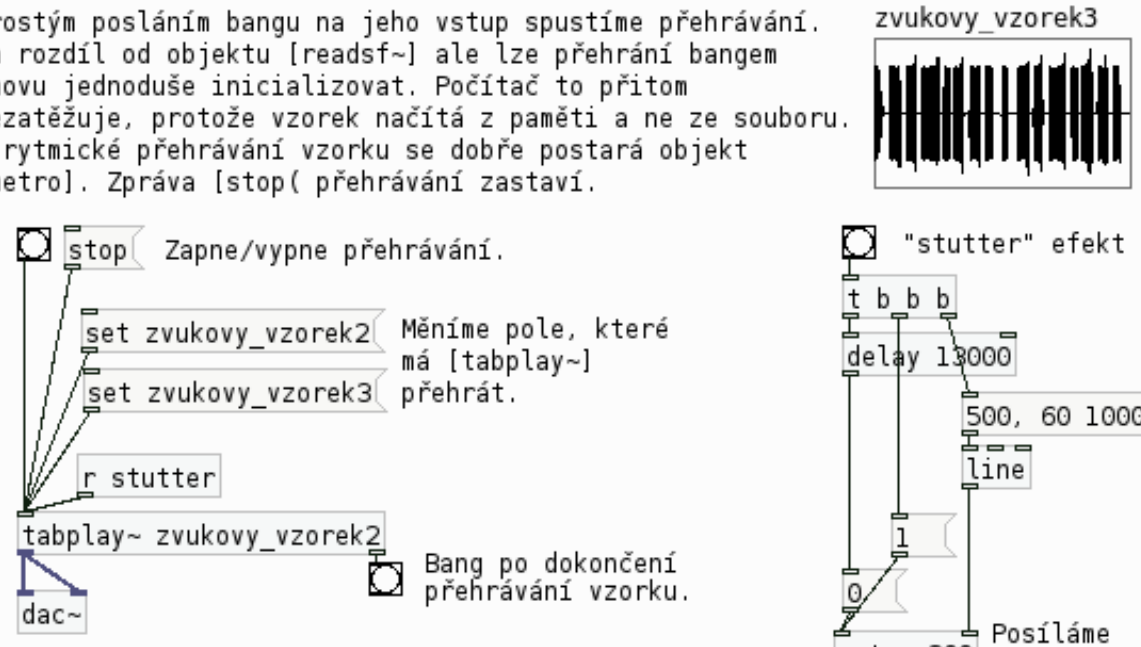
Pakliže jsme soubor načetli do pole, je umístěn v paměti počítače a máme k němu rychlejší přístup. K práci se vzorky se nám občas bude hodit objekt [arraysize], který umí zjistit délku pole. A dále interní zpráva, jíž můžeme délku pole změnit.



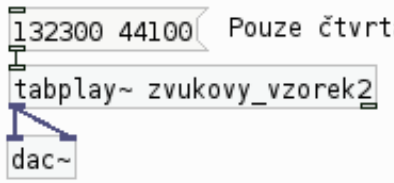
PŘEHRÁVÁME_POLE

Poté, co jsme úspěšně načetli soubor do pole, se můžeme pustit do jeho přehrávání. Využijeme k tomu objekt [tabplay~]. Jako argument mu předáváme jméno pole, které chceme přehrát. Přehrávané pole lze ale také dynamicky měnit zprávou [set jméno pole(.

Prostým posláním bangu na jeho vstup spustíme přehrávání. Na rozdíl od objektu [readsf~] ale lze přehrání bangem znovu jednoduše inicializovat. Počítač to přitom nezatěžuje, protože vzorek načítá z paměti a ne ze souboru. O rytmické přehrávání vzorku se dobře postará objekt [metro]. Zpráva [stop(přehrávání zastaví.



Kromě uvedených zpráv je [tabplay~] ještě ovladatelný zprávou se dvěma čísly: první určuje vzorek, od kterého se má začít přehrávat, a druhé počet vzorků, jež se mají přehrát. Chceme-li tedy přehrát pouze čtvrtou vteřinu z pole, do [tabplay~] pošleme zprávu [132300 44100(.

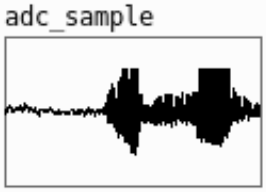


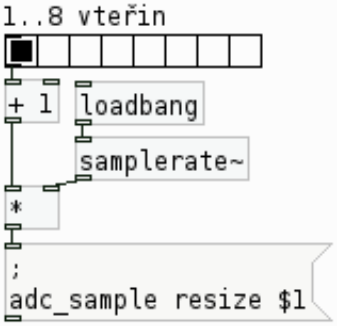
NAHRÁVÁME_ZE_VSTUPU

Vytvořit si v Pd jednoduchý sampler není nic těžkého. Pokud máte k dispozici mikrofon, zapojte ho do zvukové karty a otestujte, zda signál z něj jde do Pd (menu Media -> Test Audio and MIDI). Když vyškrtnete vypínač s názvem "monitor-inputs", měli byste slyšet signál z mikrofonu. Jednoduchý odposlech s regulovatelnou vstupní hlasitostí ostatně není složité naprogramovat.

Odposlech - signál ze vstupu zvukové karty ovládáme "logaritmickým" posuvníkem.

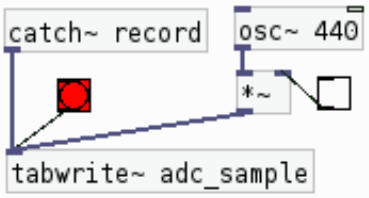
Výsledný signál posíláme na zvukovou sběrnici jménem record.



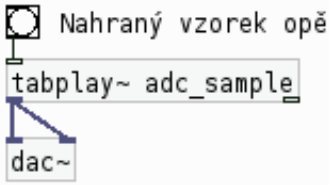


Délka pořízeného záznamu se odvíjí od délky pole. Vedlejším patchem ji "natvrdo" nastavujeme na délku 1..8 vteřin.

K záznamu do pole použijeme objekt [tabwrite~]. Zapojíme do něj signál, který chceme nahrát, a kliknutím na bang nahrávání spustíme. Ujistěte se, že máte vytažený posuvník ovládající hlasitost vstupu, a že tedy objekt [throw~] reálně vysílá signál pojmenovaný jako "record. Pokud k dispozici mikrofon nemáte, můžete do [tabwrite~] poslat signál z [osc~ 440].



Po kliknutí na červený bang nahrajeme do pole daný počet vteřin.



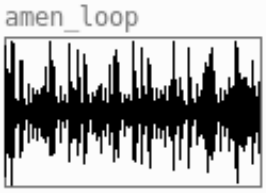
Nahraný vzorek opět přehrajeme objektem [tabplay~].

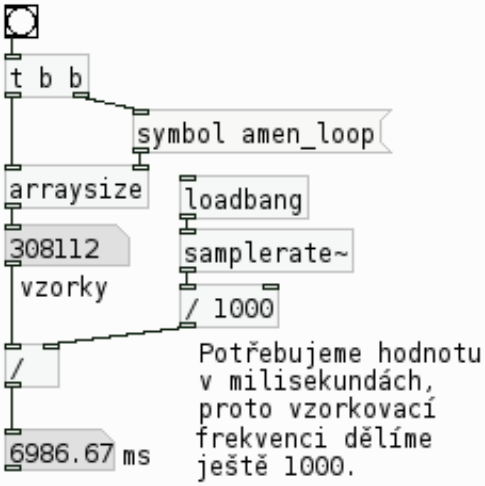
[TABREAD~]

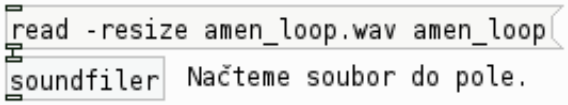
Objekt [tabread~] nám ve srovnání s objektem [tablay~] nabízí další způsoby přehrávání vzorku. Ovládá se na vstupu řídícím signálem, který přehrávání "pohání". Jestliže chceme přehrát celý vzorek od začátku do konce odpovídající rychlostí, pak je třeba poslat do [tabread~] lineárně vzestupný signál.

V případě přehrání 44100 vzorků za jednu vteřinu využijeme objekt [line~], který nejprve vynulujeme, a pak do něj pošleme zprávu [44100 1000(. Tím vytvoříme lineárně vzestupný signál, jenž za 1000 milisekund "vystoupá" na hodnotu 44100.

Chcete-li, využijte k testování vlastností objektu [tabread4~] vlastní vysamplovaný vzorek. V našem případě ale k demonstraci využijeme asi 7 vteřin z písně "Amen, Brothers" (1960) od skupiny The Winstons. Nejprve vzorek načteme do pole se jménem "amen_loop". [arraysize] nám pomůže zjistit délku pole ve vzorcích a v milisekundách.

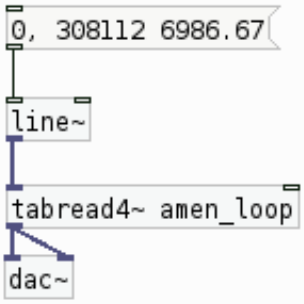






Vzhledem k tomu, že jsme načetli soubor amen_loop.wav s parametrem "-resize", změnila se délka pole dynamicky podle délky souboru. Mohli bychom údaje o počtu vzorků přebírat i z výstupu objektu [soundfiler], ale [arraysize] je výhodnější, protože umožňuje zjišťovat délku různých polí pouhou změnou názvu, který posíláme do pravého vstupu.

Vidíme, že pole "amen_loop" obsahuje 308112 vzorků, což při vzorkovací frekvenci 44100 odpovídá délce 6986.67 ms. Délka ve vzorcích a milisekundách je důležitá, protože to jsou právě hodnoty, které budeme předávat objektu [line~].



Vynuluj se a pak vytvoř signál, který lineárně vystoupá na hodnotu 308112 (tedy do konce pole) za 6986.67 ms.

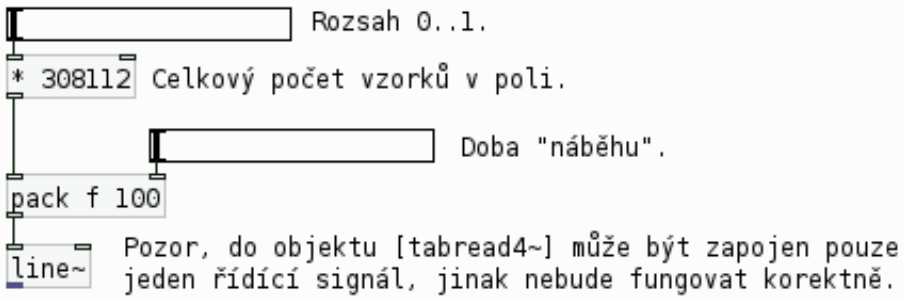
Pole by šlo cyklicky přehrávat objektem metro s argumentem 6986.67, který by pravidelně posílal bang do zprávy.

Pokud bychom nyní do objektu [line~] poslali zprávu [308112, 0 6986.67(, vznikl by lineární signál, jenž by z hodnoty 308112 klesal na nulu - opět za dobu 6986.67 ms a my bychom uslyšeli zvuk přehrávaný pozpátku. Vyzkoušejte to. A což tak změnit hodnotu označující délku přehrávání v milisekundách na dvojnásobek? Ano, přehrávání vzorku bude trvat dvakrát tak dlouho a dojde ke změně jeho frekvence.

Řešením problému zachování frekvence vzorku při změně délky jeho přehrávání (tzv. timestretching) je poněkud náročnější DSP praktika a nebudeme se jí zde zabývat. Případně prozkoumejte přímo v Puckettově dokumentaci (Nápověda -> Pd Help Browser -> Pure Data -> 3.audio examples) soubor I07.phase.vocoder.pd nebo abstrakci [timestretch~], která je součástí Pd-extended.

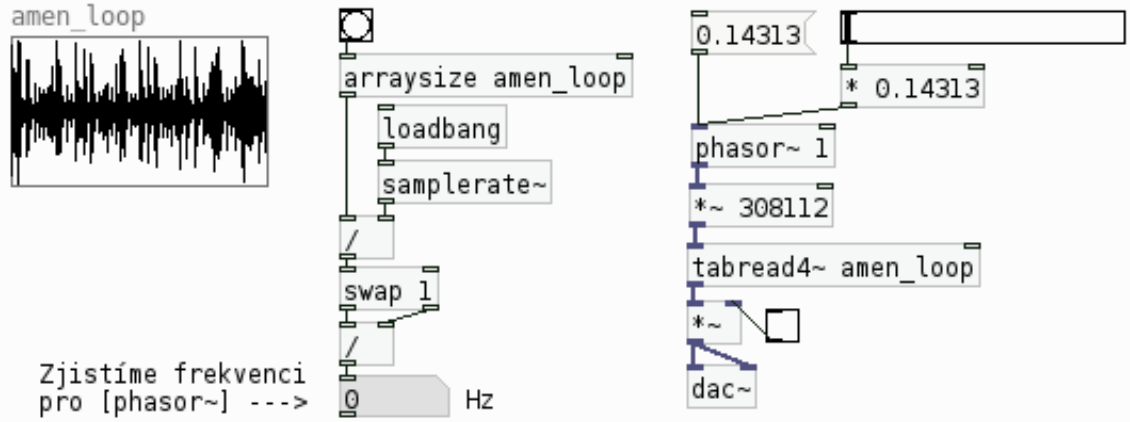
Vraťme se k našemu příkladu. Nyní by vám již mělo být jasné, jak funguje studie "scratcheru", kterou jsme uvedli v první části rukověti. V ní jsme do objektu [line~] posílali hodnou z posuvníku o rozsahu 0..1, kterou jsme

násobili počtem vzorků v samplu, takže jeho rozsah pokrýval celou délku souboru. Druhá hodnota pro objekt [line~] byla v rozsahu 100..1000 ms - šlo tedy o dobu, za jakou má [line~] na danou pozici v souboru "najet". V případě "amen_loop" bychom scratcher získali rozšířením patche o následující segment:



[TABREAD~]_A_[PHASOR~]

Řídící signál pro objekt [tabread~] nemusí zajišťovat jen [line~], ale např. objekt [phasor~], jehož jedna perioda má také lineárně stoupající průběh. Postačí, když jeho výstup vynásobíme celkovým počtem vzorků a nastavíme správně jeho frekvenci - ta odpovídá jednoduchému vzorci 1/délka vzorku ve vteřinách. Objekt [phasor~] je cyklický generátor signálu - bude tedy, na rozdíl od [line~], "pohánět" přehrávání vzorku ve stálé smyčce.



Základy práce se vzorky máme tímto za sebou. Pokud byste chtěli prozkoumávat pokročilejší techniky práce se vzorky, doporučuji k samostudiu analýzu patche ChopSuey 3.2 od id toxonic, který je po registraci stažitelný na Pd fóru. Jde o velmi pokročilý slicer (nástroj, který umí vzorky "rozřezat" a jednotlivě je přehrávat). Navíc obsahuje i funkci automatické detekce rytmu.

ChopSuey 3.2

CVIČENÍ:

- příklady, které jsme uvedli, jsou těžkopádné, protože s hodnotami délky vzorku, jeho trvání v milisekundách atd., pracovaly jako s konstantami a ne jako s proměnnými; pokuste se tuto jejich nedokonalost vylepšit a postavte si abstrakci pro načítání, záznam a přehrávání vzorků, která by byla dynamická; využijte k tomu objekty [send], [receive] a nezapomeňte na souvislost problematiky identifikátoru "$0-" a polí
- pro inspiraci můžete nahlédnout do sbírky patchů DIY2 od hardoffa, kterou jsme již zmiňovali

DIY2

MIDI_kontrolery

Již v kapitole věnující se HID (Human Interface Device) jsme naznačili, že v přístupu k nástrojům a systémům, které jsme v Pd vytvořili, nejsme odkázáni pouze na klávesnici a myš. V této kapitole se blíž podíváme na možnosti, které Pd nabízejí v komunikaci s MIDI kontrolery.

Pojmem MIDI kontroler označujeme celou širokou škálu hardwarových zařízení: standardně jde o MIDI klaviatury doplněné o další ovládací prvky, jako jsou tahové a otočné potenciometry - ty v grafických uživatelských prvcích Pd odpovídají posuvníkům a objektu [knob].

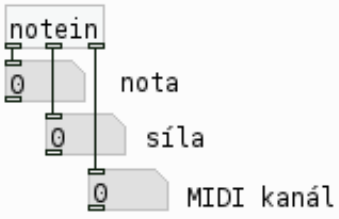
Propojení MIDI kontrolerů s Pd se liší podle toho, v jakém operačním systému pracujeme. Pod Linuxem jsou ovladače pro řadu MIDI kontrolerů většinou již součástí jádra systému a nemusíme se tedy obtěžovat s jejich instalací. Pod Windows a MacOSem je třeba nejprve patřičné ovladače doinstalovat.

Dalším krokem je zprovoznění MIDI v samotných Pd. Vstupní a výstupní MIDI porty nastavujeme v menu Media. K vlastnímu propojení kontroleru je v Linuxu vhodné použít aplikaci QJackCtl nebo Aconnectgui. V MacOSu použijte aplikace Audio MIDI Setup. Ve Windows postačí, když v menu Media -> Nastavení MIDI vyberete v položce vstupní a výstupní zařízení, které odpovídá vašemu kontroleru.

Jestliže máte kontroler připojen k počítači, nainstalovali jste ovladače, nastavili MIDI v menu Media a propojili jste kontroler s Pd, pak by komunikace kontroleru s Pd měla fungovat. Otevřete si test audia a MIDI (přes menu Meida). Objekty [notein] a [ctlin] by měly, po zmáčknutí klávesy na klaviatuře nebo použití kontrolního prvku (posuvník, knob), na výstup posílat číselné hodnoty.

[NOTEIN]_A_[CTLIN]

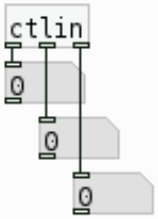
Objekt [notein] čte noty příchozí z klaviatury. Při stisku klávesy se vyšle MIDI zpráva "Note On" (klávesa stisknuta). Zpráva obsahuje informaci o čísle zahrané noty, její síle (velocity) a tzv. MIDI kanálu. [notein] má tedy tři výstupy: pro notu, její sílu a MIDI kanál.



Pokud chceme číst noty pouze z určitého MIDI kanálu, specifikujeme ho jako argument [notein].

Pouze MIDI kanál č. 10 -> notein 10

Objekt [ctlin] čte MIDI zprávy označované zkratkou CC (Control Change). Obvykle jde tedy o posuvníky, pitch-wheel, knoby, tlačítka atd. Podobně jako u [notein] je prvním výstupem aktuální hodnota daného kontrolního prvku, na druhém je jeho specifické číslo a třetí výstup je číslo MIDI kanálu.

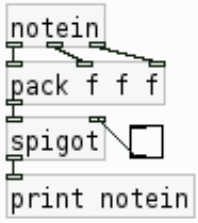


Klaviaturou i kontrolními prvky lze v Pd řídit části patche, které můžou dynamicky přijímat nějakou hodnotu. Klasicky objekt [notein] použijete k posílání MIDI not do

části patche, zaopatřující vytvoření tónu o dané výšce. Objekt [ctlin] pak často využijete často v kombinaci s abstrakcí [expr_scale], jímž lze jeho výstup škálovat na požadovaný rozsah.

[STRIPENOTE]

V případě jednohlasých (monofonních) syntetizérů najde uplatnění objekt [stripenote]. [notein] totiž funguje tak, že posílá dvě zprávy. První ve chvíli stlačení klávesy (Note On) a druhou v momentě jejího uvolnění (Note Off). Druhá zpráva má stejné číslo pro MIDI notu a kanál, pouze hodnota síly je nulová.



Zmáčkněte klávesu na klaviatuře, pak ji uvolněte a podívejte se do konzole. Uvidíte dvě zprávy. Ve druhé bude na druhé pozici nula - je to zpráva, kterou [notein] vysílá ve chvíli uvolnění klávesy.

Objektem [stripenote] tuto druhou zprávu odfiltrujeme, takže hlas monofonního syntetizéru nebude inicializován dvakrát.

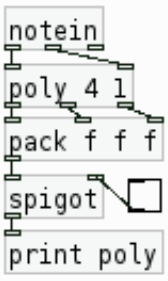
Odfiltruje zprávu Note Off.

Zapínáme/vypínáme výpis do konzole kvůli přehlednosti.

POLYFONIE

V případě vícehlasých (polyfonních) syntetizérů je situace trochu složitější. Musíme zajistit inicializaci jednotlivých hlasů syntetizéru - jejich zesilování ztišování. Počet hlasů se odvíjí od počtu stisknutých uvolněných kláves. K řešení této složitější situace nám poslouží objekt [poly] v kombinaci s objektem [route].

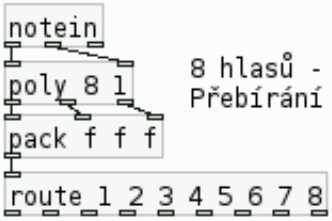
Při stisknutí klávesy pro ni objekt [poly] nejprve vytvoří jedinečné pořadové číslo a přidá k ní hodnotu MIDI noty a její síly. První agrument označuje, jaký maximální počet zmáčknutých kláves si má [poly] pamatovat - je to tedy číslo určující, kolikahlasý syntetizér je. Když zmáčkneme a držíme tři klávesy současně, vytvoříme tím tři zprávy s jedinečným číslem. Když pak jednu uvolníme, pošle [poly] zprávu s jedinečným číslem, jež odpovídá dané klávese. Objektem [route] tato jedinečná pořadová čísla "odchytáváme" a posíláme je ke zpracování jednotlivým hlasům syntetizéru.



Druhý argument [poly] je 1/0 a zapíná/vypíná tzv. přebírání hlasů. To znamená, že když v případě vedlejšího příkladu zmáčkneme 4 klávesy, držíme je a přimáčkneme k nim ještě pátou, pátý hlas přebere ten, který jsme vytvořili ve chvíli zmáčknutí první klávesy.

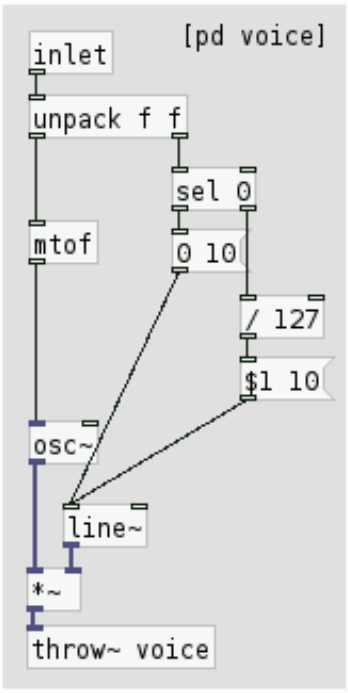
8MI_HLASÝ_SYNTETIZÉR

Předchozí látku budeme prakticky demonstrovat na konstrukci jednoduchého polyfonního syntetizéru. Část patche, která bude číst informace z MIDI kontroleru a vytvářet osm jedinečných zpráv odpovídajících až osmi současně stisknutým klávesám, bude vypadat takto:



8 hlasů - 8 zpráv s jedinečným pořadovým číselm.
Přebírání hlasů je zapnuto.

Objektem [route] budeme posílat seznam o dvou prvcích (nota a síla) jednotlivým hlasům. Nyní tedy musíme vytvořit část patche, která seznam zpracuje, zahraje notu o dané výšce a bude řídit obálku. Uzavřeme ji pro přehlednost do subpatche [pd voice]. Uvnitř bude vypadat takto:



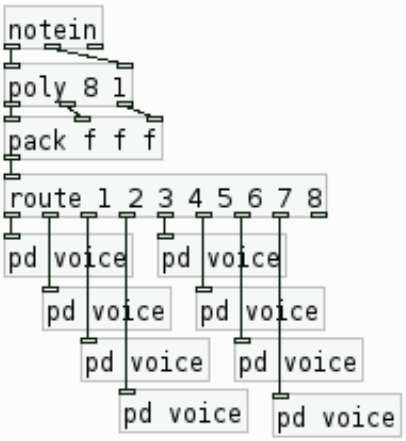
Nejprve seznam rozbalíme na samostatné hodnoty. Nalevo je hodnota MIDI noty a napravo její síla. MIDI notu převádíme na frekvenci objektem [mtof] a posíláme ji rovnou do [osc~].

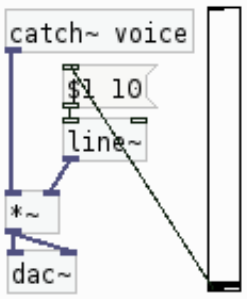
Jestliže uvolníme klávesu a hodnota síly noty je nulová, do [line~] posíláme zprávu [0 10(, což znamená, že za deset milisekund daný hlas ztišíme. Pakliže je síla různá od nuly, dělíme ji 127 (rozsah síly je v MIDI standardu v rozsahu 0..127) a posíláme ji do zprávy [$1 10(a následně do [line~].

Maximální hodnota v této velmi jednoduché AR obálce se řídí podle síly stlačení klávesy.

Signál posíláme bezdrátově na zvukovou sběrnici jménem voice.

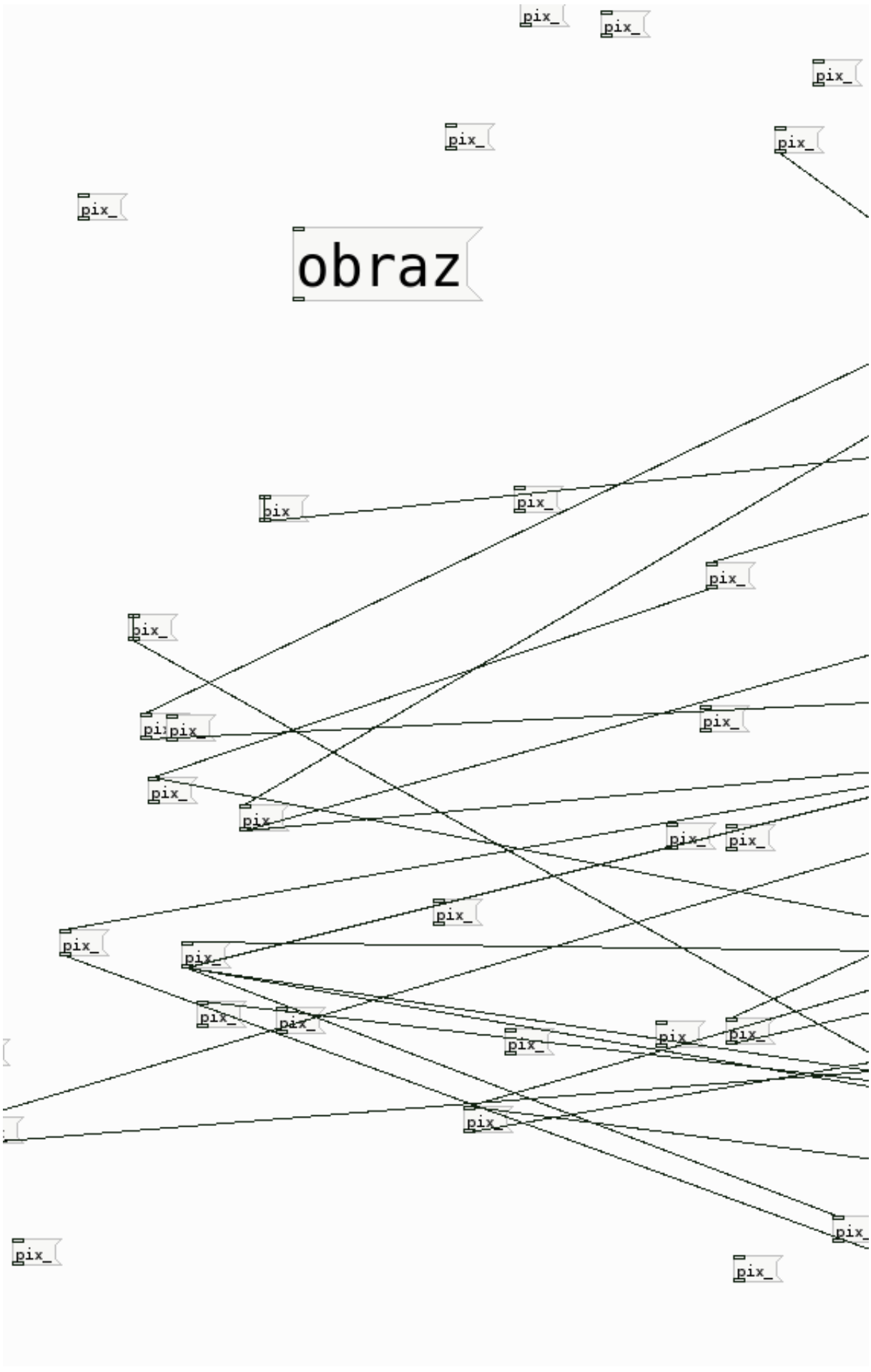
Propojíme-li nyní první část patche se subpatchi a ještě "chytáme" signál ze sběrnice voice, máme 8mihlasý syntetizér hotov.





Prověřte klaviaturou funkčnost syntetizéru a zkuste v [pd voice] nahradit oscilátor nějakým komplexnějším typem syntézy.

Ev. se pokuste vylepšit obálku tak, aby byla dynamicky nastavitelná.


obraz
pix_

knihovna_GEM

K zacházení s obrazem byly pro Pd vyvinuty knihovny pdp, GridFlow a GEM. Vzhledem k tomu, že součástí Pd-extended a multiplatformní (pro Linux, tak pro MacOS a Windows) je pouze knihovna GEM, budeme se věnovat právě jí.

pdp

gridflow

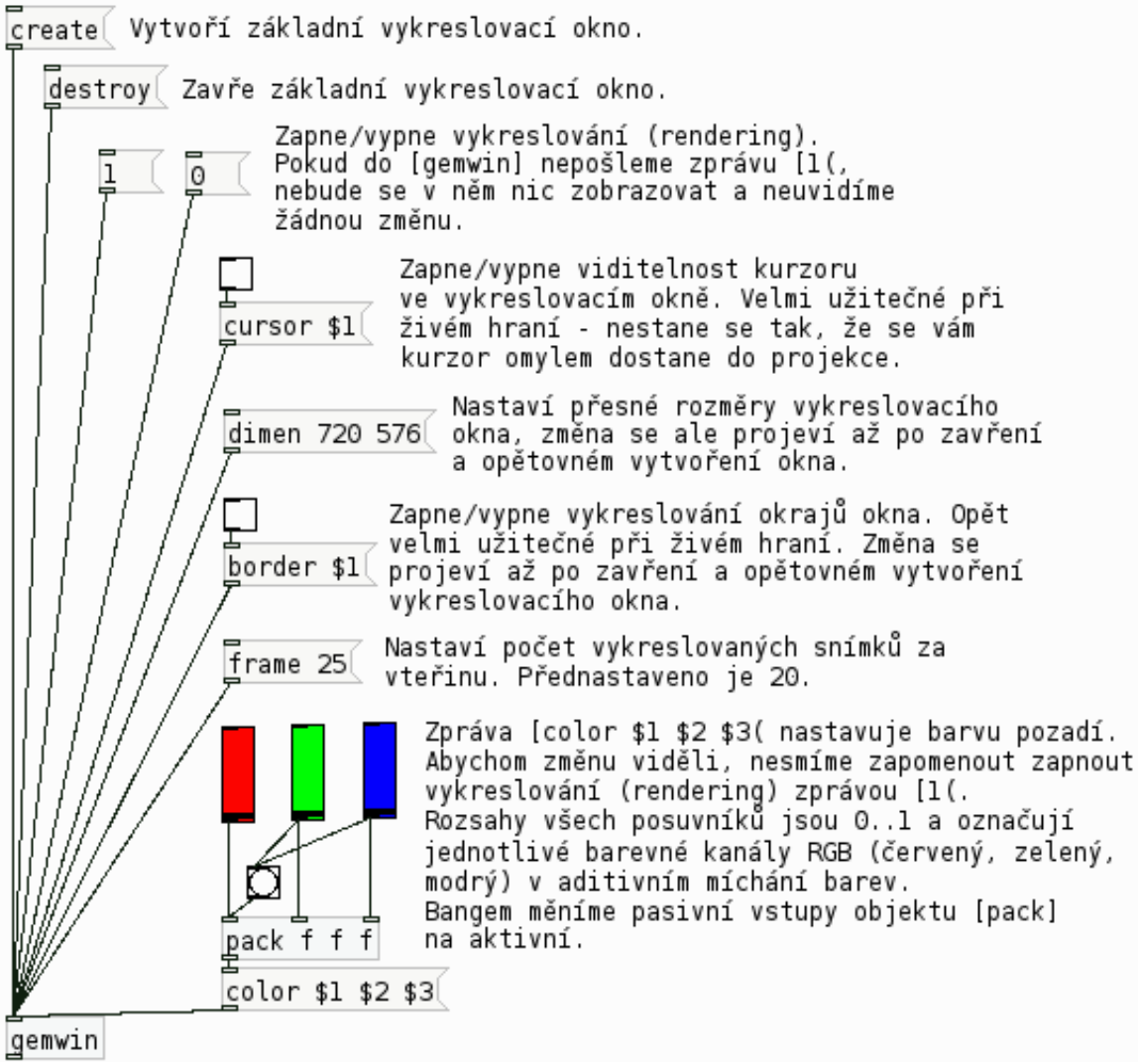
GEM

Původně ji naprogramoval Mark Danks a dnes je rozvíjena a udržována řadou dalších programátorů. Zkratka GEM znamená Graphics Environment for Multimedia a zprostředkovává nám funkce jako vytvoření okna, do něhož vizuální složky vykreslujeme, práci se statickým i pohyblivým obrazem, částicové systémy, postprodukční pluginy atd.

V případě Pd-extended se nemusíme starat o instalaci této externí knihovy - já automaticky načtena při jejich startu. Pokud byste v Pd konzoli zvětšili citlivost výstupu na Log 4 a prošli si historii vypsaných zpráv, narazíte v ní na řádek "GEM: Graphics Environment for Multimedia" a pár dalších, které nás informují o verzi a času kompilace.

VYTVOŘENÍ_ZÁKLADNÍHO_OKNA

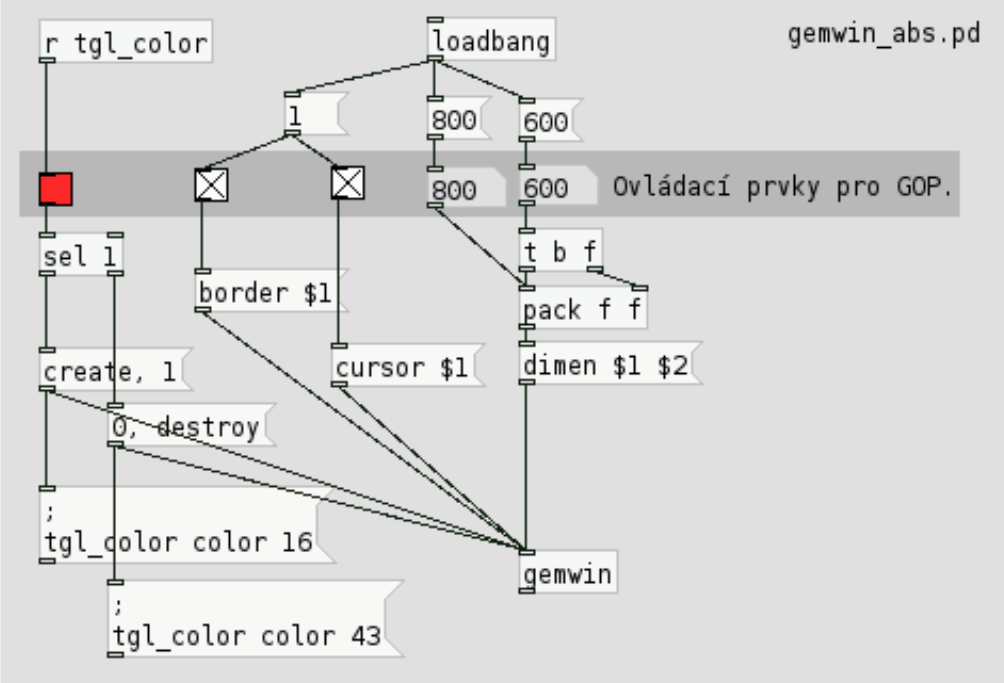
K vytvoření základního okna, do nějž se grafika vykresluje, slouží objekt [gemwin]. Zpráva [create(okno otevře a zpráva [destroy(je zase zavře. Další zprávy si ukážeme na následujícím příkladu:



Objekt [gemwin] umí přijmout ještě celou řadu parametrů. My jsme si ukázali jen ty základní. Pokud by vás zajímaly podrobnosti, podívejte se do nápovědy k tomuto objektu. Mimo jiné tam najdete informace, jak přepnout projekci z dvourozměrného způsobu promítání na 3D nebo jak zprovoznit tzv. FSAA vyhlazování.

Knihovna GEM je založena na standardu OpenGL, který k vykreslování grafiky používá výpočetní potenciál grafické karty. Závislost na OpenGL souvisí také s tím, že [gemwin]em lze vytvořit pouze jedno jediné okno. Pokud jste pokročilejšími programátory a znáte OpenGL nebo umíte vytvářet GLSL shadery, své znalosti můžete uplatnit i při práci s knihovnou GEM.

Z důvodu snadného přístupu k vykreslovacímu oknu si postavíme jednoduchou abstrakci na jeho ovládání. Pojmenujeme ji jako gemwin_abs.pd, nastavíme GOP a uložíme ji do adresáře s abstrakcemi.

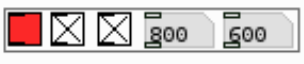


Prvním vypínačem budeme vytvářet a zavírat vykreslovací okno a zároveň zapínat a vypínat vykreslování - posíláme za sebou dvě zprávy: [create, 1(a [0, destroy(. Podle hodnoty tohoto vypínače nastavujeme interní zprávou i jeho barvu.

Další vypínač zapne/vypne vykreslování okrajů. Aby se změna projevila, je třeba vykreslovací okno nejprve zavřít a pak znovu vytvořit.

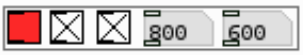
Třetí vypínač ovládá zobrazování kurzoru. Dva číselné boxy slouží pro nastavení rozměrů vykreslovacího okna. Přednastaveny jsou hodnoty 800 a 600.Pokud vám toto rozlišení nevyhovuje, hodnoty změňte. Velikost se změní také až po znovuvytvoření vykreslovacího okna.

V našem případě jsme rozměry GOP nastavili na 138 x 20 pixelů, takže výsledek vypadá takto:

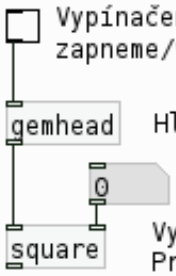


geometrické_objekty_a_operace_s_nimi

Nyní se budeme zabývat tím, jak do okna vykreslit základní geometrické objekty, řádem ve vykreslování, a také operacemi jako posun, rotace a škálování. Vložíme do patche abstrakci [gemwin_abs] a kliknutím na červený vypínač vytvoříme vykreslovací okno.



Pokud chceme v okně něco zobrazit, musíme vytvořit tzv. vykreslovací řetězec. Ten sestává vždy z hlavičky - objektu [gemhead] a je ukončen objektem, který vykreslujeme.

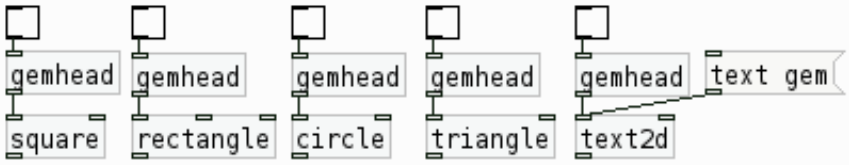


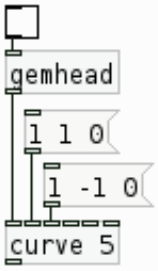
Vypínačem zapojeným do [gemheadu] celý řetězec zapneme/vypneme.

Hlavička řetězce.

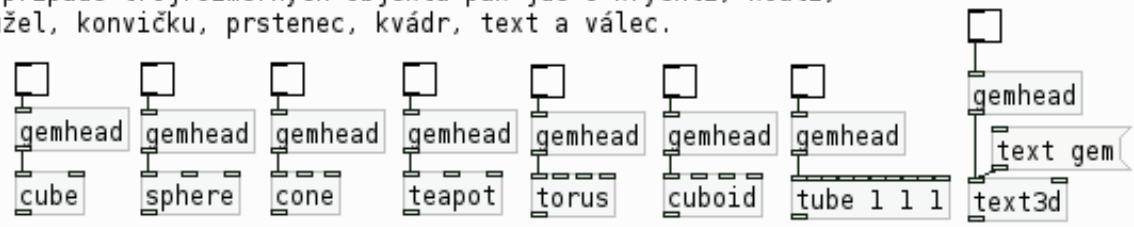
Vykreslovaný objekt - v tomto případě čtverec. Pravým vstupem měníme jeho velikost.

Mezi základní geometrické objekty, které umisťujeme obvykle na konec řetězce, patří šest následujících: čtverec, obdélník, kruh, trojúhelník, text a křivka. Objekt uvidíte, když zapnete daný vykreslovací řetězec. Nezapínejte ale víc než jeden, jinak by se objekty překrývaly.



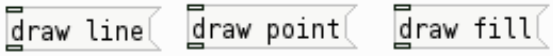


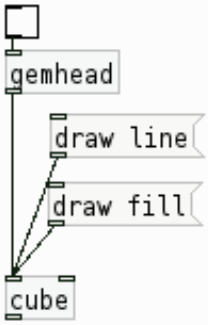
V případě trojrozměrných objektů pak jde o krychli, kouli, kužel, konvičku, prstenec, kvádr, text a válec.



Zkuste do pasivních vstupů jednotlivých objektů zapojit číselný box, stejně jako v případě prvního uvedeného příkladu můžete měnit jejich velikost. Objekt [rectangle] má pochopitelně dva vstupy, jimiž měníme rozměry na ose X a Y. U 3D objektu [sphere] pravý vstup určuje množství segmentů, ze kterých je objekt vykreslen. Pro bližší seznámení se se vstupy objektů jako je [curve], [cone], [torus], [cuboid] a [tube] se podívejte do nápovědy k těmto objektům.

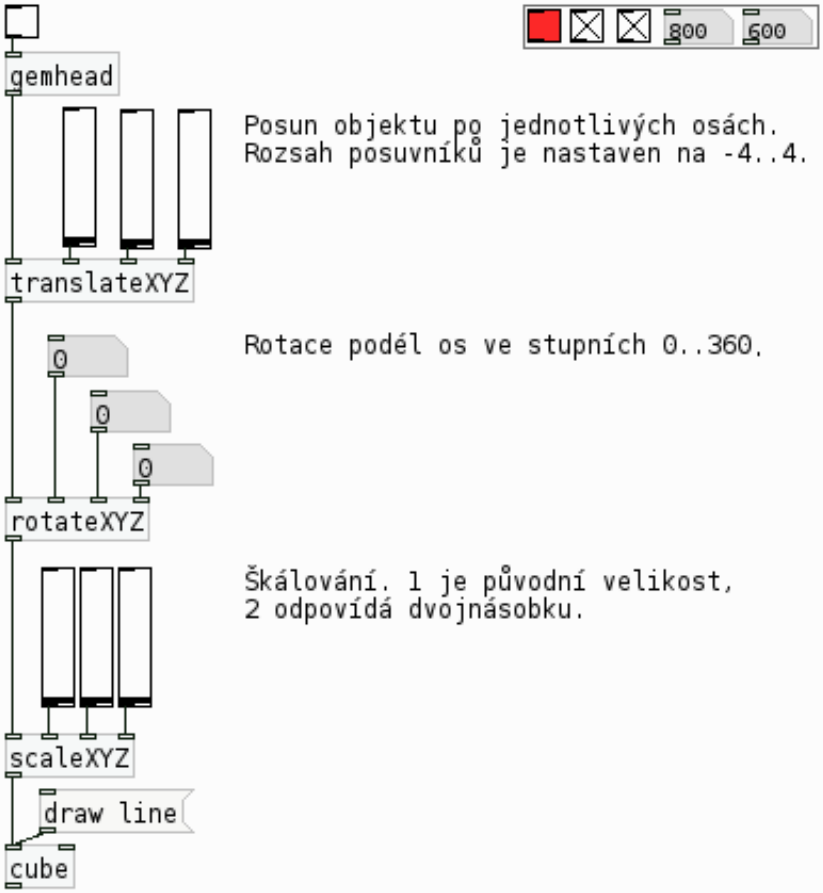
Struktura některých objektů vám bude zjevnější, když do jejich aktivního vstupu ještě zapojíte zprávu, která určuje způsob vykreslení - celkem jsou tři: vyplněné plochy, hrany a body.





POSUN_ROTACE_ŠKÁLOVÁNÍ

S vytvořenými objekty je ve scéně možné s pomocí objektů [translateXYZ], [rotateXYZ] a [scaleXYZ] provádět základní operace jako je posun, rotace a škálování. Jednotlivé vstupy odpovídají osám X, Y a Z.

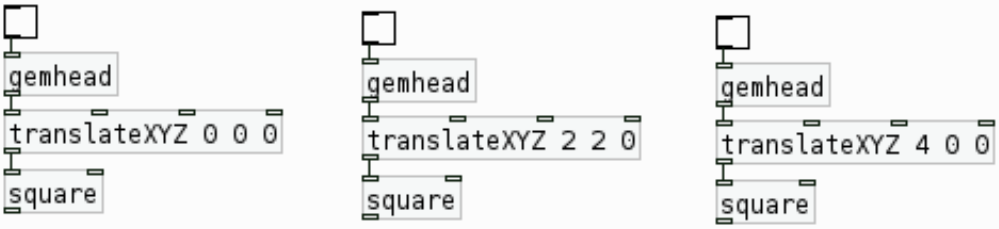


Posun objektu po jednotlivých osách. Rozsah posuvníků je nastaven na -4..4.

Rotace podél os ve stupních 0..360.

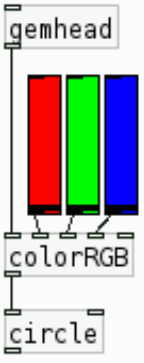
Škálování. 1 je původní velikost, 2 odpovídá dvojnásobku.

Přesné hodnoty lze objektům zadávat jako argumentny při vytváření. V následujícím příkladu předáváme hodnoty pro posun. V okně uvidíte tři čtverce, které se dotýkají rohy. Dvě jednotky posunu tedy odpovídají přesně délce hrany.



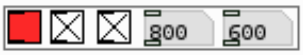
BARVA_A_PRŮHLEDNOST

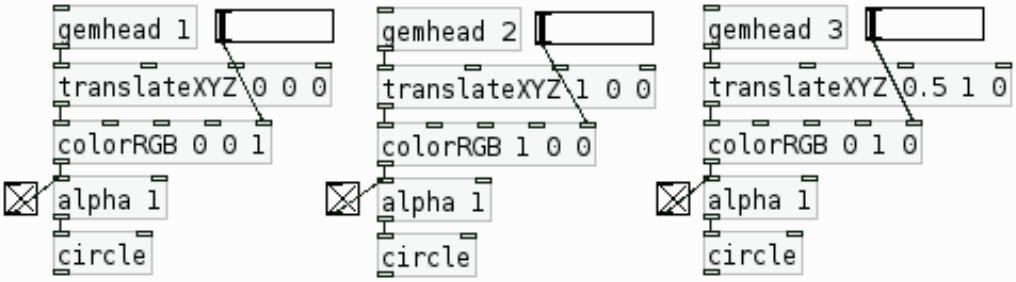
Barvu nastavujeme objektem [colorRGB]. Do vstupů pro každý jednotlivý barevný kanál (RGB - červená, zelená, modrá) posíláme hodnoty v rozsahu 0..1. Pd, ostatně jako i jiné programy pro zpracování počítačové grafiky používají tzv. aditivní způsob míchání barev, používaný v monitorech a projektorech - mícháme zde vyzařované světlo.



Pokud je hodnota ve všech kanálech 0, výsledkem je černá, pokud je 1, vidíme bílou. Když vytáhneme posuvník jedním kanálem nahoru a ostatní dva budou dole, bude výsledkem maximální barva v daném kanálu.

Čtvrtým vstupem [colorRGB] nastavujeme jeho průhlednost (tzv. alfa kanál), a to opět v rozsahu 0..1, který odpovídá škále od úplné průhlednosti po úplnou viditelnost. Změna se ale projeví teprve poté, co do vykreslovacího řetězce zařadíme i objekt [alpha]. Vyzkoušejte si chování alfa kanálu na následujícím příkladu. Průhlednost objektů se zapíná/vypíná hodnotami 1/0 poslanými do levého vstupu objektu [alpha]. Díky průhlednosti jednotlivých barevných kruhů vzniknou jejich kombinací další barvy: žlutá, purpurová, azurová a bílá.



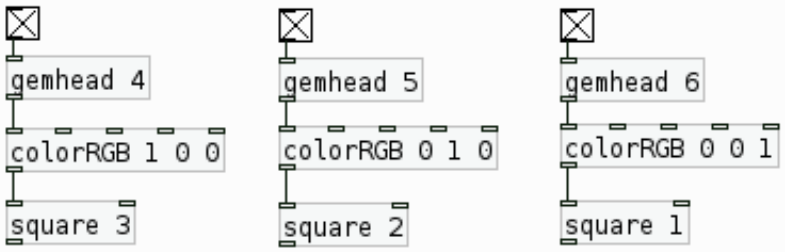


ŘÁD_VYKRESLOVÁNÍ

Pokud jste předchozí příklad prozkoumali a vypnuli jste u všech kruhů používání průhlednosti (poslali jste nulu vypínačem do objektu [alpha]), pak vidíte, že se kruhy překrývají v určitém pořadí. Tím se dostáváme k problematice řádu vykreslování.

Možná jste postřehli, že u hlaviček jednotlivých vykreslovacích řetězců je číslo - to specifikuje pořadí, ve kterém se objekty vykreslují, a tedy i způsob jejich překrytí. Modrý kruh začíná hlavičkou [gemhead 1] a byl vykreslen jako první, takže je ve vrstvách úplně vespod. Červený kruh má hlavičku [gemhead 2] a překrývá modrý kruh.

Následující ukázka vykreslí tři čtverce, u nichž v [gemhead]ech specifikujeme pořadí vykreslování. Ve scéně máme již tři objekty, takže první (červený a největší) čtverec bude mít hlavičku s číslem 4 a překryje tak již vykreslené kruhy.

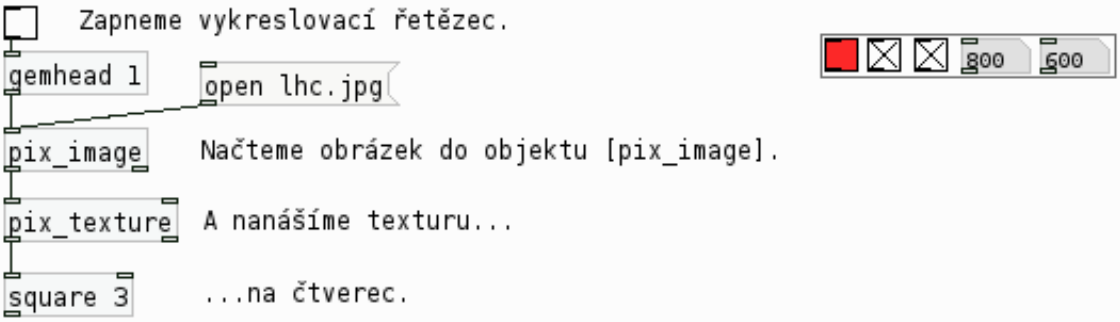


CVIČENÍ:

- postavte si interaktivní patch, který bude vykreslovat jeden objekt; pokuste se pomocí objektu [cursor] číst údaje o poloze myši a ovládejte jimi posun, škálování, rotaci i barevnost. Jakou z těchto akcí bude myš řídit, specifikujte stlačením kláves Q, W, E a R; využijete tedy i objekty [key], [demux], [pack], [unpack] a [expr_scale]

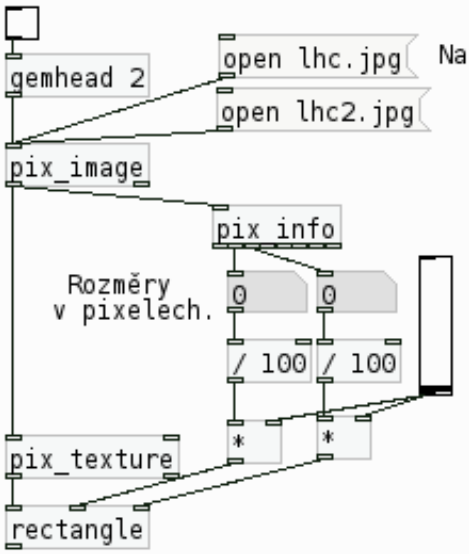
rastrová_grafika

K načítání rastrové grafiky slouží v Pd objekt [pix_image], do jeho pravého vstupu posíláme zprávu [open název souboru(. GEM podporuje zobrazování souborů ve formátu JPEG a TIFF. V zobrazovacím řetězci za [pix_image] musí následovat objekt [pix_texture], jenž se stará o namapování textury na objekt na konci řetězce. V případě rastrové grafiky obvykle budeme chtít namapovat obrázek na dvourozměrnou plochu - tedy na čtverec [square] nebo obdélník [rectangle].



Ve vykreslovacím okně bychom měli vidět fotografii - je zde ale problém týkající se jejího horizontálního zkreslení. Načtenou fotografii totiž nanášíme jako texturu na čtverec a přitom rozlišení fotografie čtvercové není.

Abychom fotografii viděli v odpovídajících dimenzích, bude třeba ji namapovat na objekt [rectangle]. S ním ale budeme mít obdobnou potíž. Jak zajistit to, aby měl obdélník rozměry odpovídající rozlišení načítané fotografie? Pomůže nám k tomu objekt [pix_info], který čte informace z objektu [pix_image] a na druhém a třetím výstupu dává rozměry grafiky v pixelech. Tato čísla pak škálujeme poděním hodnotou 100 a posíláme je do vstupů objektu [rectrangle]. Celkovou velikost obrázku pak můžeme doladit posuvníkem, který hodnoty pro vertikální a horizontální dimenzi násobí hodnotou v rozsahu 0.2..1.



Tento patch se chová dynamicky - kdykoliv načteme obrázek, dimenze obdélníku se přizpůsobí jeho rozlišení.

CVIČENÍ:

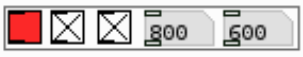
- pokuste se vykreslovací řetězec v předchozím příkladu rozšířit o objekty [translateXYZ], [rotateXYZ], [scaleXYZ] a [colorRGB]; načítání obrázků zkuste zajistit objektem [openpanel]
- výsledný zobrazovací řetězec pak několikrát zduplikujte, nastavte správně řád vykreslování u hlaviček a rozmístěte různě naškálované a rotované obrázky do vykreslovacího okna
- pokuste se vytvořit animovanou koláž; jednotlivé parametry pro posun, škálování atd. ovládejte kombinací objektů [metro], [cxc_counter] a [expr_scale]

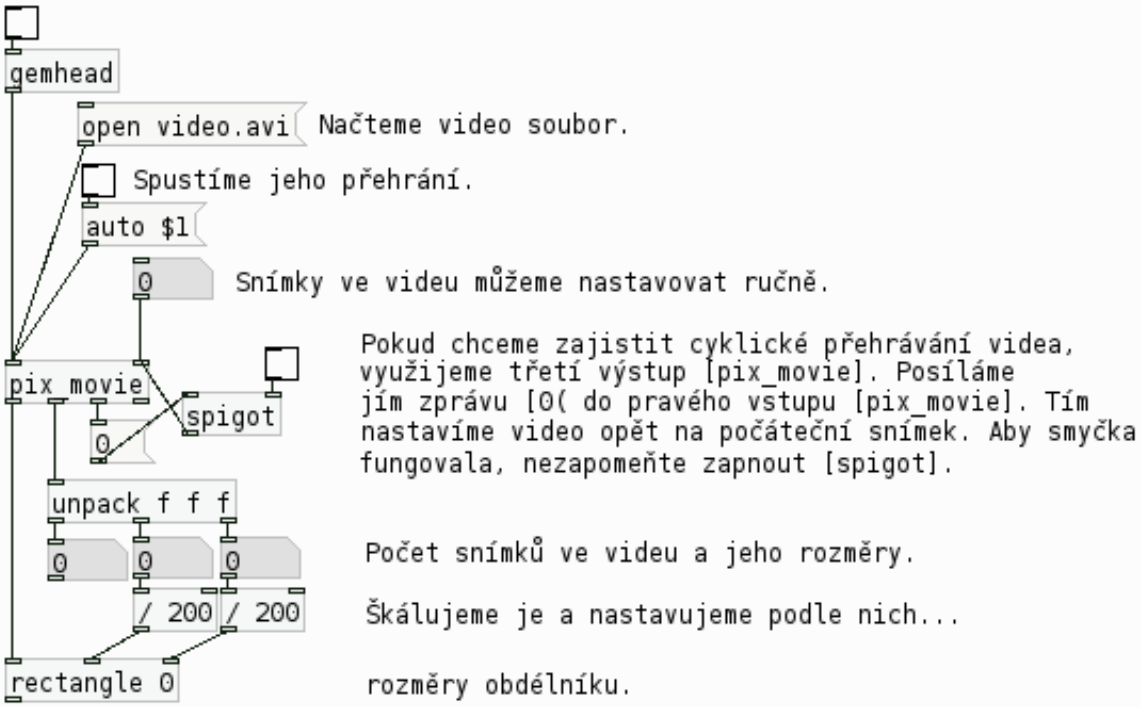
pohyblivý_obraz

Knihovna GEM obsahuje kromě objektu pro vykreslení statické rastrové grafiky i objekty [pix_movie] a [pix_film], kterými můžeme přehrávat video soubory. Situace je zde trochu komplikována množstvím různých formátů a kodeků. Obecně se dá říct, že pod MacOSem GEM umí zpracovat formát MOV (QuickTime), ve Windows pak soubory typu AVI a pod Linuxem obvykle jakýkoliv formát, s jehož podporou byla daná verze GEMu zkompilována.

Pokud se vám nějaký video soubor nepodaří načíst, zkuste jej zkonvertovat do jiného formátu. Dobře vám k tomu poslouží např. aplikace Avidemux. MPEG2 a M-JPEG by měly fungovat na všech platformách.

Video načítáme zprávou [open jméno souboru(. Zpráva [auto $1(v kombinaci s vypínačem spustí přehrávání videa. Přehraje ho jen jednou. Pravým vstupem [pix_movie] lze zobrazit ten který snímek z načteného videa. Rozbalíme-li seznam z druhého výstupu [pix_movie], pak obdržíme tři hodnoty: první je délka videa ve snímcích, druhá a třetí jsou pak rozměry videa v pixelech. Na třetí výstup [pix_movie] je poslán bang po dosažení posledního snímku videa.





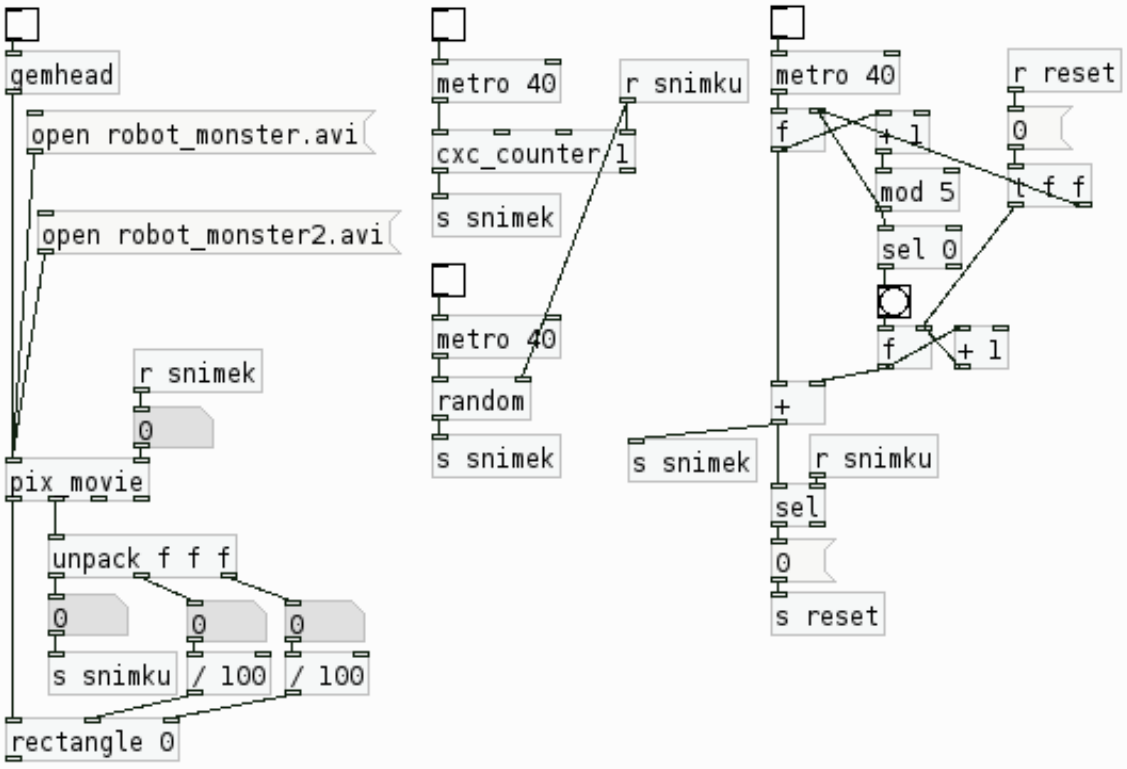
PŘEHRÁVÁME_ALTERNATIVNĚ

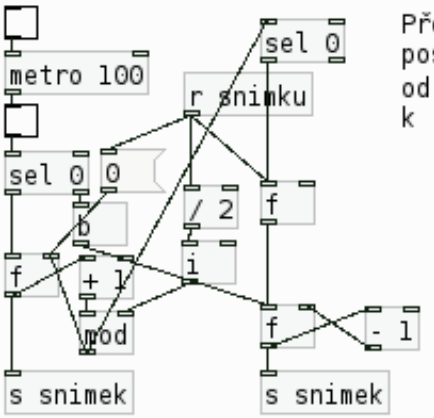
Pravý vstup objektu [pix_movie] nabízí přístup k jednotlivým snímkům videa a umožňuje alternativy v jejich řazení. Konstrukcí různých typů počítadel tak můžeme vytvářet různé způsoby interpretace daného videomateriálu.

Náš videosoubor má 25 snímků ve vteřině. Abychom jich v jedné vteřině tolik přehráli, vydělíme 1000 milisekund počtem snímků a dostaneme čas 40 ms. To je údaj pro metro, které bude pohánět počítadlo přehrávající video.

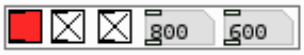
Obyčejný přehrávač postavíme na objektu [cxc_counter]. Náhodný přehrávač pak na objektu [random], do kterého posíláme celkový počet snímků. Pokročilejší přehrávač (jako od Martina Arnolda) sestává ze dvou počítadel: jedno přehrává smyčku o pěti snímcích a druhé tuto smyčku po jednom snímku posouvá dopředu.

Martin Arnold





Přehrávač, který postupuje od od okrajů videa k jeho středu.



CVIČENÍ:

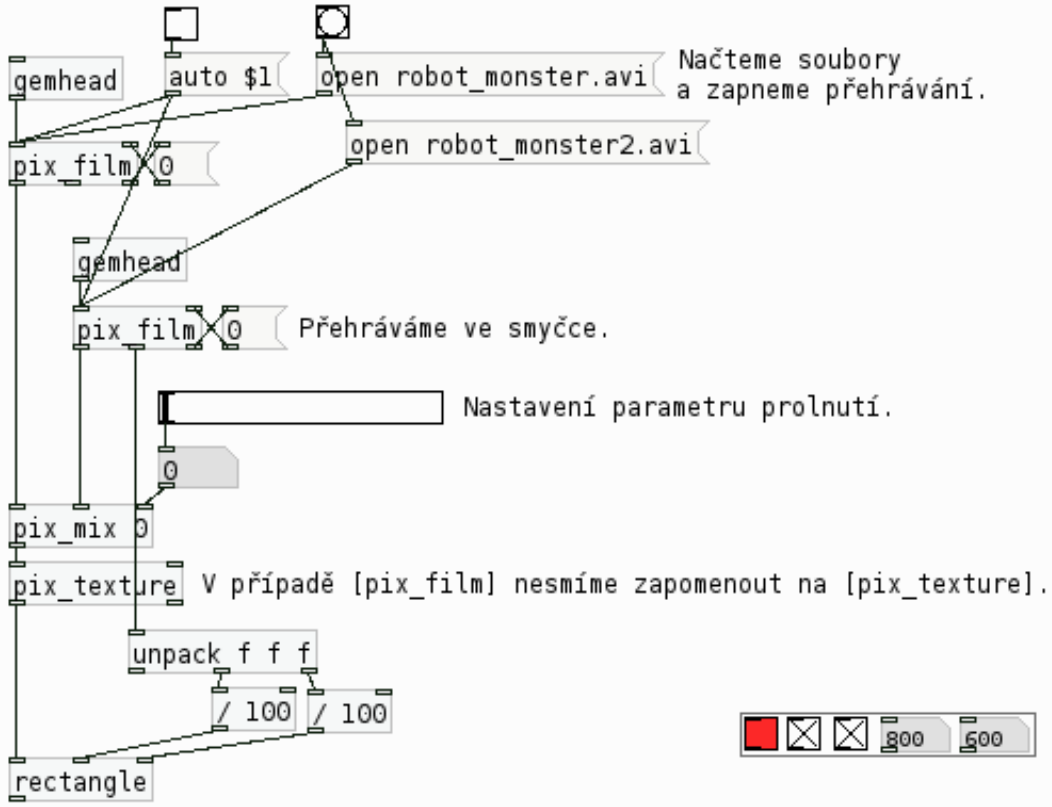
- zkuste navrhnout vlastní přehrávač videa, využijte různých typů počítadel, jejich kombinací a směrů počítání (reverzní, ping-pong)

VIDEO_MIX

Asi jste si u předchozích patchů všimli, že za objektem [pix_movie] v zobrazovacím řetězci není objekt [pix_texture]. Ten už má totiž [pix_movie] v sobě implementován. Když potřebujeme jen jednoduše přehrát video, pak použijeme [pix_movie]. Pokud ale chceme po cestě s videosignálem ještě provádět další operace - např. měnit jeho jas, kontrast nebo videosignály promíchat, sáhneme po objektu [pix_film], jenž se podobá [pix_movie], až na to, že se nestará o texturování.

Jádrem video mixu je objekt [pix_mix], jehož první dva vstupy jsou určeny pro obrazová data ([pix_mix] ale umí prolnout i statické obrazy). Třetí slouží k nastavení parametrů prolnutí. Jde o rozsah 0..1, přičemž 0 odpovídá levému signálu a 1 pravému. Posuvníkem s tímto rozsahem lze dvě videa jemně prolínat.

K tomu, aby [pix_mix] fungoval, je zapotřebí, aby oba videosoubory měly stejné rozlišení a aby oba [pix_film]y do [pix_mixu] dodávaly signál.



OHÝBÁNÍ_VIDEO_SIGNÁLU

Knihovna GEM obsahuje sbírku objektů, které začínají předponou pix_, s nimiž lze na obrazových datech provádět různé operace. V zásadě jde o sbírku tradičních "postprodukčních" video efektů. Podívejte se do dokumentace knihovny GEM (Nápověda -> Pd Help Browser -> Gem), kde najdete řadu dokumentačních souborů k jednotlivým pix_ objektům. Podívejte se např. na dokumentaci k těmto objektům:

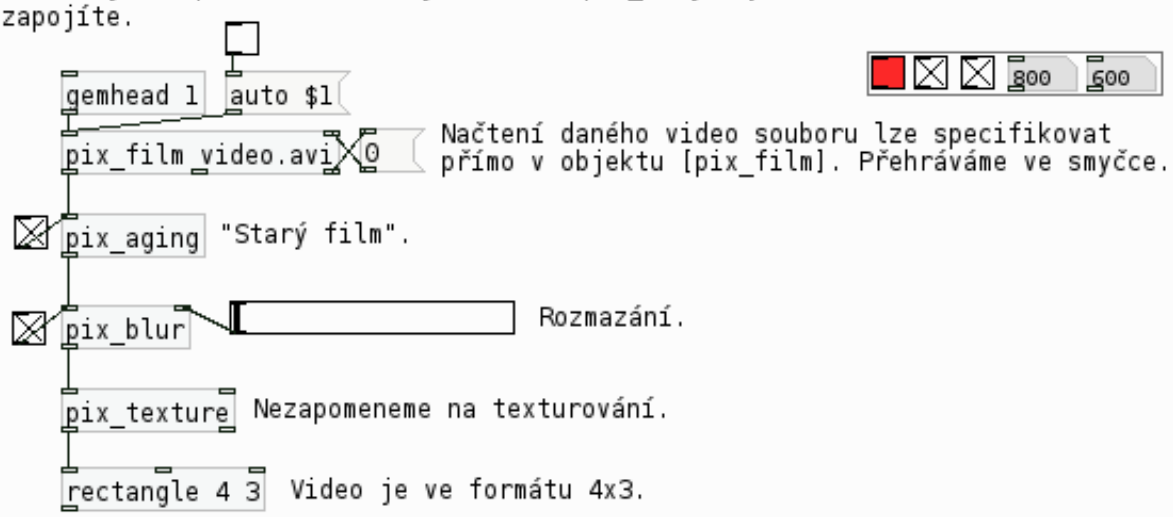
pix_gain pix_blur pix_aging pix_lumaoffset

pix_refraction pix_contrast pix_kaleidoscope Atd.

Umístěním těchto a dalších pix_ objektů v zobrazovacím řetězci mezi [pix_movie] dosáhneme různých modulací videosignálu. Podívejte se na legendární záznam Dana Sandina, ve kterém představuje svůj analogový Image Processor. S pomocí pix_ objektů můžeme obraz ohýbat podobně.

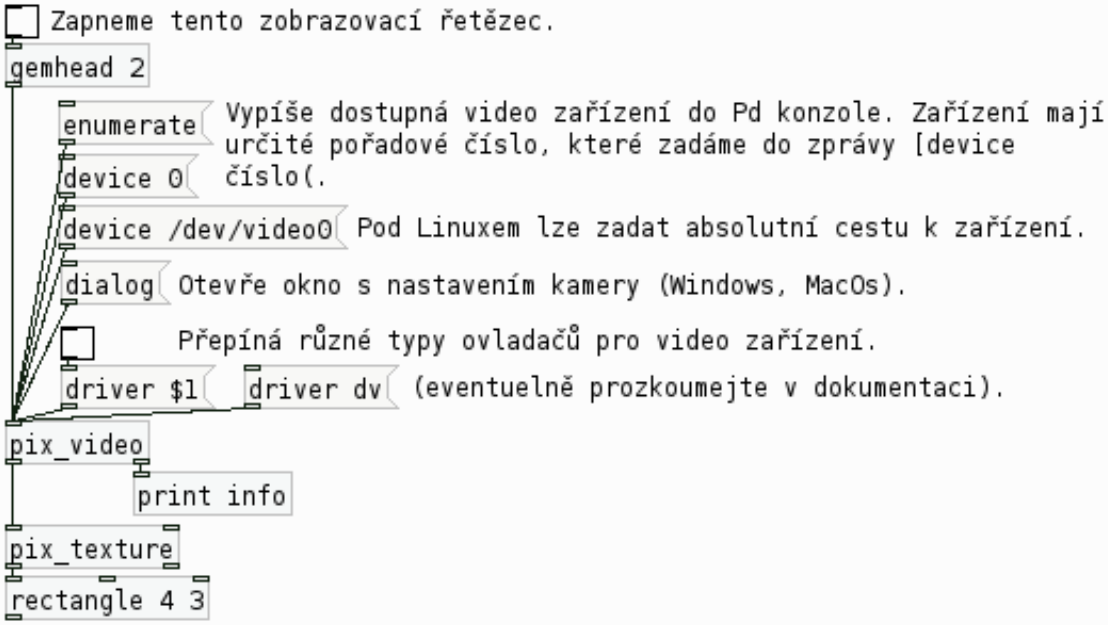
Dan Sandin a IP

Jednotlivé pix_ objekty mají různý počet vstupů, kterými lze řídit jejich parametry. Nebudeme se jim zde věnovat jednotlivě - jistě sami dokážete na základě studia dokumentace a vlastního experimentování zjistit, co který dělá. Pix_ objekty lze v řetězci zapínat a vypínat posláním zprávy [1(nebo [0(do jejich levého vstupu. Záleží také na tom, v jakém pořadí za sebe jednotlivé pix_ objekty zapojíte.



ŽIVÉ_VIDEO

Pokud máte k dispozici web kameru nebo v počítači máte firewire rozhraní, do kterého lze zapojit kameru, pak v Pd můžete pracovat i s živým videem. Kameru nejprve připojíme k počítači, a pak teprve zapneme Pd. Signál z kamery zprostředkovává objekt [pix_video].

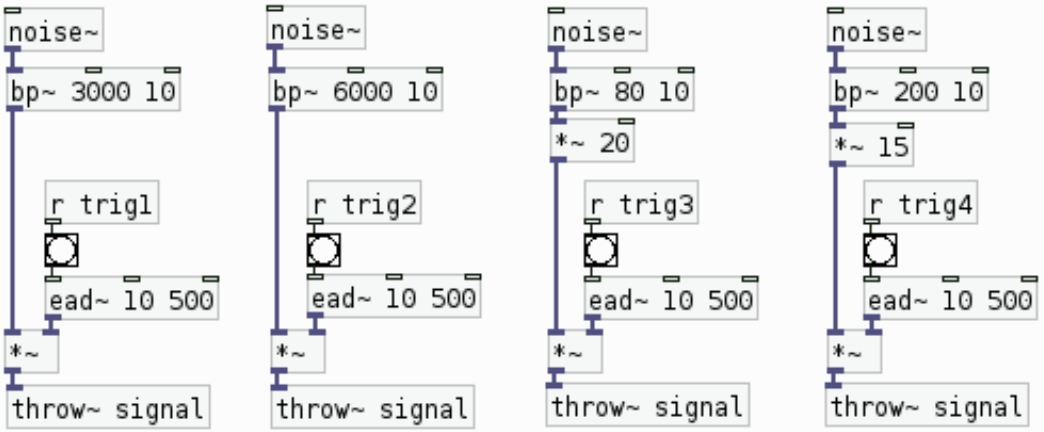


Probrali jsme jen některé z oblastí knihovny GEM a mohli bychom v průzkumu dále pokračovat. Nabízela by se témata jako zpětná vazba v obrazu, GLSL shadery, analýza videa pomocí implementace OpenCV pro GEM, import 3D modelů, spolupráce GEMu s knihovnou pmpd, videomapping atd. Náš výklad zde ale nyní přerušíme. V základech jsme probrali možnosti práce se statickým i pohyblivým obrazem a zacházení s ním, což poskytuje snad dostatečnou škálu možností pro započetí vlastních experimentů a pokusů o stavbu vizuálních nástrojů. Budete-li chtít ve studiu pokračovat, můžete nahlédnout do oficiální dokumentace (Nápověda -> Pd Help Browser -> GEM).

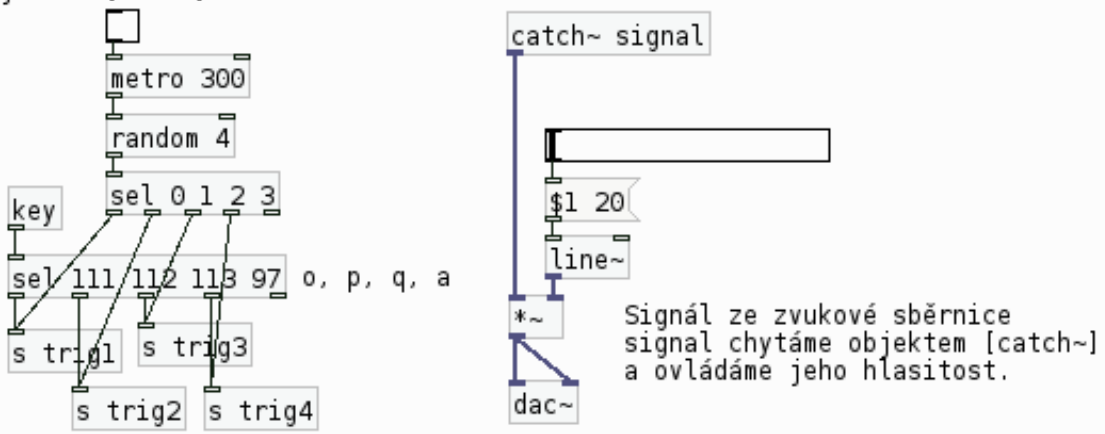
propojení

Poslední kapitola věnující se knihovně GEM nebude vlastně o ní, ale spíš o specifickém přístupu k tvorbě audiovizuálních nástrojů. Obraz, zvuk i jednotlivé události jsou v Pd reprezentovány digitálně, což umožňuje jejich vzájemné propojení. Na následující jednoduché případové studii konstrukce audiovizuálního nástroje si možnosti tohoto propojení ukážeme.

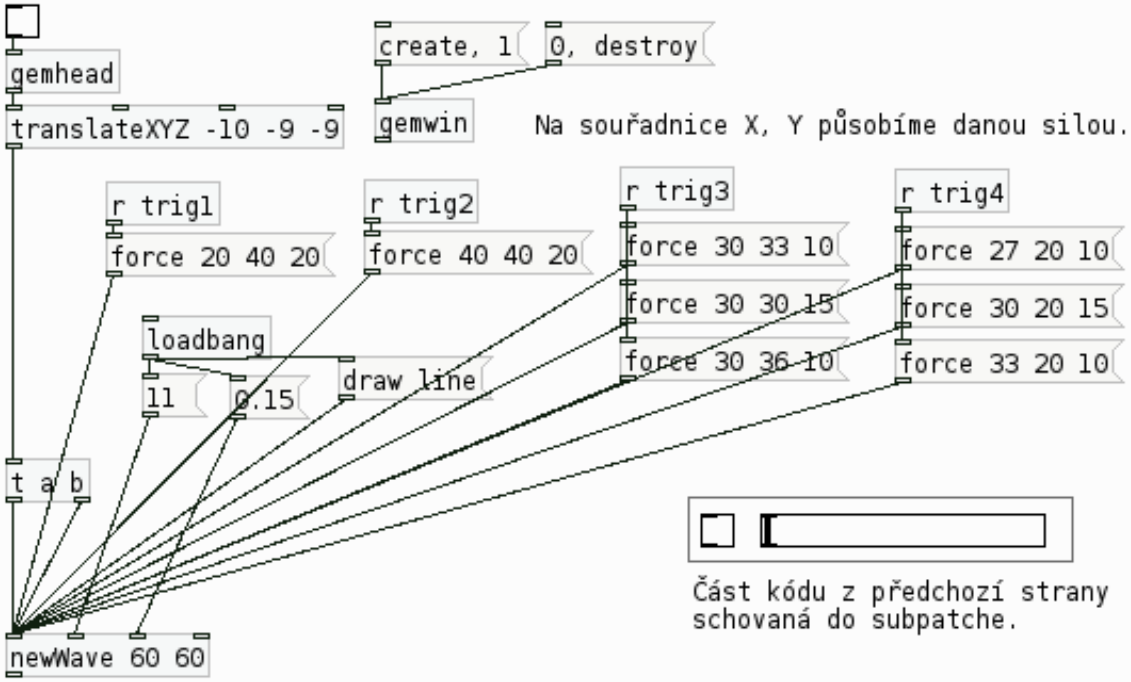
Nejprve si postavíme čtyři jednoduché syntetizátory. Z bílého šumu pásmovou propustí vyfiltrováváme určité frekvence a s pomocí [ead~] vytváříme jednoduchou obálku.



Nyní si postavíme "řídící" jednotku. Jednotlivé obálky budeme inicializovat "bezdrátově" jednak klávesami "o", "p", "q", "a" a jednak náhodně v pravidelných intervalech objektem [metro].



Nyní přichází na řadu programování vizuální složky. Zajistí ji objekt [newWave], což je dvourozměrná mřížka, do které můžeme "sahat" a "rozechvívat" ji tím, že na určité souřadnice v ní působíme určitou silou. Detailně popisovat vlastnosti [newWave] nebudeme, ev. je najdete v nápovědě.

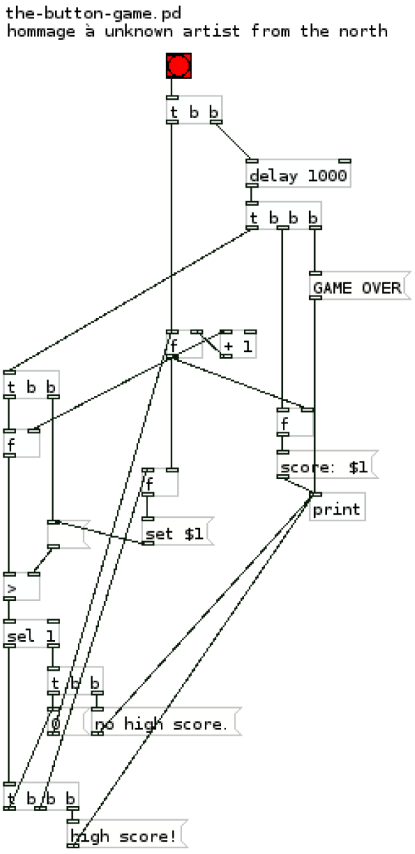


Když nyní vytvoříte vykreslovací okno, uvidíte v něm mřížku a klávesy O, P, Q a A (nebo spuštění metra) vyvolají událost jak ve zvuku, tak v obraze. Sestrojili jsme velmi jednoduchý audiovizuální nástroj, na který můžeme klávesami hrát.

Vzájemné propojení obrazu a zvuku se může odehrát na sofistikovanější úrovni. Pd-extended obsahují objekty jako [fiddle~], [bonk~], [pix_pix2sig~] a [pix_sig2pix~], které jsou určeny pro analýzu zvukových i vizuálních dat a jejich prostřednictvím lze audiovizuální složky semknout mnohem řízenějším způsobem. Tyto techniky však již ponecháme vlastní tvořivé a studijní dispozici čtenáře.

CVIČENÍ:

- pokuste se v závěrečném cvičení zúročit nastudovanou látku a sestrojte si audio-vizuální nástroj, který by odpovídal vašim vlastním představám. Pokud by vaše představa nástroje přesahovala oblasti, kterými jsme se v rukověti zabývali, zkuste si danou látku doplnit samostudiem dokumentace. Dobrým pomocníkem vám může být i Pd fórum nebo mailing list. Přeji vám v konstrukci a hraní na daný nástroj hodně zdaru!


the-button-game.pd
hommage à unknown artist from the north
t b b
delay 1000
t b b b
GAME OVER
f
+ 1
t b b
f
f
score: $1
f
print
set $1
>
sel 1
t b b
0
no high score.
t b b b
high score!

4 | Aplikace

Naše schopnosti nejsou zjevné, protože je nepoužíváme. Když stroj dělá lidskou práci, člověk tím o něco přichází. Kdysi jsme zkoumali galaxii stejně jako vy. / Vy znáte warpový pohon? / Ano, známe, ale nemůže nás donést jinam, než pryč odsud.

Star Trek: Insurrection

V předchozí kapitole jsme se seznámili se základy programovacího jazyka Pure Data a pouze v obrysech jsme nastínili, jaké možné způsoby užití v sobě tento jazyk nese. Předmětem následujících stránek budou ukázky praktických aplikací Pure Dat a „pokročilejší řečové akty". Nejprve bych rád představil čtyři díla ze světové scény, na nichž se ukazuje původní orientace Pd jakožto nářadí pro hudební skladatele a sound-artisty. Následovat bude popis tří patchů z mé vlastní dílny, které souvisejí s oblastí živé audiovizuální tvorby. Kapitolu uzavírají (ale zároveň doširoka otevírají) texty různých autorů, jež Pd používájí ve své umělecké praxi.

Snad čtenář dříve porozuměl určité zdrženlivosti týkající se definice toho, čím Pd jsou. Tato kapitola je příležitostí přistoupit k odpovědi na otázku „z jiné strany". Domnívám se, že by na základě srovnání uměleckých děl realizovaných v Pd bylo možné „vydestilovat" jejich „charakter" a vidět tak lépe stereotypy v přístupu k nim. Můj záměr je ale opačný – spočívá spíš v naznačení základní škály a různosti aplikací. Každý z autorů uvedených v této kapitole reprezentuje specifickou oblast tvorby, specifické metody a přístupy. Jsou to právě tyto vklady, s nimiž autor přichází k Pd a „otiskuje je" do nich. Z povahy otisků pak lze odčítat, čím pro kterého autora Pd jsou.

Předchozí věta nemá nic společného s adorací, ale spíš se střízlivým viděním Pd jako nářadí/nástroje, jež díky své modularitě a „měkkosti" dokáže nabývat různých podob.

4.1 | Peter Ablinger: Deus Cantando

V lineckém sídle festivalu Ars Electronica může návštěvník příležitostně vidět instalaci Petra Ablingera[1], která sestává z klavíru, elektromechanického zařízení sestrojeného ze solenoidů, jež ovládá klaviaturu a jednoho řídícího počítače. Technická stránka instalace je výsledkem delšího uměleckého výzkumu, v jehož jádru pro Ablingera jako pro hudebního skladatele stojí souvislost barvy lidského hlasu a klavíru.

Toto téma je přítomné již v cyklu hudebních kompozicí s názvem *Voices and Piano*, jímž Ablinger komentuje a ilustruje intonaci lidské řeči. Vybírá si konkrétní záznamy promluv postav jako Bertold Brecht, Morton Feldman, Agnes Gonxha Bojaxi, Mao Ce-tung, Arnold Schoenberg atd.[2] Kompoziční postupy z *Voices and Piano* pak přivádí k radikálnějšímu semknutí v případě kompozic *Schaufensterstück*, *Weiss/Weisslich 11B* a v cyklu s názvem *Quadraturen III*, pod který spadá kromě skladeb *Gegrüßet seist Du, Maria* a *A Letter from Schoenberg* také kompozice *Deus Cantando*. V případě posledních zmíněných prací Ablinger ve spolupráci s Winfriedem Ritschem a Thomasem Musilem sestrojil mechnizovaný klavír, u kterého je každá klávesa ovladatelná aplikací naprogramovanou v Pd. Ta umí ve výsledku „zahrát" takové kombinace tónů, které se ve složení spektra přibližují fonémům – takže klavír dokáže promlouvat. Winfried Ritsch i Thomas Musil jsou kmenoví členové Institutu pro elektronickou hudbu a akustiku (IEM) v Grazu. Musil pro Pd naprogramoval řadu externích knihoven (iemlib).

Mechanizovaný klavír, který předříkává Deklaraci mezinárodního soudu pro životní prostředí.

[1] *Ablinger.mur.at* [online]. [cit. 2014-03-10]. Dostupné z: http://ablinger.mur.at/.

[2] P. Ablinger - Bertolt Brecht. In: *Vimeo* [online]. [cit. 2014-04-08]. Dostupné z: http://vimeo.com/81107458. Kanál uživatele Daniel Buendía Barceló.

Kompozici *Deus Cantando* propůjčil k analýze svůj hlas mladý Miro Marcus, který předčítal Deklaraci mezinárodního soudu pro životní prostředí. Jejími autory jsou Adolfo Perez Esquivel a 14. dalajláma. Setkání etického textu s vyspělou technikou, na platformě hudební kompozice, jejíž titul odkazuje ke zpívajícímu Bohu, vytváří dostatečně bohatý interpretační rámec, jehož hranice zde nechci vlastními výklady omezovat.

Autor hovoří o svém přístupu jako o *fonorealismu*, tedy o kompoziční technice, kdy je s pomocí hardwarového a softwarového aparátu interpretační škála možností klavíru posunuta k výrazu připomínajícímu lidskou řeč. Ablinger sám také prohlásil, že to, co se v jeho díle ukazuje, není na pomezí koncertní síně, nýbrž že je to právě samotné pomezí a hraničnost. Kolem hranice, na jejíž jedné straně je oblast spektrální hudby a na druhé lidské promlouvání, osciluje po formální stránce i skladba *Deus Cantando*. Poslechnout si tuto a také další kompozice z cyklu *Quadraturen III* lze na Ablingerově domovské stránce. [3]

4.2 | Hans-Christoph Steiner: Solitude

Hudební kompozice *Solitude* v sobě spojuje okruhy, které jsou pro dějiny hudby 20. století signifikantní. Je to např. rozbití tonality ve prospěch otevření nových prostorů pro hudební gesto. Proces zahájil Arnold Schoenberg a pokračovala v něm řada dalších svým vlastním způsobem. „Rozdrolování“ tonality je typické i pro tzv. spektrální hudbu, ve které místo „kompozice s notami“ nahrazují tekuté zvuky proměnlivé v čase a přechody mezi témbrem a harmonií. Skladatelé jako Tristan Murail a Gérard Grisey odmítli temperovaný systém jako nepřirozený, stejně jako mikrointervalové systémy či utopické snahy o „čisté” ladění.[4] Spolu s tímto odmítnutím ale bylo třeba též vytvořit nový jazyk, nový „notační systém“, jenž by zohledňoval charakter otevřivšího se kompozičního prostoru. Průkopníky nové notace se v 50. letech 20. století stali Anetis Logothetis a John Cage.[5] Grafické partitury najdeme již ale například u Luigi Russola[6] a řady dalších skladatelů jako Karlheinz Stockhausen, György Ligeti, Iannis Xenakis. Ze strany výtvarného umění v této oblasti nemůžeme nezmínit práce Milana Grygara.

Linii grafických notačních partitur následuje i kompozice *Solitude*. Domnívám se, že Hans-Christoph Steiner v ní využil jako jeden z mála dispozicí, které Pd

[3] *Deus Cantando* [online]. [cit. 2014-04-08]. Dostupné z: http://ablinger.mur.at/txt_qu3god.html.

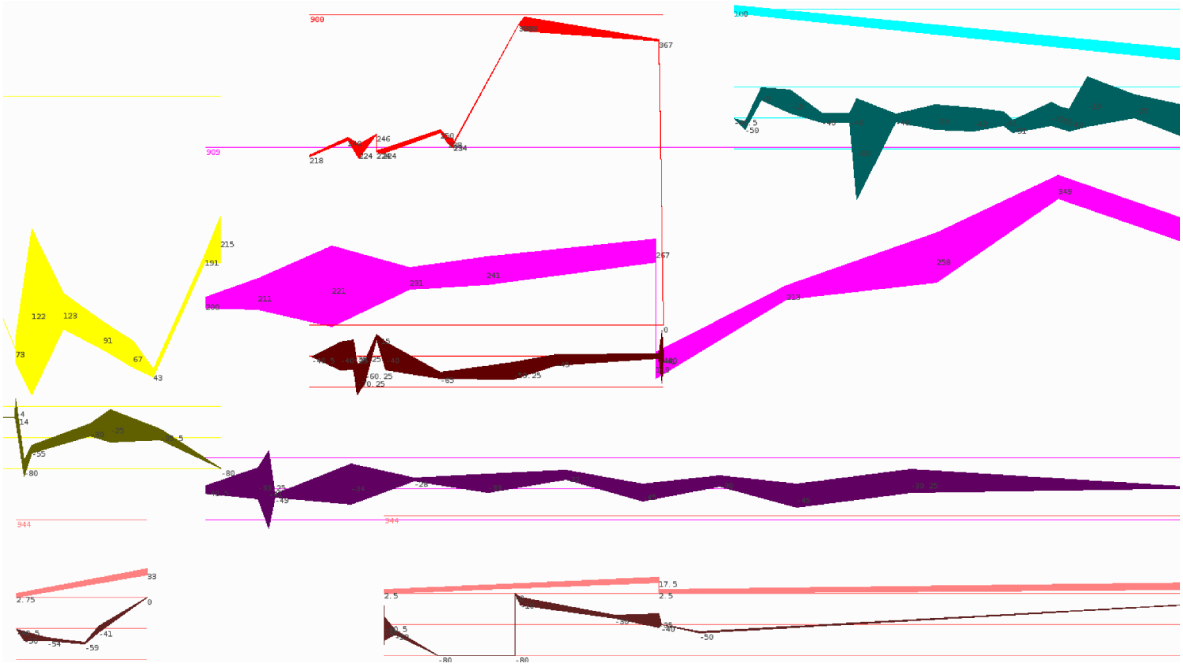
[4] PUDLÁK, Miroslav. Spektrální Hudba. *His Voice*. 2006, roč. 6, č. 5, s. 22-25. ISSN 1213-2438.

[5] KOFROŇ, Petr a SMOLKA, Martin. *Grafické partitury a koncepty*. Olomouc: Votobia, 1996. 89 s. s. 7.

[6] Za grafickou partituru lze považovat jeho *Il risveglio di una città* z roku 1914.

nabízejí ve vytváření tzv. grafických datových struktur. Přitom právě zdokonalení a zjednodušení přístupu k datovým strukturám byl motiv, se kterým Miller Puckette výslovně při programování Pd počítal.[7] Datové struktury v Pd jsou zatím ne příliš využívaným, a přesto poměrně silným nástrojem.

Steiner v *Solitude* používá osm tonálních zvukových vzorků, které si vypůjčil ze skladby *Warm Valley* od Duka Ellingtona, a ty s pomocí vlastních nástrojů sestavených v Pd „rozdroluje“ a „vynáší“ do spektra. Ve Steinerově grafické notaci odpovídá osa X času, na ose Y jsou zaznamenány hodnoty, související s hlasitostí, panoramou a „šířkou mraku vzorků“. Každý z osmi vzorků je přitom reprezentován dvěma grafickými poli – to obvykle větší a barevné řídí přehrávání daného vzorku. Pod barvou zvukových vzorků (žlutá, fialová, azurová, červená, zelená, modrá, šedá a růžová) se vždy nachází ještě jedna tmavší linka, jež popisuje hlasitost a panoramu. Sám autor se odvolává na inspiraci u Ludgera Brümmera a jeho kompozice *Le Tombeau de Maurice*[8] a u Iannise Xenakise a jeho ručně kreslené kompozice *Mycenae Alpha*.[9]



Začátek grafické partitury kompozice *Solitude*. Na začátku zní dva vzorky, žlutý a růžový.

Kompozice včetně zdrojových kódů v Pd je ke stažení na domovské stránce autora.[10] Mimo jiné jsou na ní umístěny i aktuální instalační verze Pd-extended a řada objektů ([hid], [pduino], [autoscale]), které Hans pro Pd naprogramoval.

[7] Podívejte se v Pd do oficiální Millerovy dokumentace (Menu Nápověda > Pd Help Browser > Pure Data > 4.data structures > 07.sequencer.pd), kde je ukázka techniky, kterou použil i Steiner.

[8] *Ccrma.stanford.edu* [online]. [cit. 2014-04-08]. Dostupné z: https://ccrma.stanford.edu/CCRMA/Music/CDs/VolumeTwo/Brummer.html.

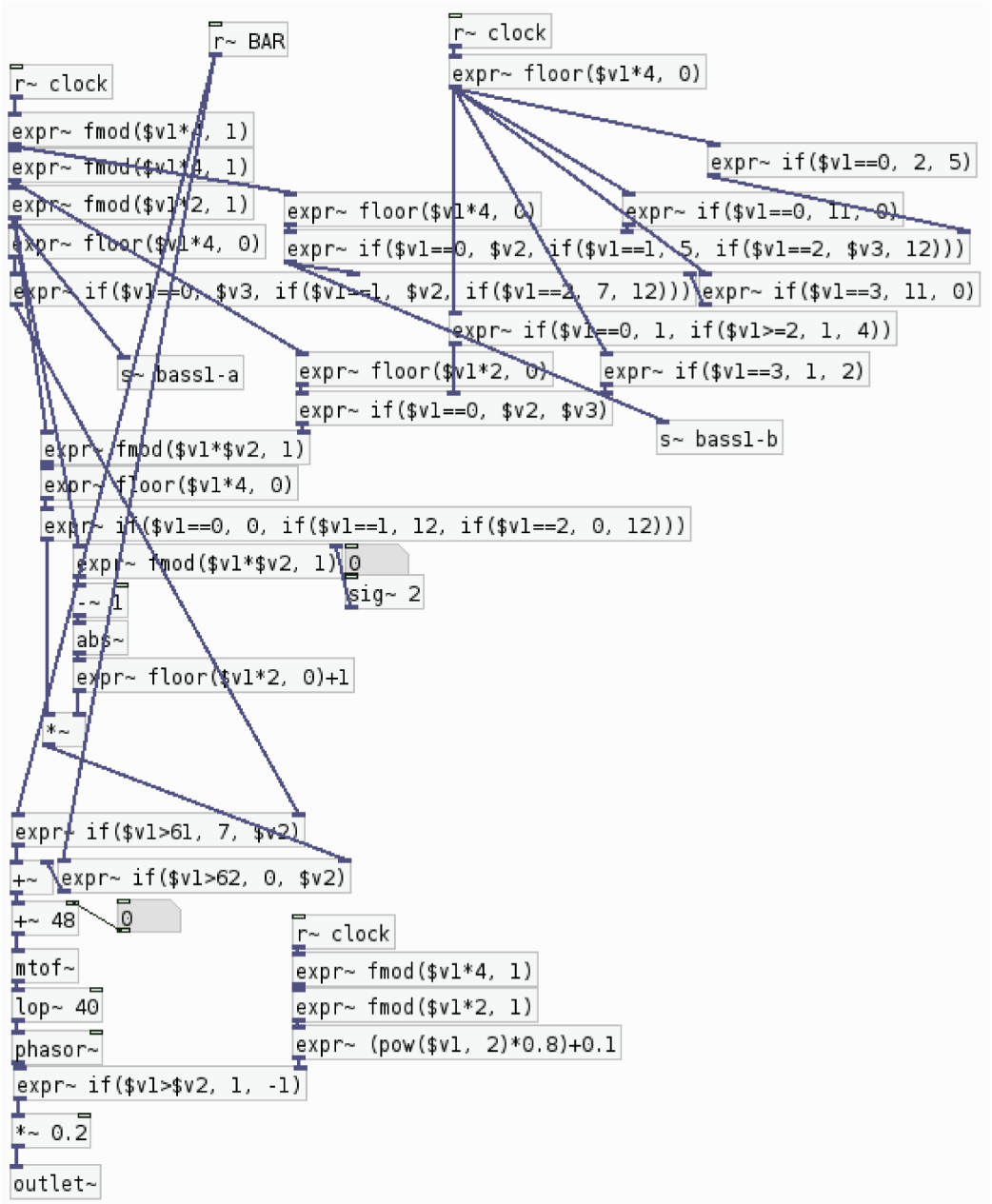
[9] Mycenae Alpha. In: *YouTube* [online]. 12. 07. 2007. [cit. 2014-03-10]. Dostupné z: http://www.youtube.com/watch?v=yztoaNakKok&feature=kp. Kanál uživatele Donald Craig.

[10] *At.or.at* [online]. [cit. 2014-03-08]. Dostupné z: http://at.or.at/hans/solitude/.

4.3 | 0xA: expr~

0xA[11] je skupina sestávající z objektů a patchů. Tři hudebníci a programátoři – Aymeric Mansoux, Chun Lee and Olivier Laruelle – jsou „pouze“ její přispěvatelé. Skupina za sebou má řadu vystoupení a vydání jedné desky, jejíž jednotlivé písně i zdrojové kódy jsou ke stažení pod licencí copyleft na webu labelu gosub10.[12]

Zvláštnost a jedinečnost této skupiny spočívá v tom, že hudbu vytváří v Pd zejména prostřednictvím objektu [expr~]. S jeho pomocí lze definovat komplexní matematické operace, které buď přímo generují nebo modulují zvukový signál. Kromě toho ale slouží [expr~] také k vytvoření partitury.



Ukázka kódu písně *0903-4 .pd*, který vytváří melodii.

[11] *Gosub10.org* [online]. [cit. 2014-03-10]. Dostupné z: http://gosub10.org/0xa.html.

[12] *Gosub10.org* [online]. [cit. 2014-03-10]. Dostupné z: http://gosub10.org/GOSUB10-004.html.

Přístup 0xA je ukázkou precizního zvládnutí programátorské instrumentace, znalosti nástroje a zároveň ukázkou poměrně vysoké míry abstrakce v zacházení s „materiálem“ a pochopení kompozice jako takové. Lze ho přirovnat ke komponování Andyho Farnella[13] (ten se ale programově neomezoval pouze na objekt [expr~]) nebo k extrémním minimalistickým kódům o 140 znacích, které vytvořila skupina hudebníků a programátorů v programovacím jazyku SuperCollider a které byly též vydány jako deska.[14] Hudební výraz 0xA je ve srovnání s kompozicemi autorů z kompilace sc140 poplatný hudebnímu žánru, jenž byl charakteristický pro počítačovou demoscénu, a není tolik rozvolněný a experimentální. Minimalismem, jednoduchostí a přitom bohatostí výrazu jsou oba počiny ale spřízněné. Dodejme, že obal desky je také generován počítačem, a to v programovacím jazyku Processing.

4.4 | Roman Haefeli: Netpd

Netpd[15] je tzv. CRNMME projekt, u jehož zrodu stál Roman Haefli a postupně se k němu přidala řada dalších programátorů. Za tajemnými písmeny se skrývá zkratka *Collaborative Realtime Networked Music Making Environment.* – jde vlastně o kolekci patchů (syntetizéry, sekvencery, efektové jednotky atd.), které umožňují uspořádání živé hudební produkce. To by samo o sobě nebylo ničím zvláštním. Netpd je ale unikátní tím, že síťově zprostředkovává komunikaci mezi nástroji jednotlivých hudebníků, takže každý zúčastněný má zároveň přístup k nástrojům a nastavením ostatních. Představte si, že hrajete na kytaru – přistoupí k vám další hráči a začnou přidávat nové prstoklady, nebo před vašima očima přepíšou partituru, kterou jste před chvílí napsali. K jedné kytaře se vejde nanejvýš kolem šesti hráčů – v Netpd jich můžou u jednoho syntetizéru nebo partitury „stát“ desítky. Projekt Netpd přinesl do způsobu „jak dělat hudbu na počítači“ to, co je běžné v jakékoliv jiné hudební jam-session, ale na jiné úrovni. V rámci dění na elektronické hudební scéně lze Netpd jistě vidět v souvislosti se síťovými performancemi hudebního tělesa The Hub a výzkumem Tima Perkise.[16] Rozdíl je ale v tom, že tím, co bylo pro The Hub MIDI rozhraní, je pro Netpd internet.

Historie Netpd sahá do roku 2004, kdy vznikla první testovací verze. V roce 2008 začal Haefli kód Netpd kompletně revidovat, odstranil řadu chyb a převe-

[13] *Obiwannabe.co.uk* [online]. [cit. 2014-03-10]. Dostupné z: http://obiwannabe.co.uk/html/compositions/compositions.html.

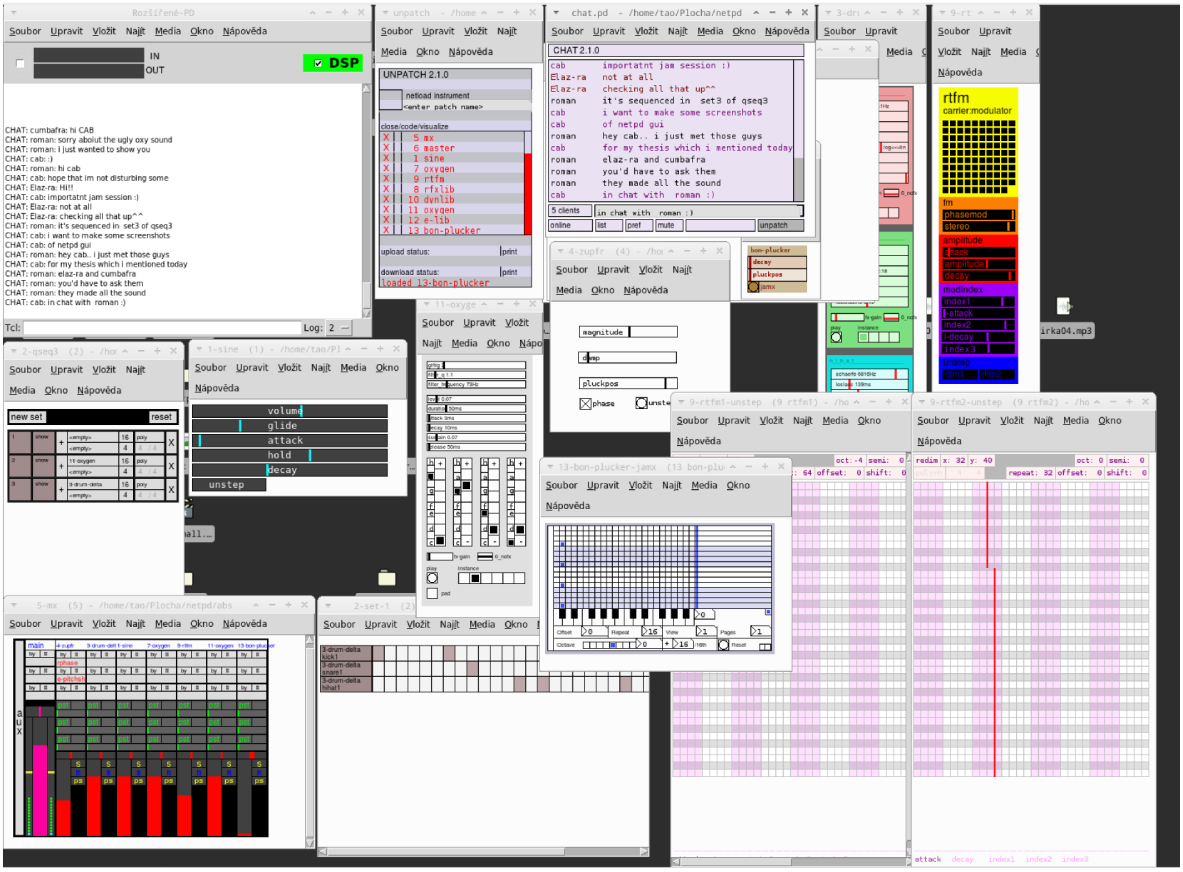
[14] *Supercollider.sourceforge.net* [online]. [cit. 2014-03-10]. Dostupné z: http://supercollider.sourceforge.net/sc140/.

[15] *Netpd.org* [online]. [cit. 2014-03-15]. Dostupné z: http://www.netpd.org/.

[16] *Perkis.com* [online]. [cit. 2014-03-15]. Dostupné z: http://www.perkis.com/_site/writings/w_hubem.html.

dl síťový provoz na protokol OSC. Revidovaná verze je ke stažení na domovské stránce projektu od roku 2012. Kolem domovské stránky Netpd se utvořila také malá komunita stálých návštěvníků hudebních sezení, která probíhají obvykle každé úterý. Po doinstalování Netpd patchů do Pd se „jamu“ může kdokoliv zúčastnit.

Podobným projektem jako Netpd je PPooll[17] pro Max/MSP. Komunita kolem kolaborativně síťových hudebních projektů není nijak velká ani živá. Přesto se ale zdá, že potenciál koncepce společného vytváření hudby „pochopily“ i společnosti zabývající se vývojem „zábavného software“, a tyto tendence začínají pomalu zapouštět kořeny i v komerční sféře.[18]



Ukázka patchů z kolekce Netpd: sekvencery, mix, syntetizéry, chatovací okno.

Netpd zajišťují synchronizaci nástrojů a událostí – mezi počítači se přitom nepřenáší zvukové streamy, ale pouze řídící data pro jednotlivé hudební nástroje. Veškerý zvuk je syntetizován na tom kterém počítači, takže synchronizovat jednotlivé klienty lze i v sítích s pomalejším připojením.

[17] *Ppooll.klingt.org* [online]. [cit. 2014-03-15]. Dostupné z: http://ppooll.klingt.org/.
[18] *Bitwig.com* [online]. [cit. 2014-03-15]. Dostupné z: https://www.bitwig.com.

4.5 | Michal Cáb: Ty & Ra, Událost I., II.

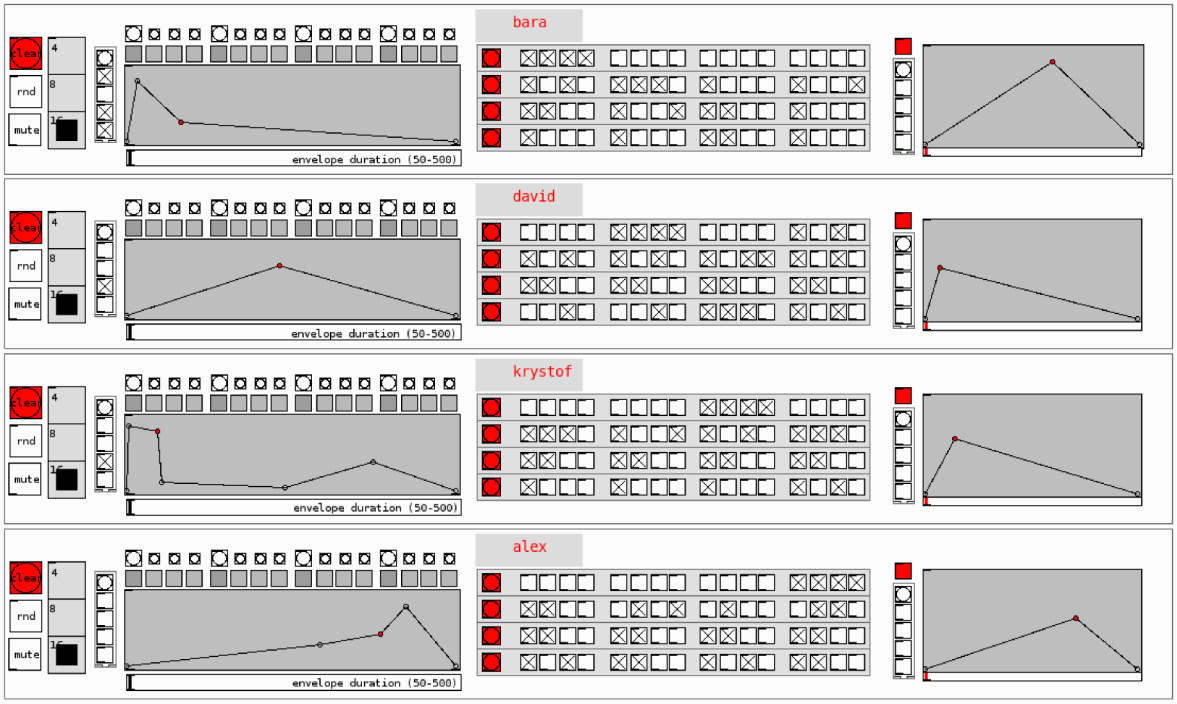
S Pd jsem se seznámil někdy kolem roku 2004. Tomuto setkání ale předcházelo několik let, během kterých jsem hledal způsob, jak se ve své hudební tvorbě vymanit ze stereotypních postupů – chtěl jsem opustit dlouhé harmonické plochy ambientních kompozic směrem k neharmonickým zvukovým fragmentům. Pojmy a jména jako „konkrétní hudba", noise, Luigi Russolo, John Cage atd. pro mne byly naprosto neznámé. Rok 2002/03 jsem trávil studijně ve vlámském Leuven, kde jsem často navštěvoval jeden obchod s deskami – tam jsem poprvé uslyšel hudbu od Auterche, Aphex Twina, Teamu Doyobi a řady dalších autorů, které by v té době bylo možné zařadit mezi elektronickou avantgardu. Poslech těchto nahrávek byl pro mě zkušeností s ještě neslyšeným – Auterche a Aphex Twin přišli s až chirurgickým přístupem k zvukovému materiálu. Na druhou stranu jsem sledoval dění na ukrajinské experimentální scéně, kterou v té době reprezentoval Andrej Kiritchenko a jeho label Nexsound. V rozhovoru s Andrejem[19] jsem pak poprvé narazil na Kima Casconeho a jeho esej o postdigitálním umění, ve kterém byly zmíněny i softwarové nástroje jako Reaktor, Max/MSP a Pd. Intuitivně jsem cítil, že posun v tom, jak „dělat hudbu", pro mě souvisí právě s posunem k těmto nástrojům. Průzkum nových oblastí mohl začít, první experimenty a pokusy s Pd mě ale dokonale odradily, protože se mi nepodařilo z nich „dostat" vůbec nic. Fakticky jsem se k hlubšímu studiu Pd dostal až díky návštěvě Letných dielní v Bratislavě, které organizovala Mária Rišková, Magdalena Kobzová, Dušan Barók a další. Letné dielne[20] byly platformou, na níž se setkávali nejrůznější lidé – umělci, teoretici, inženýři, hackeři, i „obyčejní" lidé se zájmem o daná témata – v živé mezioborové diskuzi i v rámci prakticky orientovaných dílen. Jedna z dílen se týkala právě i Pure Dat.

Pd svým způsobem kladou na svého uživatele nároky, v kontaktu nimi se obvykle nutnosti studia nevyhnete. Pro mě bylo toto studium, poté co jsem na Letných dielňach překonal „fázi odporu", veskrze dobrodružnou zkušeností. Pd byla prostředkem, díky kterému jsem se blíž začal věnovat technicistní a psychoakustické stránce zvuku. Zároveň s tímto studiem jsem se začal více zajímat i o dějiny intermediálního umění a experimentální hudby. Jinými slovy: rozšířil jsem si mírně horizont možností a s tím i míru frustrace.[21]

První „úspěšné" kódy, které jsem v Pd vytvořil, byly nejrůznější ruchové generátory. V počátcích jsem pracoval nepoučeně a intuitivně, tzn. že jsem zvukový

[19] *Dreamface.net* [online]. [cit. 2014-03-12]. Dostupné z: http://www.dreamface.net/modules.php?name=News&file=article&sid=1215.
[20] *34.sk* [online]. [cit. 2014-03-12]. Dostupné z: http://www.34.sk/text.php?text=3-175.
[21] Viz s. 36.

signál zkoušel tvarovat „metodou“ pokus/omyl, nebo jsem remixoval kódy zkušenějších programátorů. Výsledky byly někdy překvapivé a z tohoto období mám ve sbírce několik patchů. S postupem času jsem si sestavil nástrojový repozitář z nejrůznějších syntetizérů i samplerů, které dodnes při živém hraní používám a průběžně je obměňuji a modifikuji. Uplatnění tyto patche našly v živých improvizacích s Peterem Gondou a Markétou Cilečkovou, P.O.P.Corpem, v expanded-cinema performancích se skupinou Mikroloops, ale také v divadelních vystoupeních, pro která jsem zajišťoval zvukový design. Výsledkem „průzkumu“ na vizuálním poli byl videomixer a efektová jednotka pro Kláru Doležálkovou, které využila pro postprodukci některých svých prací[22] i při živých vystoupeních.[23]



Ukázka GUI projektu Ty & Ra: 4 sekvencery pro 4 signály od ladičů rádií.

Nyní bych se rád věnoval popisu tří konkrétních prací. Tou první je specifický hudební nástroj pro skupinu Ty & Ra, která vznikla v rámci dílen na Institutu intermédií.[24] Výchozím zvukovým materiálem v hudební improvizaci jsou pro Ty & Ra procházky napříč pásmy rádiových vln. Čtveřice ladičů hledá v pásmech brumy, šumy, šelesty, ale i mluvené slovo a reprodukovanou hudbu. Zvukový signál je pak od jednotlivých ladičů přiveden do vícekanálové zvukové karty a dále zpracováván nástrojem sestrojeným v Pd, jehož funkce spočívá v sekvencování,

[22] Plop. In: *YouTube* [online]. 12. 03. 2009. [cit. 2014-03-13]. Dostupné z: http://www.youtube.com/watch?v=qgmz0UorhtU. Kanál uživatele Melodyha Klar.

[23] NAPALMED & VJ Melodyha. In: *YouTube* [online]. 18. 02. 2011. [cit. 2014-04-08]. Dostupné z: https://www.youtube.com/watch?v=fRlDGdqywgw.

[24] *Iim.cz* [online]. [cit. 2014-03-13]. Dostupné z: http://www.iim.cz. Spoluzakládající členové byli Michal Kindernay, Alex Moralesová, Bára Švarcová, Viola Ježková a Magdalena Kobzová. Později se v pozici ladičů rádií ještě vystřídali Kryštof Pešek a David Šmitmajer.

promíchávání a organizaci jednotlivých zvukových nálezů v pásmech. Se signálem se pracuje v reálném čase. Software pro Ty & Ra je určen pro dva hráče se dvěma oddělenými počítači – patche jsou ale propojeny sítí a uzpůsobeny tak, že jakákoliv změna v jednom grafickém uživatelském rozhraní nástroje se projeví bezprostředně i ve druhém. Hráči si tak navzájem mohou zasahovat do svých nastavení. Je zde patrná spřízněnost s projektem *Netpd*, i když Ty & Ra nedosahuje ani zdaleka takové kompexnosti a technické vyspělosti, protože ji ve své jednoduchosti nepotřebuje. Ve škále stylů se Ty & Ra pohybuje někde mezi ambientem, noisem a minimal technem.[25]

Další, tentokrát sólový projekt, je postaven na audiovizuálním nástroji, který se původně jmenoval n0|$3-f|l+3r.[26] Posléze jsem ale performance s tímto nástrojem přejmenoval na Událost I., protože pro mě v „opačné polorovině" souvisí s algoritmickou kompozicí Událost II. Na vzniku Události I. ilustrovat „měkký" způsob práce s Pd. Kód totiž vznikl velmi intuitivně a v podstatě jako náhodný výsledek studia při prozkoumávání externí knihovny *pdp* a *pidip*, jež slouží ke zpracování videosignálu. Neměl jsem zrovna k dispozici kameru, takže jsem využil objektu [pdp_noise] ke generování vizuálního šumu a na něm jsem testoval základní možnosti knihoven. Tok šumu jsem s jejich pomocí „rozmazával", „zaostřoval", „zachytával" a „popouštěl". Výsledkem jsem byl překvapen. Později jsem se k náhodně vznikému kódu vrátil a doprogramoval složky umožňující ovládání parametrů MIDI kontrolerem a analýzu celkové světelnosti obrazu, která se převádí na zvukový signál. Tím byl „vývoj" audiovizuálního nástroje hotov. Úplně jinou kapitolou pak ale bylo učení se na tento nástroj hrát.

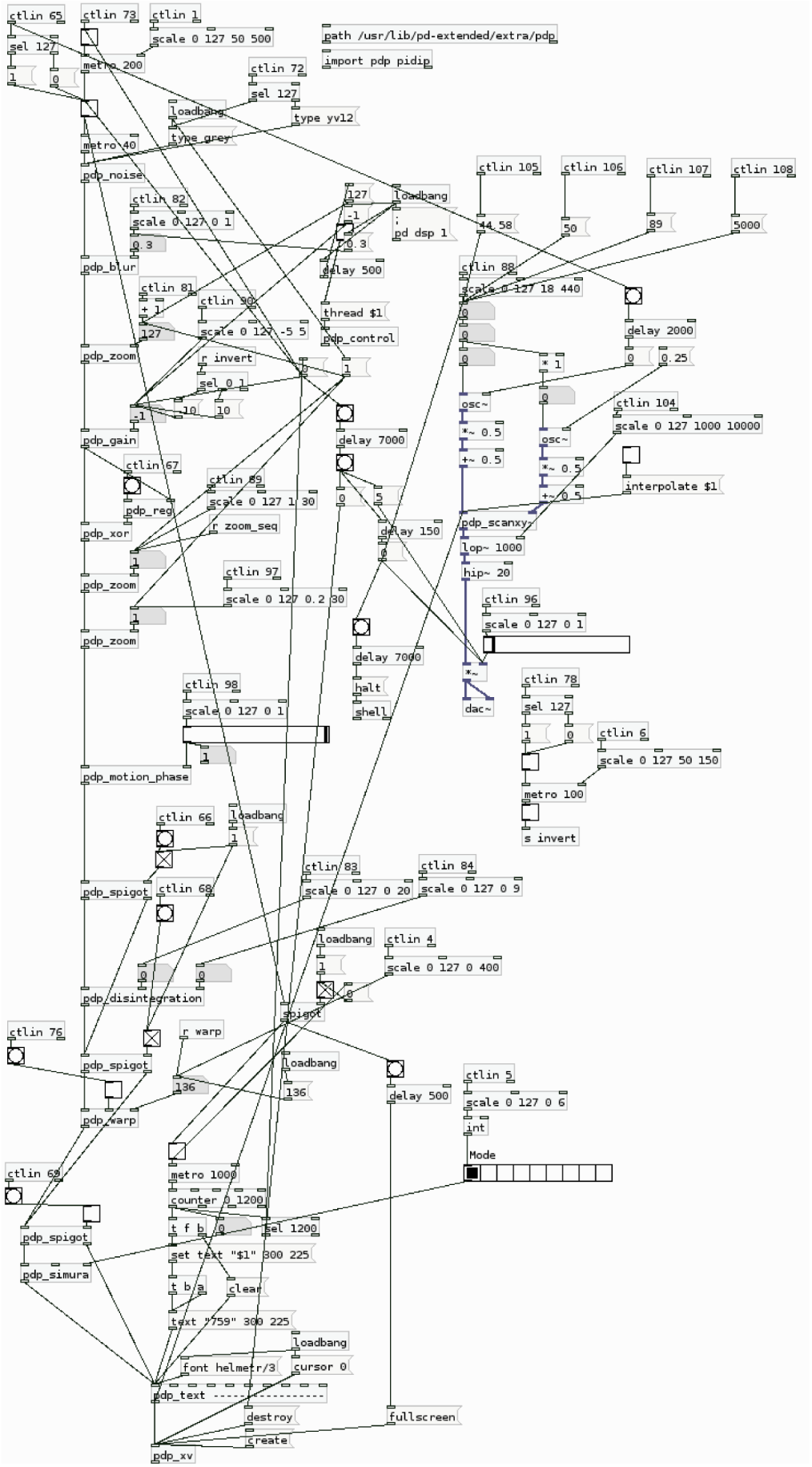
Spíše než o hru jde o zápas s instrumentem. Základem, z něhož v Události I. vyvěrá obraz i zvuk, je totiž zmíněný šum, který performerovi neustále podsouvá nové impulzy a výzvy k jeho „zkrocení". Jedním z utopistických motivů v Události I. je snaha o překročení běžné intenzity audiovizuálních podnětů přítomných v dnešní hypermedializované společnosti za účelem vytvoření nového prostoru, v němž by se ucho a oko mohlo svobodněji pohybovat. Za specifikum tohoto nástroje považuji to, že referenčním prvkem při „hraní" zde není jeho GUI, kterou by performer sledoval na monitoru, ale přímo výsledná projekce. Krátkou ukázku lze shlédnout na portálu artycok.tv.[27]

---

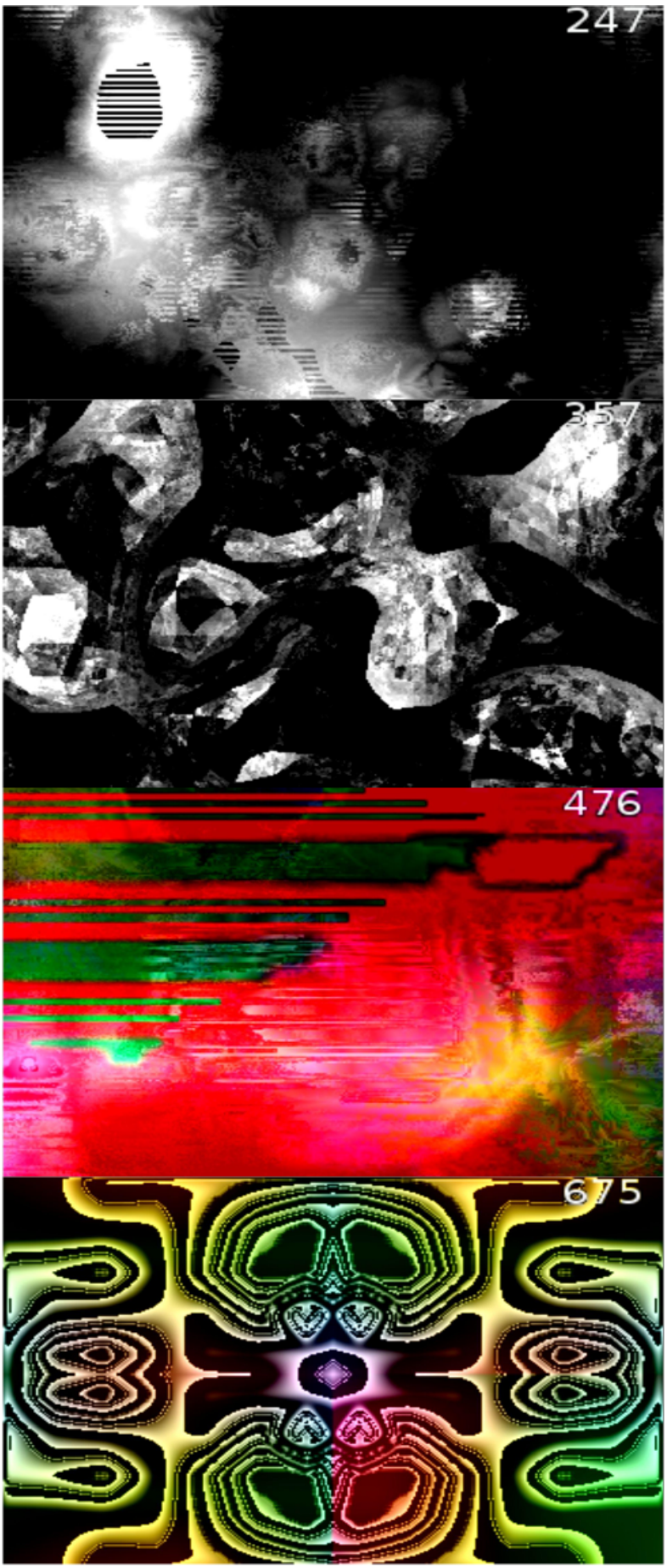
[25]S Ty & Ra jsme vystoupili v několika galeriích a klubech a také v rámci akce Narozenin umění v roce 2011 (v Česku se slaví od roku 2005), kterou spoluorganizoval Český rozhlas. *Rozhlas.cz* [online]. [cit. 2014-03-13]. Dostupné z: http://www.rozhlas.cz/artsbirthday/2011.

[26]BLAŽÍČEK, Martin. Slovo autorství je mi trochu nepříjemné. *Cinepur*. 2011. č. 73. ISSN 1213-516X.

[27]*Artycok.tv* [online]. [cit. 2014-03-16]. Dostupné z: http://www.artycok.tv/lang/en-us/8130/cab-michal. Zkrácený záznam performance 9:00 až 12:27.

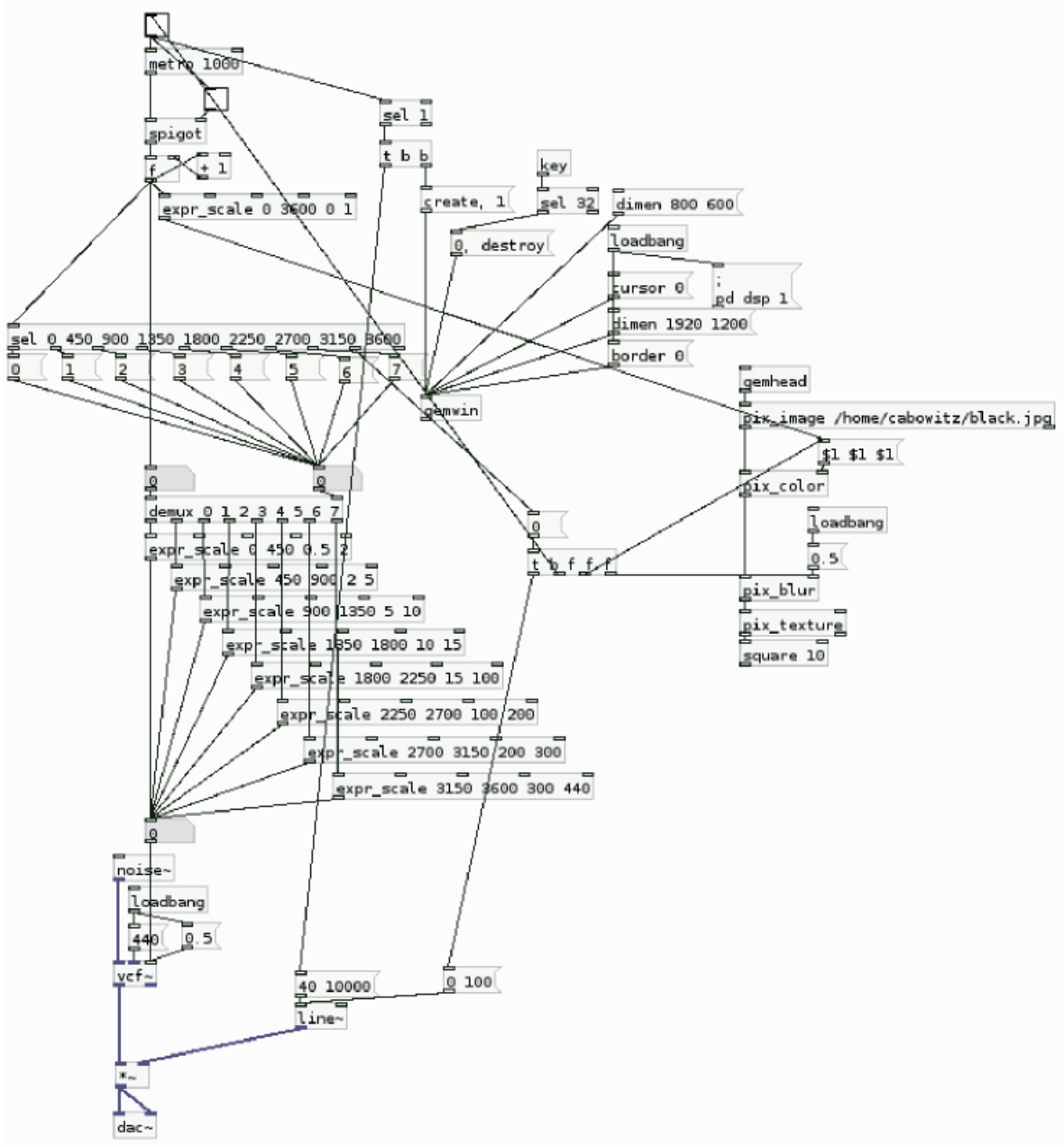


Patch tvořící jádro performance Událost I.



Ukázky vizuálního výstupu performance Událost I.

Událost II., na rozdíl od předchozích ukázek, není nástrojem, ale algoritmickou kompozicí. Okolnosti jejího vzniku byly též jiné – nevznikla totiž „náhodou“, ale velmi řízeně. Samotné programování a odladění kódu trvalo přibližně půl hodiny a šlo v podstatě jen o „zhmotnění pojmu“, který zde byl dříve a nezávisle na kódu. Událost II. je spřízněným protipólem Události I. – také se pokouší v percepci jít za hranici, ale zcela jiným způsobem.



Kód algoritmické kompozice Událost II.

Celá audiovizuální kompozice trvá jednu hodinu a – formálně řečeno – se v ní stane „pouze“ to, že se černá projekční plocha promění v bílou a z bílého šumu se s pomocí pásmové propusti vyfiltruje tón komorního A. Jestliže se Událost I. snažila překročit percepční hranici rychlostí a intenzitou podnětů, pak Událost II. se pokouší jít za ni pomalostí a „absencí“ dění. Při první projekci v galerii AM180 jsem sice věděl, jaký bude průběh kompozice. Jaká bude ale odezva a zkušenost

po absolvování celé události, jsem neměl ponětí. Subjektivně mě překvapilo postupné prolínání stavů: fyzicky nepříjemné, nudné, překvapující, sebeprojekční, tiché atd. Reakce všech zúčastněných se dost lišily, obecně by bylo ale možno říct, že pokud recipient přijme „způsob vyprávění“ této algoritmické kompozice, pak se pro něj stane druhem projekční plochy, do níž promítá každý sám sebe.

Nevyužiji zde totalizující možnosti interpretace díla autorem, ani nebudu poukazovat na asociace k spřízněným dílům. Jediné „mantinely“, které bych čtenáři v interpretaci předchozích dvou prací dal, by byly vyznačeny následujícími citacemi:

> Není žádný skutečný rozdíl mezi šumem a signálem, ten existuje pouze v záměru. Podobně je v elektronické hudbě rozdíl mezi šumem a zvukem daný jejím skladatelem, který svému publiku nabízí směs zvuků k interpretaci. Ale zaměřuje-li se na maximální neorganizovanost a maximální míru informace, bude muset obětovat něco ze své volnosti a uvést do své práce něco z řádu, který pomůže posluchači najít cestu skrze šum, který bude automaticky interpretovat jako signál, neboť ví, že byl vybrán a do určité míry organizován.[28]

> Žádné nevinné oko neexistuje. Oko vždy přistupuje k dílu poznamenané, posedlé vlastní minulostí a tím, co mu v době dávné i nedávné podstrčily uši, nos, jazyk, prsty, srdce a mozek. Nefunguje jako samostatný a soběstačný nástroj, nýbrž jako poslušná součást komplexního a rozmarného organismu. Nejen jak, ale i co oko vidí, se řídí potřebami a předsudky. Vybírá si, odmítá, pořádá, rozlišuje, spojuje, třídí, analyzuje, tvoří. Spíše než aby zrcadlilo, uchopuje a vytváří: své výtvory nevidí holé, jako položky bez vlastností, ale jako konkrétní věci, jako potravu, jako lidi, jako nepřátele, jako hvězdy, jako zbraně. Nic nemůže vidět nahé ani naze.[29]

4.6 | Michal Kindernay: Art of pollution

Vizuální programování v mé tvorbě znamenalo zásadní zlom a iniciovalo svobodné experimentování se zvukem a obrazem. Především jsem se odklonil od linearity. Zaujalo mě užití různých zdrojů signálů, jejich propojení a aplikace, práce se zvukem v úzkém spojení s dalšími vstupy či volnost generování obrazů. Protože mne vždy fascinovaly světy mikro i makro kosmické, reflektuji ve

[28] ECO, Umberto. *Otevřené dílo* [online]. [cit. 2014-05-10]. Dostupné z: http://cirkus-umberto.ic.cz/OperaAperta.pdf. s. 77.

[29] GOODMAN, Nelson. *Jazyky umění*. Praha: Academia, 2007. 212 s. ISBN 978-80-200-1519-8.

své práci jejich existenci a naše počínání, případně vedlejší účinky a okolnosti. Znečištění, které je přirozené a může být příčinou zrodu alternativních evolucí, nicméně reflektuji kriticky a s důrazem na odchylku od relativního bodu klidu k chaosu, což zejména v globálním měřítku získává sociální a ekologické aspekty.

Ukázka z *Noise Windows*.

Na reflexe bezprostředního prostředí reaguji v audiovizuálním překladu. Například studie různých forem znečištění a jejich vizualizace a sociální kontexty. *Noise Windows* je jedna z prvních vizualizací zvukového znečištění, které bez ustání naráží na okna obyvatel města. Rušná dopravní situace, obzvlášť v centru a podél výpadových komunikací, celkově zvyšují zvukový práh města. Navštěvoval jsem byty mých přátel a analyzoval hluk ulic. Kamera snímala výhled z okna na rušné či klidné ulice a hluk aut se zapisoval do obrazu ve formě červených vertikálních zářezů. Princip horizontálního postupného zápisu v obraze jsem použil i v dalších projektech. Je to jeden z ideálních způsobů časově progresivní vizualizace, protože zanechává v obraze detailní stopu. Dekonstruovaný obraz zachycuje určitou audiovizuální paměť prostoru přeloženou do obrazového dvourozměrného pásu. V prostředí Pure Data jsem pracoval s velmi úzkým výřezem reálného obrazu z kamery, který jednoduchým způsobem postupně přičítal hodnoty decibelů do generovaného obrazu. Po shlédnutí výsledných obrazů je jasné, kdy

byla ulice relativně rušná a kdy klidnější. Spolu s hlučnými zářezy je totiž obraz samplován jen v určitém čase a místě obrazu, tj. podobně jako v technice „slit-scanu“. Projekt tedy apeluje na zvukově klidnější prostředí měst. Na možnost otevřených oken do výfukovými plyny nezamořených ulic. Sociálním rozměrem byly diskuze nad dopravní a zvukovou situací města při dlouhých návštěvách u přátel, zatímco se generovaly zvukové obrazy za jejich okny.

Projekt *Pollution Movies*[30] byl komplexnější v rozsahu použitých senzorů. Nejen zvuk, ale i teplota, vlhkost a zejména kvalita ovzduší byly analyzovány a zapisovány v reálném čase do obrazu. Protože výstupem měl být „pollution“ film. Snímkování či zápis dat reflektující znečištění ve vrstvách obrazu byl urychlen a vzniklé obrazy uloženy do video sekvencí. V tomto případě vidíme proměny v ovzduší oku jinak neviditelné. Struktury často jemných neviditelných dějů generují „časosběrný“ materiál. Zajímalo mě rovněž, do jaké míry vnímáme znečištění v obraze (tj. šum, nereálné barvy, vypadlé snímky či nejrůznější glitche). Analogové „hand painted“ techniky užívané v experimentálních filmech, které strukturují filmový materiál, v tomto případě střídají atmosferická data. Samo prostředí se otiskuje do obrazu často v abstraktních obrazových „kobercích“. V tomto projektu jsem nicméně použil ekvivalentu Pure Data a sice Max/MSP, který mi umožňoval pracovat velmi detailně a efektivně s „matrixem“ obrazu.

Rád vytvářím své vlastní nástroje. „Větrná kamera“ *Wind Box*[31] je vybavena senzorem a větru užívá jako základního spouštěcího mechanismu při tvorbě obrazů. Je to již druhá verze. První byla podstatně větší díky DIY anemometru. Větrné kamery patří do rodiny dalších nástrojů, jako například autonomní a podstatně komplexnější Camera Altera. Větrná kamera užívá pouze větru jako fundamentálního hybatele. Metaforou je pomyslná klika ruční kamery raného filmového kameramana. Podle toho, jak rychle s ní točil, byl materiál mírně zpomalený či zrychlený. Dalším aspektem je, že v případě větrné kamery všechno zůstává uloženo v krabici. Všechno, co je zachyceno, je také uchováno uvnitř skříňky a malým otvorem je to možné vidět. Může to znít pateticky, ale myslím, že kamera točí docela v jiném přírodním rytmu, jiném než najdete u konzumních kamer. Vítr ovlivňuje počet snímků za sekundu a jejich vzájemné prolínání. A když není vítr, proces natáčení se zastaví a obrazy ustupují do pozadí. Je to černá skříňka. Nikdy nevíte, co dělá, jestli nahrává či uvnitř promítá.

[30] Art Pollution Movies. In: *Vimeo* [online]. [cit. 2014-04-08]. Dostupné z: http://vimeo.com/16186696. Kanál uživatele Insectual_mkin.

[31] Wind*cam – test I. In: *Vimeo* [online]. [cit. 2014-04-08]. Dostupné z: http://vimeo.com/15963085. Kanál uživatele Insectual_mkin.

Projekt *Angular Landscapes*[32] jsem realizoval společně se Signe Lidén v rámci festivalu Piksel v norském Bergenu. Festival se zaměřuje na podporu *open-source* projektů. Signe Lidén je zvuková umělkyně a často pracuje se sonifikací objektů. Předpokladem spolupráce bylo synestetické propojení zvukové a vizuální složky. Dlouhé hodiny ve vlacích „levitujících" v krajině můžou navozovat stavy hypnotické povahy. Inspirovali jsme se abstrahujícími obrazy za okny, zvukem motorů, větru a konstrukce vlaku a dráhy v cyklických strukturách. Vytvořili jsme několik velkých rezonujících objektů z recyklovaného materiálu a experimentovali v prostoru galerie, kde se rychle proměňovaly světelné podmínky a teplota. Jednoduchý systém snímající zpětnou vazbu kontaktními mikrofony a cívkami generovaný signál rozeznával různě rezonující objekty. Ty byly propojeny dlouhou nataženou pružinou, která mezi nimi přenášela vibrace. Systém senzorů analyzující zvuk, teplotu, vlhkost a světlo ovlivňoval množství signálu napojeného do objektů. Instalace tak interagovala s prostředím galerie, se světlem, který do ní pronikal skrze velká boční okna a s teplotou díky výrazným proměnám během dne a noci. Senzory také modifikovaly dvě projekce. Data se zapisovala do dlouhých obrazových pásů a výrazně modifikovala dvě video sekvence – pohledy z vlaku. Pásy se pomalu generovaly a pohybovaly dle povahy zvuku v místnosti. Interakce s příchozími návštěvníky započala již otevřením dveří do chladné ulice zimního Bergenu. Instalace byla navržena jako uzavřený systém s proměnlivými hodnotami galerijního klimatu. Festival Piksel je proslulý svou angažovaností a *open-source* kulturou. Projekty, instalace, dílny či performance, často interaktivní povahy, musí být bez výjimky programované v otevřeném software. Pure Data patří mezi nejpoužívanější software mnoha intermediálních umělců a instalace na tomto festivalu dokazují, jak důležitou roli hrají v současném umění.

V neposlední řadě zmiňme projekt komplexního senzorického nástroje *Camera Altera*[33], který pořizuje audiovizuální deníkové záznamy bez možnosti jakéhokoliv ovládání. Samotné prostředí, ve kterém se kamera nachází, určuje strukturu natočeného materiálu a reflektuje tak svého uživatele. Kamera je vybavena množstvím senzorů, které analyzují prostředí a dle statistické analýzy určují, kdy kamera pořizuje audiovizuální záznam. Kamera navíc data analyzuje se zpětnou vazbou, dá se říct, že se učí, pamatuje si, v jakých prostředích se v minulosti nacházela, a tím neustále upravuje svůj režim. Pouze jediným tlačítkem je možné zasáhnout do procesu pořizování záznamů (například, když máme pocit, že nastal ten magický okamžik, který musíme zachytit). Nicméně kamera režim zazna-

[32] Angular_Momentum. In: *Vimeo* [online]. [cit. 2014-04-08]. Dostupné z: http://vimeo.com/17500220. Kanál uživatele Insectual_mkin.

[33] *Yo-yo-yo.org* [online]. [cit. 21. 3. 2014]. Dostupné z: http://www.yo-yo-yo.org/camera_altera.html.

menávání nespustí, pouze pozmění své nastavení, pravidla, situaci. Pro technické řešení jsme nakonec s programátorem použili skriptů a speciálních knihoven v linuxovém prostředí. Pure Data nicméně sloužila v první testovací fázi a pro první vizualizace dat v obraze. Testovací fáze zahrnovala hlavně kalibraci senzorů, dlouhodobé zápisy dat a jejich simulace a postprodukci.Camera Altera je neustále se vyvíjející projekt. Především je nástrojem osobním. Uživatel/aplikant si k nástroji vytváří specifický vztah. CA se pak stává součástí jeho prostředí. Samotný význam CA se tedy proměňuje. Útržkovité nahrávky dne, uložené v „magické skříňce“, získávají další rozměr. Ideálně CA večer promítá malým otvorem či našeptává večerním ambientním vysíláním. Na samotném uživateli primárně záleží, jakým způsobem se CA chová a jaký kreativní potenciál a význam získává.

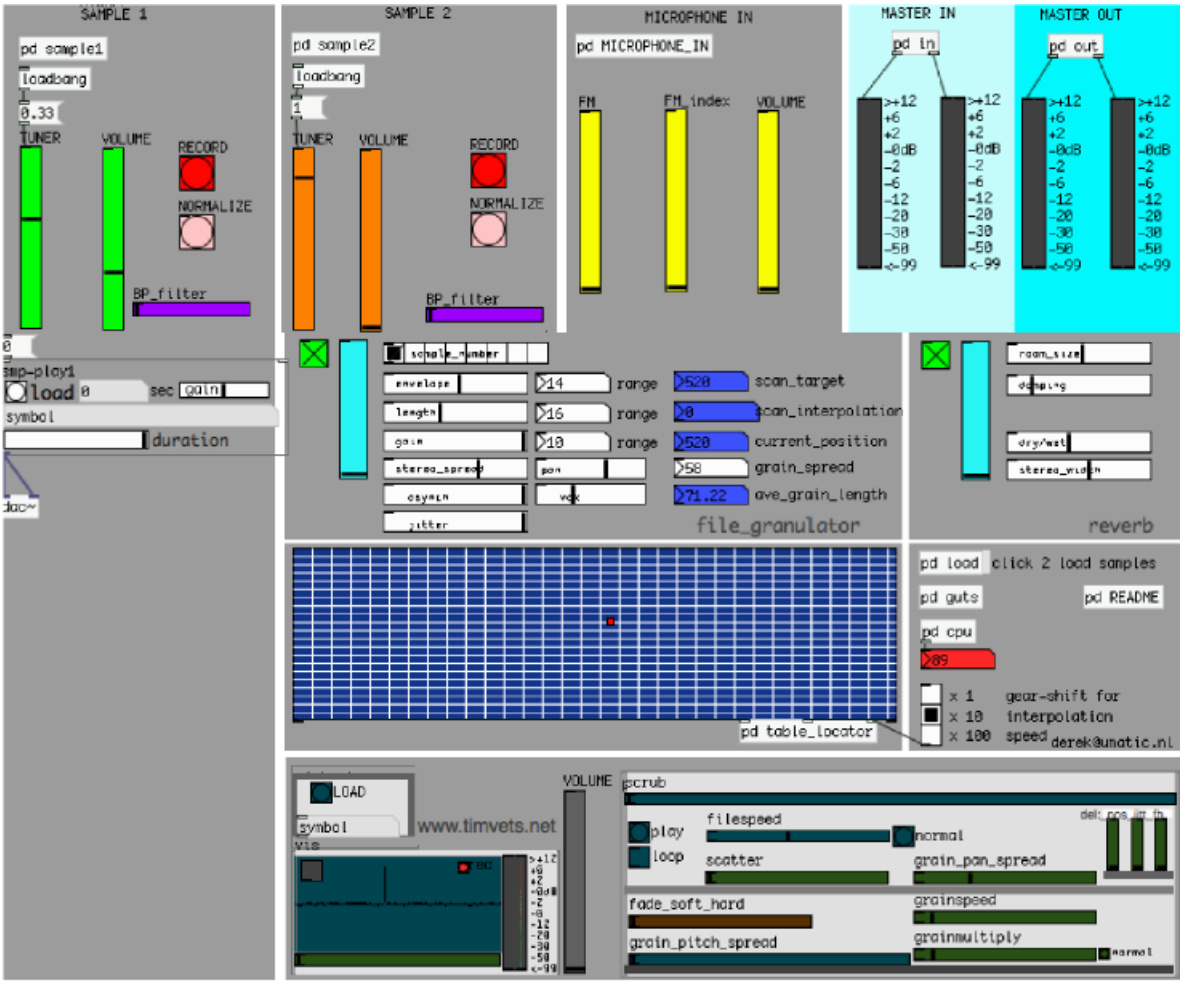
4.7 | Tomáš Šenkyřík: 001_start.pd

Od roku 2008 využívám pouze jeden patch nazvaný *001_start.pd*, který je sestaven z několika dalších patchů. Na hlubší a systematičtější programování jsem v roce 2007, tedy rok po mém osobním „objevení“ programovacího prostředí Pure Data, rezignoval. Uvědomil jsem si, že nejsem programátor, že nemám dostatečné teoretické znalosti i nutné programátorské nadání. Po pochopení základních principů, jak Pure Data fungují, a naprogramování základních patchů nebo jejich menších modifikací z patchů již připravených jsem se rozhodl připravit si základní setup, který by mi co nejvíce vyhovoval.

Sestavil jsem jej z patchů, které jsem získal především na diskusním fóru Pure Data, a také na základě patchů, které byly uveřejněny na stránkách vyuka.avu.cz. Zde se jednalo o patche, které sloužily jako studijní podklad k workshopům, které vedl Michal Cáb.

Se zmíněným patchem *001_start.pd* hudebně pracuji ve dvou základních oblastech. V prvé řadě při živém hraní, a to jak sólovém, tak společném v duu, především s cimbálem. V řadě druhé jej pravidelně využívám ve svém studiu v rámci „domácích sessions“. Všechny zvukové události, které v rámci *001_start.pd* vytvářím, nahrávám do jednotlivých stereo či mono stop, s nimiž následně pracuji v multitracku. Vybírám nejzajímavější a nejinspirativnější místa, vytvářím z nich smyčky, nebo je nadále zvukově upravuji. Výše zmíněný soubor patchů se skládá z těchto sekcí:

Sample 1 je patch pro nahrávání krátkého zvukového vzorku (délku vzorku upravuji dle hudebních záměrů), rozšířen o band-pass filtr, možnosti změny ladění, hlasitosti a okamžité normalizace. Sample 2 má totožnou architekturu jako Sample 1.



Grafické uživatelské rozhraní patche 001_start.pd.

Patch Microphone IN nejčastěji využívám pro akustický nástroj. Základem tohoto patche je objekt [fiddle]. Patch je rozšířen o FM modulaci. Výstupní signál nejčastěji vedu do hardwarového řetězce v tomto pořadí: Moogerfooger, Boss Harmonizer, Delay Wasabi.

O existenci Particlechemberu jsem se dověděl ze stránek www.puredata.info, v sekci Forum (patches). Jedná se o 32hlasý granulární syntetizéru pro manipulaci se zvukovými vzorky. Zde využívám zejména předem připravené krátké zvukové soubory. Další patch Grains, který využívám v rámci svých performancí je rovněž postaven na granulární syntéze. Objevil jsem jej na stránce Tima Vetse. Je také podstatnou součástí mého setupu, který pro živé hraní využívám. Tento patch jsem také mnohokrát využil jako základní zvukový zdroj ve svých elektroakustických skladbách[34] – jmenovitě jde např. o kompozici *Hallo Laurie*[35], která vyšla na kompilaci *Znění z mezí*[36] nebo skladbu *Hallo Steve*.[37]

[34] *Rozhlas.cz* [online]. [cit. 21. 3. 2014]. Dostupné z: http://www.rozhlas.cz/radiocustica/osoby/_zprava/senkyrik-tomas-bio--479176.

[35] *Soundcloud.com* [online]. [cit. 21. 3. 2014]. Dostupné z: http://soundcloud.com/tomassenkyrik/hello-laurie.

[36] Znění z mezí. HIS Voice. 2009, roč. 9, č. 6. ISSN 1213-2438.

[37] Guitar and Pure Data performance. In: *YouTube* [online]. 11. 11. 2013. [cit. 2014-03-21] Do-

4.8 | Peter Gonda: Internal Messages VJing

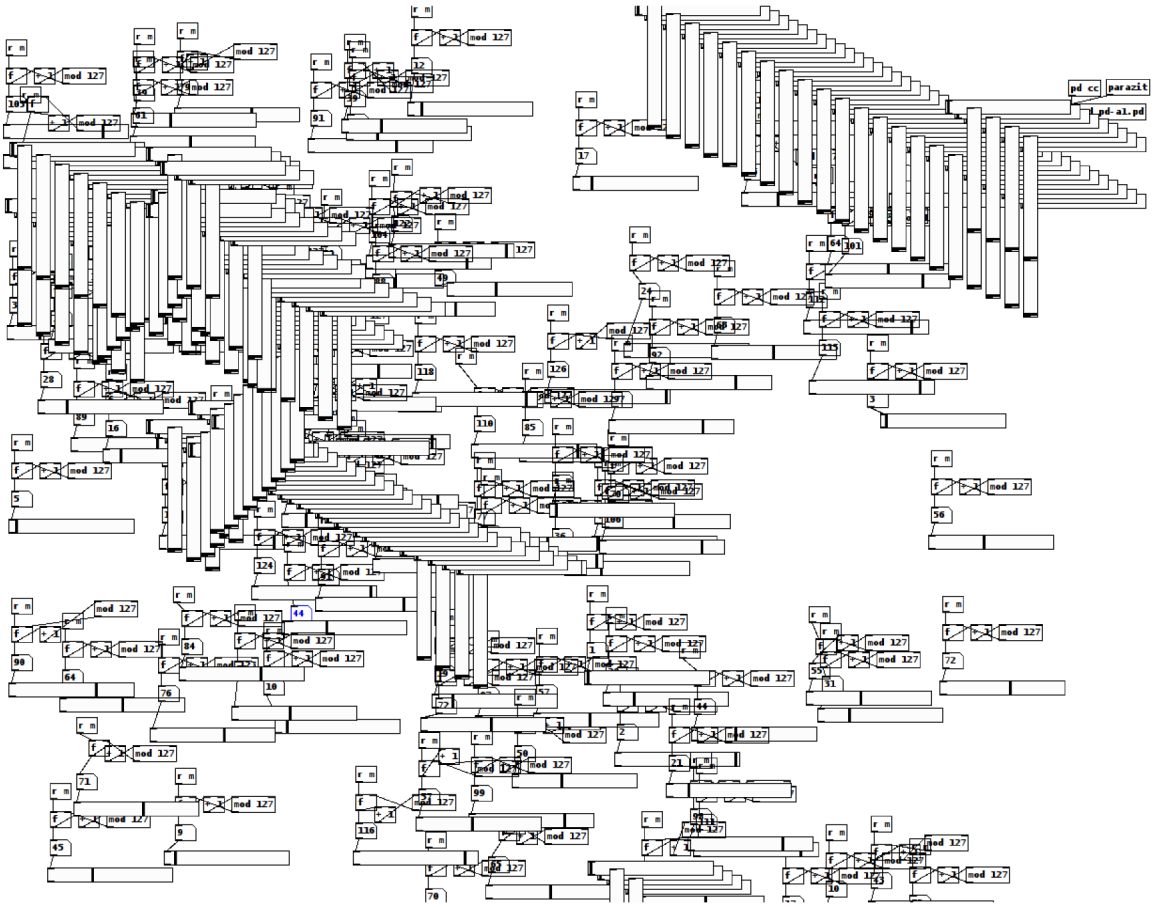
Počas strednej školy som na nemeckej televíznej stanici Viva Zwei každú nedeľu sledoval hodinový program zvaný 2Step. Ku setom elektronickej hudby od rôznych producentov bol celú hodinu robený živý vizuál rôznymi VJmi alebo VJskými skupinami. Fatálne inšpirovaný, tiež som zatúžil robiť pohyblivý obraz naživo ku hudbe. Nevedel som, aký nástroj použiť. Zdalo sa mi, že potrebujem niečo, čo by umožnilo veľkú interaktivitu a slobodu. Vtedajšie VJ-ske softy boli ešte v plienkach. Vedeli len loopovať videá, robiť prelínačky, prípadne efektovať video sadou štandardných (a už vtedy opozeraných) efektov. Hľadal som niečo slobodnejšie. Pure Data som spoznal v roku 2003 na workshope v bratislavskom priestore Buryzone (Buryzone bol organizovaný Marišou Riškovou a kolektívom, a ako taký predchádzal neskoršiemu media/datalabu Burundi). Na workshope rakúsky multimediálny umelec Hiaz (Mathias Gmachl) ukazoval, ako sa pracuje so softwarom Max/MSP. Prezentoval jednoduchý patch pracujúci s webkamerou snímajúcou miestnosť. Keď kamera detekovala pohyb, nahrala ho a zaloopovala. Bol som uchvátený. Dozvedel som sa, že existuje open-source obdoba Max/MSP zvaná Pure Data.

Celé prázdniny som strávil pripravujúc si svoj prvý VJ set z fotiek rozpadávajúcej sa košickej magnezitky. Fotky som naskenoval a rozanimoval v programe After Effects a videá naživo efektoval v Pure Data. Vytvoril som aj svoju prvú abstrakciu na frekvenčnú analýzu zvuku, ktorá riadila živý processing videa. Prvé vystúpenie prišlo na jeseň. Bol som pozvaný VJ-ovať v Prahe. Bolo to fiasko, na mieste požičaný laptop nemal taký výkon ako moje PC s grafickým akcelerátorom. Vizuály bežali rýchlosťou 2 obrázkov za sekundu. Odvtedy viac alebo menej intenzívne pracujem s Pure Data. Okrem živých vizuálov som v Pure Data robil aj interaktívne inštalácie, rôzne veci do divadla, streaming zvuku. Pure Data však primárne vnímam a používam ako nástroj na živú tvorbu vizuálu. Za vyše 10 rokov som si vytvoril sadu nástrojov a postupov, akými s Pure Data pracujem. Niektoré z nich stále rozvíjam. Pred každým vystúpením sa snažím prísť s niečím novým (vizuálne, štrukturálne). To, že niektoré svoje nástroje a postupy recyklujem, vedie k tomu, že mám akýsi rukopis (nadužívanie feedbackov, veľa farieb, trashová abstraktná estetika). Chcem, aby video bolo „tekuté“ a všetko bolo schopné stať sa všetkým v priebehu sekúnd. Domnievam sa, že ku niektorým z týchto vecí ma nezvádza moja kreativita, ale samotné Pure Data, ktoré samé od seba podmieňujú akúsi estetiku a spôsob práce s obrazom.

---

stupné z: https://www.youtube.com/watch?v=XKE3yAQMGRQ&feature=youtu.be. Kanál uživatele Tom Šen.

Z môjho pohľadu je používanie Pure Data nutnosť. Pure Data sú pomerne rýchle na to, čo potrebujem (oproti povedzme prostrediu Processing). S mojou skúsenosťou je ich používanie jednoduché. Pure Data bežia aj na operačnom systéme Linux (napríklad VVVV nebeží). Poskytujú mi slobodu ohýbať hotový kód podľa potreby. Takisto mám pocit, že som ešte možnosti Pure Data zďaleka nevyčerpal. Vyhovuje mi otvorenosť Pure Data v interakcii so senzormi, HW rozhraním Arduino, sieťovými protokolmi atd. Niekedy som na workshopoch zvykol Pure Data charakterizovať ako nástroj, v ktorom sú dáta tekuté, obraz sa môže prelievať do zvuku a vice versa, všetko môže byť všetkým. Chcel by som však poznamenať, že by som si Pure Data nevybral ako nástroj na precízne mapovanie priestoru, alebo komplexnejšiu prácu so živým video-vstupom (rozpoznávanie a trackovanie obrazu tanečníkov, rýchlych objektov atd). Pre rozsiahlejšie a komplikované projekty môžu byť Pure Data neefektívne, pomalé a v niečom „lajdácke“. Všetky high-level funkcie je totiž potrebné si v Pure Data vytvoriť. Otázku užívateľsky nepriateľského grafického rozhrania nechávam bokom. Pure Data však presne vyhovujú tomu, čo robím – vyrábam si nástroje, slúžiace na živú vizuálnu improvizáciu. Nástroje, ktoré viem recyklovať, alebo počas vystúpenia pretvárať.



Ukázka z VJingu s Internal Messages.

Okrem práce pre bratislavské divadlo P.A.T., pre ktoré som v Pure Data vytváral nástroje na živú prácu s viacerými kamerami, videom a obrázkami, som vo

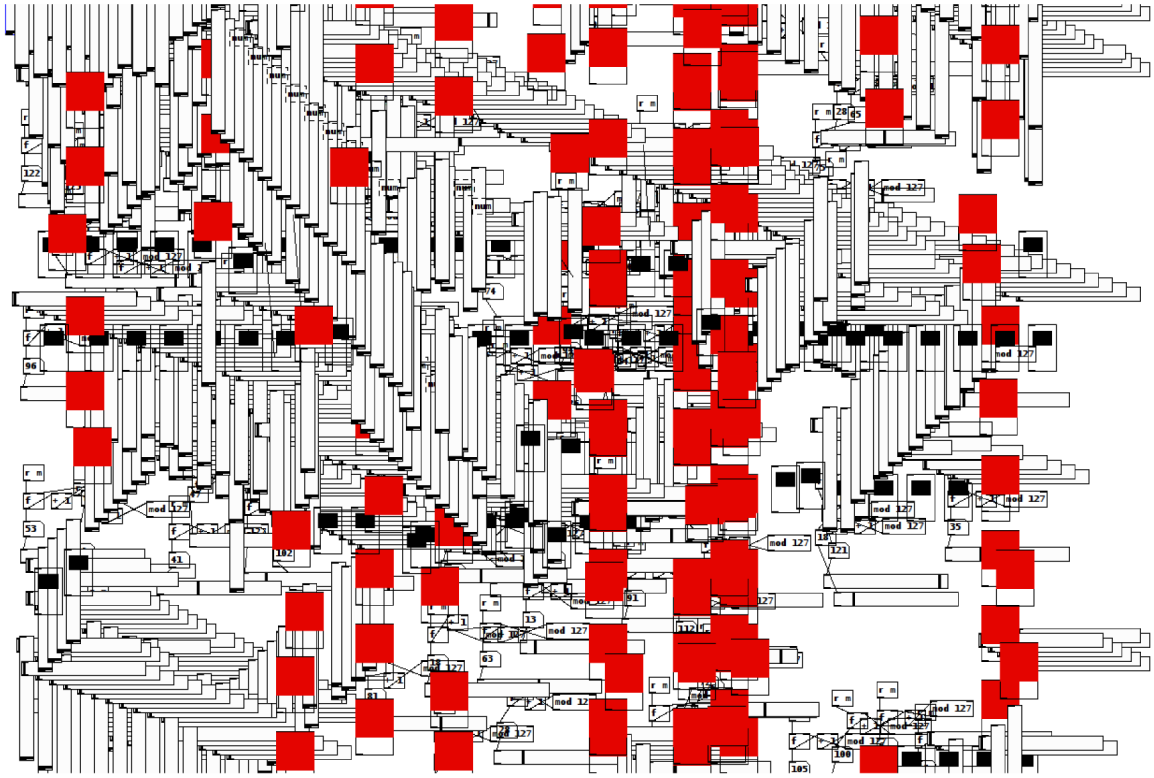
svojej divadelnej praxi použil Pure Data zatiaľ len raz – v predstavení Nagggnag. Stredočeská rodina večeria poslednú večeru pred previbrovaním sa do vyššej dimenzie. Jednou z podmienok previbrovania je schopnosť zaspievať čistý tón C. Túto schopnosť si rodina skúša na "trenažéri". Je to mikrofón napojený na analyzátor zvuku, ktorý vyhodnocuje čistotu a výšku tónu a podľa nej generuje grafiku. Pokiaľ herec spieva čisté C, tak sa na projekcii pred ním otvárajú rôzne "astrálne roviny" – tým rýchlejšie, čím dlhšie vydrží udržať tón. Patch, ktorý budem opisovať, som vyvinul pre diskusiu o generatívnej architektúre vo viedeňskom Museumsquartier v roku 2012. Bol som vyzvaný, aby som počas cca 3 hodinovej diskusie vytváral vizuálne pozadie. Ako koncept som si zvolil prácu so samotným grafickým rozhraním Pure Data. Pure Data v sebe obsahujú funkcionalitu, ktorá dokáže generovať samotný kód, prípadne manipulovať s celým patchom. Je to akési metaprogramovanie, kde programovací jazyk má schopnosť zasahovať sám do svojho vlastného kódu a behu programu. Táto schopnosť sa v Pure Data volá „internal messages“. Sú to pokyny, ktoré Pure Data posielajú samé sebe (vytvoriť alebo zmazať objekt, vytvoriť prepojenie medzi objektmi, označiť, prípadne posunúť objekt).

Počas diskusie sa zvyšovala komplexita toho, čo sa s patchom deje. S použitím Markov chains (matematický konštrukt, ktorý generuje nedeterministickú postupnosť stavov) som kontroloval postupné vytváranie a umiestňovanie rôznych objektov v pracovnej ploche. Počas príprav som zistil, že grafická nadstavba, v ktorej je interface Pure Data robený – programovací jazyk Tcl/Tk, znesie len určitý počet vytvorených objektov. Keď sa ich počet blížil ku tisíckam, interface Pure Data začal mrznúť a hrozilo, že nad patchom stratím kontrolu. Pridal som mazanie náhodných objektov, čo celý vizuál spravilo zaujímavejším.

Neskôr som začal používať nielen vytváranie objektov, ale aj ich prepájanie. Pravidlá generovania prvkov, určené podliehajucou logikou Markov chains, vytvárali náhodne pôsobiacu sieť objektov, ktorá ale mala akúsi skrytú 'usporiadanosť'. Do kompozície, ktorú generovanie objektov vytváralo, som začlenil aj nefunkčné objekty – veľké farebné triggre, slajdre alebo plochy farby. Generovaný kód menili na abstraktné, takmer až konštruktivistické plátno.

Najkomplexnejší vizuál pozostával z vygenerovaných množín objektov, ktoré nepôsobili len vizuálne – mali konkrétnu, aj keď nepodstatnú funkciu. Napríklad to boli počítadlá, ktoré zaslučkovane rátali od nula do sto. Boli napojené na slajdre, ktoré toto počítanie vizualizovali. Generoval som teda funkčný kód, ktorého funkčnosť sa s náhodným mazaním objektov rozpadala. Samozrejme oproti práci so živou syntézou videa sa jednalo o pomerne pomalú vec, ktorej čaro spočívalo v graduálnom vrstvení a komplikovaní „kompozície“ kódu. Zaujímavá je práve

táto dvojznačnosť – kód v Pure Data pôsobí často nesmierne vizuálne esteticky. Na druhej strane sa jedná o funkčný kód, tvoriaci časť programu, ktorý je možné vnímat v rovine estetiky kódu vďaka jeho kvalitám ako sú stručnosť, čistota alebo efektivita. Samozrejme, kód generovaný v Museumsquartier mal primárne 'vizuálne estetickú' funkciu, ale umožňoval vnímať ho aj z tejto druhej, programátorskej roviny.



Ukázka z VJingu s Internal Messages.

4.9 | Jiří Rouš: Cesta k Pd

Budu při popisu okolností, které mě přivedly k užívání programovacího jazyka Pure Data (dále jen Pd), průběhu a vedlejších vlivů, postupovat chronologicky. Vzniklá časová osa se dá rozdělit do tří základních částí. První část je před znalostí paradigmat grafického programování (dále jen gp), druhá fáze je rámována objevením gp, jeho principů a experimentování s ním hlavně skrze jazyk Pd. Třetí a poslední část spočívá v paralelní aplikaci načerpaných zkušeností a dalším zkoumání jak Pd tak i jiných programovacích jazyků a nástrojů, ale primárně v příklonu k principům freesoftware a opensource.

O první fázi cítím potřebu psát, protože obsahuje základní pochopení fundamentálních principů, které k druhé fázi snad i nepřímo vedou, ale hlavně mi byly oporou při studiu gp v Pd. Mezi mé tehdejší hlavní nástroje uměleckého snažení patřilo následující trio programů. Reason byl pro mě nástroj pro tvorbu hudby, nebo obecněji zvuku, což tou dobou bylo mým částečně utajeným koníčkem. Fla-

sh a jazyk actionscript byl nástroj, který nás učili na škole a který mě, sic umělce, ale přeci jen zasvětil do oblasti interakce, procedurálního a částečně také objektově orientovaného programování. Poslední nástroj té doby pro mě představoval program Cinema4D na tvorbu trojrozměrné grafiky. Každý z těchto programů mi dával k dispozici možnosti, které vedly k mé kreativní saturaci.

Zlomovým bodem byla idea interaktivního objektu, který by skrze senzorický aparát byl ve spojení s fyzickým světem. Pro tuto ideu jsem shledal nástroje, které jsem ovládal, jakožto nepoužitelné. To byl krok, který mě přivedl na scestí našeho tehdejšího školního plánu vzdělání a skrze hypertext k programu Max/MSP. Seznámení se se samotným paradigmatem gp pro mě nebylo něco zcela nového, hlavně díky předchozí zkušenosti s odvrácenou stranou softwarových syntetizérů v programu Reason. Po nějakém čase jsem se, dílem provázanosti programů samotných (paradigma, autor), dostal k Pd. Přechod z Max/MSP na Pd byl pozvolný, hlavně kvůli grafickému uživatelskému rozhraní (dale jen GUI), které mě v prostředí Max/MSP osobně více uspokojovalo, a toho jsem se nechtěl vzdát. Nakonec však ideologické založení Pd a jejich strohost získaly plně mou důvěru a jako nástroj pro uskutečnění zmíněné ideje se výborně osvědčily.

Další časté experimentování s kódem v Pd mě pozvolna dovedlo k uznání principů freesoftware a open-source jakožto rozumného způsobu, jak zacházet s kódem, mentálním vlastnictvím atd. Snad možno říct, že vedlejším účinkem objevení a používání Pd je v mém případě přechod z užívání proprietárních programů a operačního systému Windows na prostředí GNU/Linux a převážně programy ctící principy freesoftware a opensource.

Nyní se pokusím o shrnutí možností a limitací Pd. Rád bych zdůraznil osobní rovinu následující části textu, protože jsem přesvědčen, že většina zmíněných možností a především limitací bude pravděpodobně vyvratitelná či sporná, či vzhledem k podstatě jazyka Pd odstranitelná.

Pd jsou pro mě v současné době primárně nástrojem používaným k rychlým zvukovým skicám, občas vytvářeným v kontextu livecodingu (zmíněná občasnost je v kontextu četnosti veřejně prezentovaných akcí na pražské potažmo české livecodingové scéně). Dále pak Pd často využívám jako nástroj pro algoritmické řízení ostatních programů, které tyto kapacity ve své přirozené formě nemají. Jako příklad tohoto využití můžu uvést mnou často používané spojení programů Renoise jako zvukový generátor a Pd jako řídící komplement. Jeden z důvodů četnosti tohoto propojení vidím v absenci elementu časové osy v Pd, což na jedné straně vidím jako nedostatek a na druhé jako prostou vlastnost, která vede k jinému, nelineárnímu přemýšlení o vnitřním čase vytvářeného kódu nebo kompozice.

Jeden z nedostatků programovacího jazyka Pd vidím v absenci jednoduché možnosti jednorázového vytvoření mnoha instancí nějaké konkrétní abstrakce či objektu a následné hromadné manipulace s nimi. V textuálně orientovaných programovacích jazycích jsou k dispozici základní metody jako for cyklus atp., které se dají pro takový účel snadno použít. Jejich částečné ekvivalenty můžeme v Pd nalézt také, příkladem uvádím objekt [until], avšak jejich využití pro zmíněný účel v kontextu gp již není tak samozřejmé jako v jazycích textuálně, objektově orientovaných.

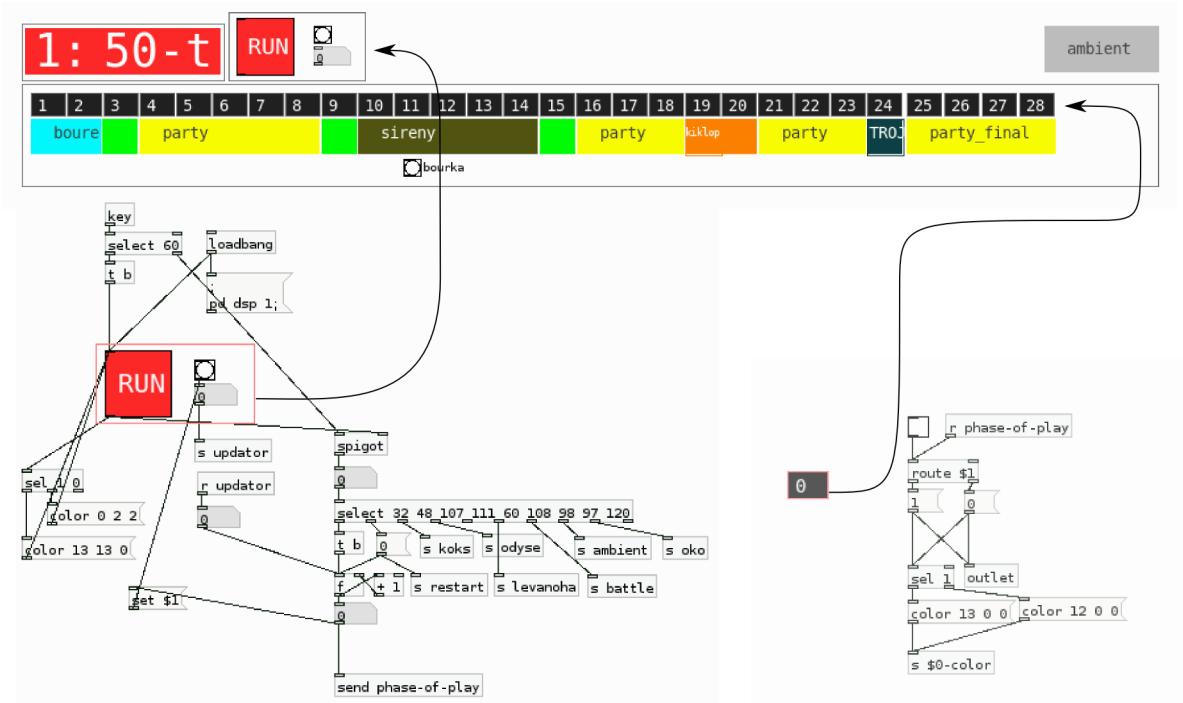
Charakteristickou vlastnost, kterou na Pd potažmo na celém gp vidím jako silnou a atraktivní, je propojení grafické složky a logické struktury kódu. Je však patrné, že ve své podstatě grafické rozvržení kódu je v gp abstraktní složka, která se samotnou funkcí programu nemá pevné spojení, na rozdíl od textuálně orientovaného programování, kde je dodržování syntaxu, tedy v podstatě prostorového rozvržení kódu, nezbytné. Čistota nebo čitelnost kódu plynoucí z logiky programu je tedy spíše konvencí, kterou je při běžné práci dobré dodržovat.

Pd se z mé zkušenosti neosvědčila jako báze pro rozsáhlejší ideje. Elegance kódu se v komplexnejších systémech postavených na Pd dle mého úsudku ztrácí za nepřeberným množstvím vnořených subpatchů a nebo v obrovské ploše potřebné k napsání kódu v jedné vrstvě. Mnoho větších komplexnějších kódů, řekněme systémů, které jsem během svého dosavadního používání Pd tvořil, buď zůstalo nedokončených, nebo bylo párkrát použito a pak byly opuštěny, a jejich další použitelnost postupně odumírala. Pd jsou pro mě skvělým nástrojem na výrobu jednorázových kódů k řešení konkrétní potřeby nějaké ideje, avšak jako nástroj pro výrobu univerzálně použitelných nástrojů se mi příliš neosvědčili.

Možná zásadní limitace jazyka je dle mého názoru oddělení číselných kontrolních a zvukových signálů, kteréžto mi svou podstatou připadají víceméně identické. V Pd jsou samozřemě dostupné prostředky ke vzájemnému převedení těchto typů, avšak jejich fungování, čerpám-li z mých zkušeností, nebylo vždy uspokojivé.

Patch, který k textu přikládám a který se v této části textu pokusím vysvětlit, je příkladná ukázka výše zmíněného typu zapomenutého kódu. Abych zde byl schopen jeho funkční stránku popsat, musel jsem podniknout drobné reverzní inženýrství na vlastním kódu, což bylo do jisté míry zábavné, posléze však stejnou měrou iritující. Zábavná stránka procesu spočívala v pronikání do již opotřebených záhybů paměti, v hledání a nalézání vzpomínek spojených s výrobou kódu a s událostí, pro kterou byl psán, tedy s divadelním představením Odysseus, které se hrálo na DAMU a režírovala je Bojana Sekicki. V tomto jsem však nebyl příliš úspěšný, a tak jsem musel dekódovat patch samotný. Absence komentá-

řů tento proces mírně komplikovala. Chaotický, respektive předem ne detailně promyšlený hrubý funkční koncept a jeho průběžné ohýbání a záplatování vedly ke vzniku kódu, jehož části by vzhledem k nepřehlednosti bylo elegantní napsat znovu, což se na druhou stranu dá vyřknout o celém programu. Leč neztrácejme čas a pusťme se do výkladu příkladu.



Ukázka GUI oddyseus.pd, subpatche RUN a abstrakce ody-phase.pd

Patch odyssesus.pd je ve své podstatě přehrávač konkrétních zvukových samplů k určitým segmentům divadelního představení. Některé stopy jsou vytvořeny jako smyčky, umožňující plynulé cyklické přehrávání, ostatní jsou jednorázové zvukové události. Vrchní vrstva programu spočívá v GUI umožňující jeho velmi omezené ovládání, avšak jeho primární podstata má charakter informační. GUI funguje tedy jako vizuální zpětná vazba stavu, ve kterém se program, jakýsi stavový automat, právě nachází, a jak dlouho od spuštění prvního stavu funguje. Ovládání programu samotného umožňuje klávesnice. Tlačítko s označením RUN je hlavní spouštěcí element, který aktivuje ovládání klávesnicí. Stisknutím mezerníku dojde k lineární změně stavu na další fázi, tedy změně zvukové složky představení. Stisknutím klávesy 0 (nula) je celý proces přehrávání přerušen a vrácen na začátek. Stisknutím klávesnic a, b, k, l, o a x se spouštějí jednotlivé zvukové efekty konkrétních hereckých akcí, které potřebují preciznější časování.

Přehrávání samplů samotných je zajištěno sítí objektů s jádrovým objektem [readsf], který umožňuje přehrávat dlouhé zvukové soubory z disku bez nutnosti je nahrávat do paměti počítače. Každá z těchto sítí je ovládána minimálně jedním objektem časové osy, který mění svůj stav z vypnutého na zapnutý porovnáním privátní proměnné odrážející pořadí na ose a aktuálního čísla hlavního

počítadla. Každá síť svým stavem bezprostředně ovlivňuje minimálně jednu předešlou sousedící síť. Toto ovlivnění ve valné většině spočívá v postupném zeslabení signálu či v opětovném spuštění přehrávání zvukové události. Samostatné zvukové události spřažené s konkrétními klávesami jsou ovládány podobnou sítí objektů. Jejich řízení však probíhá jednorázově, pomocí události bang. Mapování jednotlivých kláves probíhá v subpatchi s tlačítkem run porovnáváním číselných hodnot zmáčknutých kláves s vybranou sérií číselných hodnot konkrétních mapovaných písemných charakterů.

Program používá vlastní osobní abstrakci ody-phase.pd, nezbytnou k jeho fungování. Tento objekt obsahuje jednoduchou síť, vytvářející filtr vstupních dat, na základě parametrů objektu zadaných při jeho vzniku. Je-li číslo na datovém vstupu objektu stejné jako jeho iniciační parametr, pak objekt změní svou barevnost na červenou, není-li podmínka splněna, pak se barevnost vrací zpět do černé. Vnitřní objekty dohromady v podstatě tvoří klasickou podmínku if, která je zde vystavěna z bazálního Pd objektu [route]. V tomto případě objekt route směruje svůj vstup buď na jeden nebo druhý výstup, tedy spouští vždy jednu ze dvou zpráv, definujících změnu barevnosti GUI prvku canvas. Objekt ody-phase slouží jako jednoduchý grafický prvek s informativní hodnotou. Tento objekt není součástí běžných knihoven pro Pd, a proto byla výsledkem práce složka, obsahující všechny potřebné komponenty a zdrojové soubory spolu s textovým souborem, který popisuje projekt a používání patche jako takového.

4.10 | Jakub Pišek: Hallogenerator a Turbosampler

Na workshope Ping Pong na Stanici Žilina Zárieče som si uvedomil, že vytvoriť si program nieje nedosiahnuteľné. Krátko na to som sa začal venovať Flashu a jeho Action Scriptu, no rýchlo som pochopil, že hrabanie sa v stovkách riadkov kódu nieje jednoduché, ale zdĺhavé. Chcel som tvoriť rýchlo, multimediálne a modulárne. Na týždňovom workshope nieje čas na zdržiavanie sa low level jazykmi. Keď som sa začal venovať Pure Data, bolo pre mňa veľmi dostupné vytvoriť jednoduchú interaktívnu inštaláciu, a mnohí spolužiaci ma žiadali o pomoc, čo mi dodalo ešte viac skúseností. S pozvoľným prechodom na GNU/Linux som sa začal venovať aj elektronike, skrze dosku Arduino som dostal možnosť pripájať senzory, tlačidlá, svetlá a iné.

Performancie Hallogenerator a Turbosampler stoja na opačných koncoch určitého pohľadu na hudbu. Kým jeden používa overené kompozície a hrá ich na svojich nástrojoch, druhý používa fragmenty svetovej hudby a vytvára z nich vlastnú kompozíciu.

Jakub Pišek a *Turbosampler*.

Hallogenerator[38] je multimediálna elektrosága nešťastného párty robota z budúcnosti. Pri transporte časom sa jeho softwarové vybavenie určené na elimináciu populácie ľudí poškodilo mutantmi vírusov. Robota sa podarilo naštartovať iba v párty režime.

Je to môj prvý patch pre performancie na stage (myslim nie inštalácia, gadget). Jedným z mnohých naplnených cielov bolo zistiť, či experimentálny a prototypovací software patrí na javisko. Systém založený na MIDI timeline obsluhuje mnohé hudobné nástroje vrátane vokóderu, metronómu pre bubeníka, robotov aj projekcie.

Turbosampler[39] je zábavný mashupový projekt spájajúci prvky nového disca ako napríklad Huoratron alebo Mr. Oizo s postupmi z experimentálnej scény využívajúc rôzne glitche a pod. Vizuálna stránka jeho vystúpenia nám absolútne vyvracia ilúziu, že svet k nám obracia čitateľnú tvár, ktorú nám treba už len rozlúštiť. Ide o hru predbežných významov, ktoré však nevedú nikam, čo je pre akciu tohto typu turbosympatické.

[38] *Kubriel.servus.at* [online]. [cit. 2014-04-08]. Dostupné z: http://kubriel.servus.at/hallogenerator.

[39] *Kubriel.servus.at* [online]. [cit. 2014-04-08]. Dostupné z: http://kubriel.servus.at/turbosampler/.

Po rokoch experimentovania, desiatkach projektov, stovkách prezentácií prišlo niečo, čomu hovorím Turbosampler. Power mashup projekt, s ktorým som za posledný rok odohral performancie v Bordeaux, Prahe, Budapešti, Cluj – Napoca, Berlíne i niektorých slovenských mestách.

Turbosampler[40] je mashup a tak vyzerá aj ten patch. V priebehu dvoch rokov pribúdali záplaty, feature, kontrolery a niektoré abstrakcie sa asi nedajú a ani netreba poupratovať. Hlavne, keď je všetko prepojené so všetkým.

Program obsahuje dve abstrakcie audio playera, ktoré na vstupe berú jednoduchý message systém (číslo samplu, hraj, efekt, pretoč, freeze), ktorý vzniká z kontrolérov nad ním (klávesnica, myš, MIDI). Do výstupu zase dávajú informáciu o tom, ktorý sampel hrá a na ktorej pozícií sa prehrávanie nachádza. Tento výstup sa posiela cez lokálny internet do druhej instancie Pd, ktorá obsluhuje player na video.

Audio i video player čerpajú informácie o súboroch médií v textovom súbore, kde v každom riadku sú informácie o audio súbore, jeho samplerate, hlasitosti, video súbore, začiatočnom snímku a záverečnom snímku. Takto je priradená každému zvuku určitá časť videa.

Vďaka modulárnosti a flexibilite sa mi podarilo využit nepoužívaný mikrofón laptopu, na ktorom performacia bežala na nahrávanie audio záznamu mnohých koncertov. Raz v rýchlosti, cestou na koncert, som dorobil systém na nahrávanie troch kanálov. Dvoch, ktoré sú hlavným audio výstupom a zapájajú sa do aparatúry, a jedného kanálu z mikrofónu. Nebol čas to testovať, ale naostro všetko fungovalo ako malo. Vďaka tomu mám výborné záznamy z mnohých predstavení. Sám neviem, ktoré iné prostredie by mi ponúklo možnosť takéhoto (ale aj mnohých iných) zásahu v limitovanom časovom rozmedzí, bez testovania a v ostrom nasadení.

V čase, keď som začal tvoriť patche Turbosamplera, bolo pre mňa nepredstaviteľné, že by som niečo také vytvoril v inom programovacom prostredí, ako je Pure Data. Skúsenosti a znalosti Pd mi dovolili vytvoriť komplexný a fungujúci systém.

4.11 | Martin Blažíček: Pd jako „pomůcka k myšlení“

Jak se stane to, že umělec, který byl zvyklý používat ve své tvorbě převážně analogové nástroje, dospěje k rozšíření svého nástrojového repozitáře o software jako Quartz Composer, Pd a další?

[40] *Kubriel.servus.at* [online]. [cit. 2014-04-08]. Patch je ke stažení na: http://kubriel.servus.at/uploads/turbosampler.tar.gz.

Digitální nástroje vidím jako logické pokračování analogového světa, protože do určité míry principy analogového světa imitují, případně rozvíjejí dále. Např. systém patchbaye, který existuje v nástrojích od 60. let, je přítomný i v nástrojích, které se používají virtuálně, ale de facto plní podobné funkce jako nástroje analogové.

Co to znamená, proč se používají nástroje, které vycházejí z analogového paradigmatu na digitální platformě?

Mně dnes přijde logické spíš analogové nástroje nepoužívat, je to něco, co patří do minulého století a je svázáno s určitým výrobním a průmysovým řetězcem. Používat dnes „analogy" je gestem proti času. Ne že by to nešlo, ale nemůžeme se tvářit, že to není významné. Kdybych se rozhodl, že bych nechtěl používat digitální nástroje a chtěl mít např. analogovou syntézu, úmyslně popírám aktuální vývoj a „ducha doby". Samotné rozhodnutí je vlastně velká věc. Na rozdíl od toho, když používám digitální nástroj, ten používám samozřejmě – je k dispozici tady a teď a vypovídá i o tom, jak jsou v současnosti nahlíženy určité principy jako synestezie, morfování signálu…

Je použití analogového nástroje dnes sentimentální?

Asi ani ne, spíš je to něčím významné – to rozhodnutí. Sáhnout po analogovém nástroji jako je VHSka není dnes nejjednoduší cesta – naopak je nejobtížnější a nejdražší. Když sahám po VHSce, revokuji estetiku média, která už zašla a je obtížně dostupná. Je to ale zároveň druh útoku na osobní vzpomínky a zážitky, které spousta diváků má.

A vidíš kromě té imitace v digitální sféře i nějaké novum?

Digitální nástroje do značné míry simulují „analogy". Druhá věc je ale koncepční – ta souvisí s tím, jak se ty jazyky používají a k čemu byly navrženy. Je v nich určitá intence. Film má strukturu následných snímků v čase – to je něco, co v nástrojích, které mají modulární rozhraní, tak moc přítomné není. V nich se s obrazem a zvukem zachází jako s časovým průběhem, tekutým signálem. Nějaké podobnosti tady ale budou, např. formalistní přístup k montáži, kdy digitální nástroje umožnily vytvářet algoritmy pro střih. Ale apriori to určeno není.

Otvírají pro tebe nástroje jako Pd nový prostor pro to, jak pracovat a tvořit?

Např. prvek okamžitosti, také to ale není nic nového – digitální obrazový artikulátor Woodyho Vasulky dosahoval vyšších výkonů, ale nebyl to obecně rozšířený nástroj. Zatímco dnes je možné věc udělat ihned, časová báze audiovizuálního díla vlastně neexistuje, nebo je možné ji libovolně variovat. Všechny snímky jsou dostupné hned, což nebylo vlastností analogového videa ani filmu.

Možná ta okamžitost směřuje k aplikacím alternativních časových průběhů, které se vymykají tomu, jak se média tradičně přehrávají. Také možná směřuje k pohledu na film jako ne na kontinuální bázi nějakých snímků v čase, ale jako na sumu okének, se kterými můžu snadněji dělat to, že vyberu např. červená políčka. Časové podmíněnosti najednou mizí a je možné zacházet s časem na jiné úrovni, než před tím.

Positivum Pd spočívá v tom, že jsou dostupná komukoliv – je to univerzální platforma, i když má své „mouchy“ a může posloužit vlastně i těm, kteří se jí nechtějí příliš zabývat. Pd jsou multiplatformní a člověk to vlastně ani „nemusí umět“ - je to vývojové prostředí, ale již hotová řešení se v něm dají snadno implementovat. Ostatní software jsou obvykle za peníze nebo jsou pro specifickou platformu. Např. Quatz běží na OpenCL a díky tomu je asi trochu výkonnější než knihovna GEM pro Pd. Ale zase je jednoduchý a neumožňuje pokročilejší matematické konstrukce, vícevláknové procesování atd.

Pd jsou ideální v místech instalace, která by šla vyřešit s pomocí elektroniky – může to být spouštění světla, zvuku, čtení dat ze senzorů. Pd umožňují řešit tyto situace i začátečníkům.

Puckette prohlásil, že poslední dvě dekády patřily tvůrcům systémů a budoucnost bude patřit tvůrcům algoritmů. Máš dojem, že se toto projevilo a projevuje i na audiovizuální scéně?

Osobně myslím, že ne – že to není podstatné. Pro někoho na „vyšší úrovni“ to může být zajímavé. To už je pak nějaká liga, kde není podstatné, jestli to byl ten, nebo onen nástroj, ale spíš ochota pracovat s něčím komplexnějším. Mám zkušenost, že když si někdo zvykne pracovat s nějakým vývojovým prostředím, pak většinou dokáže pracovat s jakýmkoliv. Mám ale dojem, že většina lidí na audiovizuální scéně se vývojovým prostředím nezatěžuje a používá jednodušší aplikace jako Resolume, Vidvox, Modul8 a je to dostačující. Pakliže používají ty složitější postupy, obvykle se tím řeší nějaký dílčí problém. V tom mainstreamu si nemyslím, že je to populární nástroj. Spíš je to charakteristické pro určitou komunitu, která má náklonost k určitému druhu složitosti.

Co se týče budoucnosti, která bude patřit tvůrcům algoritmů, to se myslím nestalo. Přítomnost a budoucnost patří tvůrcům obsahů, kteří používají různé nástroje. Toto tvrzení se nepotvrdilo, naopak dnes jsou některé nástroje tak důmyslné, že v sobě nesou způsoby používání, které od uživatele očekávají. Ableton automaticky detekuje BPM a warpuje do rychlosti, kterou hudebník vyžaduje. Tyto nástroje už samy předpokládají, co má být tím cílem, a ve spoustě věcí dokážou tvůrcům pomoci – tvůrčí role se pak posouvá do úrovně obsahu.

A co performance, kdy se z interface dělá obsah?

To je normální. Záleží ale na tom, co se tím myslí. VJ Aka47 (Andrea Pekárková) také hraje interface a používá Resolume. Tento přístup není něco, co by bylo limitováno Pd.

To si také nemyslím. Jde spíš o to, co to znamená – že se najednou nástroje a interface začaly objevovat a tematizovat přímo.

Po pravdě té tematizaci interface moc nerozumím, zajímavější mi přijde třeba synestetická kvalita, např. Ryoji Ikeda.

Sám nemám tendence takto uvažovat, ale přijde mi, že tu tematizaci interface je možné pochopit jako určité politikum. Jako by autoři – ať už výslovně, nebo nevýslovně – říkali: budeme tematizovat, problematizovat a rozkládat interface, který je nám dán k užívání, podívejte se na to…

Stanovit, co to znamená, že se začal tematizovat interface, je asi dost problematické. Znám ale performance s interface, které vytvářejí bizarní a komické situace. Navazují na určitý typ zkušenosti, který diváci s interface mají. Viděl jsem krásnou performance, kde někdo „letěl“ kurzorem po desktopu, na pozadí měl hvězdnou oblohu a do toho na pozadí zněl start rakety.

Co říkáš přítomnosti postdigitálních témat na letošním Transmediale?

Neviděl jsem všechno. Výstava byla out-sourcingovaná, nebyla kurátorovaná. Předpokladem bylo pracovat s digitálním odpadem z Nigérie. V teoretické úrovni pro mě zazněla zajímavá myšlenka v jedné diskuzi – o tom, že digitální média vlastně neexistují, že nemají žádnou specifickou digitální vizualitu. Jak imitují předchozí média, stejně jako kindle imituje papír a monitor imituje projekci

v kině, tak tam nevzniká nová kvalita, jen posunutí. Jak popsal Bolter a Grusin nebo Manovich: média v sobě obsahují jiná média a nějakým způsobem je transformují. Digitální média vlastně nic nového nevytvoří, mixují a transponují ta předchozí média. Otázkou je, jestli, když se za 150 let podíváme na tuto krátkou etapu digitální kultury, která už nebude existovat a nebude se na ni možné podívat, nebudou žádné počítače, nanejvýš nějaké 3D tisky (smích), největší část digitální kultury zanikne… jestli náhodou ten post-digitální svět nebude mluvit o té digitální periodě jako o něčem, co bylo transformační a co proměnilo stávající média. Co bychom z toho získali pro naše poznání, by byla otázka a snad i odpověď, týkající se toho, jak média vypadala před digitální etapou a jak vypadají po ní. Ale znovu budeme mít k dispozici ta původní média jako socha, papír, fotografie, která zůstanou.

Jinak ze všech stran „zazníval“ zájem o fyzický svět. Pro současnou generaci je lákavější „otisknout svou stopu do světa“ fyzicky, protože ta digitální „stopa“ jim přijde nedostatečná.

To mi ostatně připomíná i tendence na „novomediální“ scéně, kdy s příchodem Arduina začala vznikat spousta elektromechanických instalací. Také se tam ta „hmota“ začala projevovat…

Má produkce v Pd nějaký charakter?

Já myslím, že má – jsou určité momenty, které jsou punketem toho, že jsou to Pd. To poznáš. Například při práci s obrazovým matrixem Max umožňuje progesivnější sampling, dokážeš ta data z grafického pole zapisovat a ukládat, pracovat ve vrstvách. V Pd je to trochu problém. Přijde mi, že Pd jsou charakteristická „vytěžováním“ módů, které ti ten který objekt poskytne. Na výstupech je to vidět. Skrze dílčí omezení se zkrátka „vytěžují“ možnosti. Knihovna GEM má specifické problémy – např v efektech, gain a saturaci…

Takže ten charakter vyplývá ze specifických chyb? (smích)

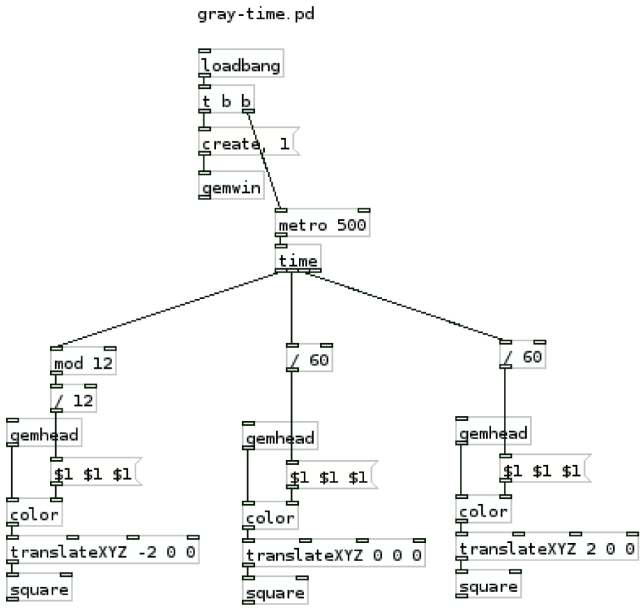
No, třeba když si někdo postaví řetězec v GEMu, málokdy si ze signálu udělá RGB s alfa kanálem. A když v tom řetězci pak je gain, nebo saturace, ten obraz se pak „rozpadá“. Spoustě lidí to ale přijde fajn a pracují s tím normálně dál. Je to problém dokumentace ke knihovnám. Tyto chyby vznikají poměrně často a autorům přijdou zajímavé.

Co ty jako pedagog říkáš na použití Pd jako pedagogické pomůcky? Například co se řemeslného zázemí v „novomediální“ tvorbě týče…

Mohli bychom mluvit o nějaké familiárnosti ve vztahu k nástroji. Když budu několik let malovat akvarely, získám určitou suverenitu. Můžu v něm objevovat polohy, které začátečník nedokáže – mám v tom pak jemnější a intimnější polohy. Zároveň k tomu budu mít osobní vztah, protože je ta věc fyzická. Proč by to tak nemělo být i ve světě počítačů? Prostředí operačního systému nebo aplikací se stávají „životním prostorem“. Dokonce se mi zdá, že řada mladých lidí dnes zaujímá víc citový vztah k operačnímu systému než k něčemu, co je ve fyzickém světě. Na druhou stranu je ten fyzický svět velice láká – třeba vyvolat si film z „osmičky“, to je něco speciálního, složitého a není to na „jeden klik“.

Já jsem Pd používal jako specifickou „pomůcku na myšlení“. Není to o tom naučit se ten konkrétní jazyk, ale spíš jde o to, jak rozčlenit nějaký problém na dílčí kroky, naučit se algoritmizovat. To už je něco, co je nad rámec zacházení s aplikacemi jako je Photoshop atd.

Pak je rozdíl v tom, že například při střihu videa neděláš to, že bys ho manuálně rozřezával a posouval na časové ose. Místo toho jen napíšeš, co se má stát, a ono se to stane a to je velká změna v uvažování. Myslím, že v Pd je tento způsob práce dobře demonstrovatelný a že si jej studenti můžou třeba i oblíbit. Pro mě osobně už je teď třeba skoro nemožné se vrátit k tomu starému způsobu práce s časovou osou…


gray-time.pd
loadbang
t b b
create 1
gemwin
metro 500
time
mod 12
/ 12
gemhead
$1 $1 $1
color
translateXYZ -2 0 0
square
/ 60
gemhead
$1 $1 $1
color
translateXYZ 0 0 0
square
/ 60
gemhead
$1 $1 $1
color
translateXYZ 2 0 0
square

5 | Souvislosti

A potom Pozemšťané objevili nástroje. Náhle mohl souhlas s přáteli znamenat formu sebevraždy nebo ještě něco horšího. Ale souhlasilo se dál, nikoli už ze zdravého rozumu nebo slušnosti nebo sebezáchovy, ale z přátelství. Místo aby přemýšleli, pěstovali nadále přátelství. A i tehdy, když sestrojili počítače, aby myslely za ně, nepožadovali od nich ani tak moudrost, jako přátelství. Proto byli odsouzeni k zániku.

Snídaně šampiónů

Výsadou toho kterého autora je nevidět věc jednoduše. Mohli bychom rukověť zakončit předchozí kapitolou a spokojit se fakty: Pure Data jsou programovací jazyk, který se dá použít tak a tak, má tyto výhody a nevýhody atd. Tím bychom ale byli ochuzeni o asociace, významy a možné světy, které lze v souvislosti s Pure Daty rozehrát. To, co se má tak a tak, může být ještě něco jiného. K našemu tématu můžeme přistoupit jako ke hranolu, jímž nahlédneme povahu přítomnosti a „ducha doby“. Poslední kapitola rukověti bude tedy pokusem o nahlédnutí Pure Dat v širších souvislostech – zejména zde půjde o úvahy týkající se postdigitální kultury. Budeme se opírat o „kanonizované“ texty a rozšíříme je o aspekty, které se v nich výslovně nezmiňují. Ke slovu tak přijdou například antropologické a utopické motivy, související se vztahem umělce a jeho nástrojového repozitáře. Nezanedbatelným pojmem pro nás bude chyba neboli *glitch*. Pojednáme též o specifickém étosu postdigitální umělecké praxe. Té lze rozumět v souvislosti s DIY kulturou, otevřeným software a hacktivismem. A konečně: pokusíme se také nahlédnout některé z tendencí v postdigitální umělecké praxi jako „avantgardu“.

5.1 | Postdigitalita

S předponou post- je spojený určitý „intelektuální humbuk“ a nedůvěra související s jejím nadužíváním. K postmodernismu, poststrukturalismu, postkolonialismu, postfeminismu, postkomunismu, postpunku atd. se přidává ještě pojem postdigitality. Můžeme k němu přistoupit s apriorní nedůvěrou a prozkoumat ho s nadějí, že jeho denotace není nulová.

Adjektivum *postdigitální* se kolem roku 2000 začíná objevovat v řadě textů, které se zabývají vztahem vývoje techniky a kultury.[1] Domnívám se, že zásadní roli v jeho vymezení hrála esej s názvem *Estetika selhání*[2] od Kima Casconeho. Naposled pojem postdigitální zaznívá z textů od Floriana Cramera, Erica Snodgrasse a dalších[3] a také z diskusí na Berlínském Transmediale v roce 2014.[4]

Kim Cascone začíná svůj text citací Nicholase Negroponteho, který v roce 1998 prohlásil, že digitální revoluce skončila. To je pozoruhodné a provokativní tvrzení, které nás na první poslech zarazí svou zdánlivou nepravdivostí – digitální přístroje se přeci stále zdokonalují v oblasti kvality reprezentace dat. Negroponte měl ale na mysli patrně něco jiného: zdokonalování kvality reprezentace není možné chápat v tak silném smyslu jako například přechod od analogových technologií k digitálním – nejde zde o změnu paradigmatu. Tato snaha o zprostředkování skutečnosti, v snad ještě lepších barvách než kterými sama disponuje, je spíš manýrou usvědčující digitální éru z její vyčpělosti. A nové paradigma, na které se můžeme „těšit“? Bude-li jaké, budou v něm nějakou roli hrát patrně kvantové počítače nebo biodigitální technika. Po hardware a software se ke slovu dostane wetware.

Ještě před tím, než přistoupíme ke konkrétním motivům Casconeho textu, bych se pokusil zprostředkovat zjednodušené vidění pojmu postdigitality v souvislosti s širším pojmem postmoderny. Nemáme přitom na mysli onu bezbřehou a „anything goes“ postmodernu, ve které není problém namíchat koktejl z freudismu, esoteriky a populární interpretace věd, ale spíš onu postmodernu přesnosti, jak ji interpretuje Wolfgang Welsch.[5] Tu Welsch, v odkazu na Lyotarda, opisuje vícero způsoby: hovoří o ní jako o stavu radikální plurality, kdy pojmy jako pravda,

[1] *Monoskop.org* [online]. [cit. 2014-04-10]. Dostupné z: http://monoskop.org/Post-digital.

[2] CSERES, Jozef a MURIN, Michal. *Od analógového k digitálnemu...* Banská Bystrica: Fakulta výtvarných umení, 2010. 219 s. ISBN 978-80-89078-78-3. s. 111-119. CASCONE, Kim. The Aesthetics of Failure. In: *Subsol.c3.hu* [online]. [cit. 2014-04-04]. Dostupné z: http://subsol.c3.hu/subsol_2/contributors3/casconetext.html.

[3] *Aprja.net* [online]. [cit. 2014-04-09]. Dostupné z: http://www.aprja.net/?page_id=1291

[4] *Transmediale.de* [online]. [cit. 2014-04-09]. Dostupné z: http://www.transmediale.de/content/after-the-revolutions-internet-freedoms-and-the-post-digital-twilight.

[5] WELSCH, Wolfgang. *Naše postmoderní moderna*. Praha: Zvon, 1994. 200 s. ISBN 80-7113-104-0.

spravedlnost a lidskost existují již jen v plurálu[6], nebo jako o myšlení, pro které je charakteristický odpor vůči hegemonii a univerzálním příběhům. Postmoderna přitom není něčím, co by bylo vynálezem filozofů nebo teoretiků umění. Radikální pluralismus a destrukce snu o jednotě (ať už vědecké, ekonomické, politické nebo kulturní) není něco, co by bylo třeba vymýšlet, ale něco, co se fakticky odehrálo a co se reflektuje.[7] Dále spolu s Welschem můžeme říct, že postmoderní myšlení vychází z kritické revize ideálů moderny[8], demytologizuje je a pokouší se na „zbořeništi" budovat nové myšlenkové podhoubí, jež respektuje jinakost a mnohost. Právě díky respektu k mnohosti však postmoderna nevylučuje ani způsob myšlení, který byl charakteristický pro modernu – jen ho již nevnímá jako jediný možný.

K nástinům chápání postmoderny, díky kterým pak budeme s to vidět lépe podobnost s pojmem postdigitality, přidejme ještě tyto Welschovy poznámky:

> Konec moderny podle Amitai Etzioniho nenastává jejím opuštěním, ale transformací. Úvodem k postmodernímu období je proměna techniky. Po druhé světové válce skončila moderní epocha radikální proměnou způsobů komunikace. Etzioniho pozice se dá shrnout tak, že chce, aby v postmoderní době techniky nepůsobily jako nástroje stupňovaní moderních technokratických tendencí, ale aby se technická racionalita relativizovala, měla jen statut nástroje a aby hodnoty znovu získaly prioritu. Tyto hodnoty konvergují k ideálu „aktivní společnosti" jako společnosti, která je vnímavá pro potřeby svého měnícího se členstva a nachází se ve stavu intenzivní a stálé transformace.[9]

> Pro Lyotarda je otázka po hledání odpovědi a orientace v postmoderní situaci determinována informační technikou. Argument zde ale nezní: tato nová technika existuje, její vzestup je nezadržitelný, vytvořme tedy myšlení, které jí odpovídá. Ale: nová technika přichází a ovlivňuje vědění, učiňme si tedy jasno o vnitřní povaze a cílech současného vědění, abychom mohli správně reagovat na výzvu techniky, to znamená, abychom ji mohli využít, pokud je slučitelná s touto povahou vědění, a abychom se proti ní postavili tam, kde tomu tak není.[10]

[6] Ibid., s. 13.
[7] Ibid., s. 12.
[8] Zde musíme upřesnit, že když užíváme slovo moderna, nemyslíme tím umělecké a vědecké moderny počátku 20. století. Modernou se zde míní období ohraničitelné časově osvícenstvím a počátkem 20. století. Myšlenkovým ohniskem této moderny je jednak racionalita, která je zakořeněna v osvícenském pojmu pokroku, kumulaci vědění, a jednak idea universalismu, tj. obecné platnosti a závaznosti této racionality. Obojí je v ní legitimizováno s pomocí nových mýtů, jako je například věda.
[9] Ibid. s. 35.
[10] Ibid. s. 40.

Z uvedených ukázek je snad patrný kritický a rezervovaný postoj ve vztahu k technice. Do určité míry je v těchto citacích možné nahradit pojem moderna pojmem digitál a pojem postmoderna pojmem postdigitál. Předešleme zde, že pro postdigitalitu je charakteristická kritika techniky. A stejně jako postmoderna nevylučuje modernu, ale vymezuje se vůči ní, tak je pro postdigitální postoj charakteristické vymezování se vůči digitálu a může se dít i digitálními prostředky. V tomto duchu lze nahlédnout například destruktivně-kreativní praxi *circuit-bendingu*.[11]

Postdigitalita není vyhlášením konce éry digitálu, ale spíše její transformací. To, co se proměňuje, je pak postoj umělců k prostředkům produkce. V tomto transformačním ohledu vidí postdigitální tendence i Florian Cramer, když se z roku 2014 ohlíží o jednu dekádu zpět a k osvětlení pojmu postdigitální si pomáhá například pojmem postapokalypsy – ten naznačuje, že svět již prošel oním momentem proměny a pokračuje dále, událost se transformuje do stavu.[12]

Nyní se vraťme zpět k textu *Estetiky selhání*. Cascone se v něm pokouší popsat tendence, charakteristiky a změny, ke kterým podle něj došlo zejména na elektronické hudební scéně v druhé polovině 90. let 20. století. V návaznosti na Negroponteho se rozhodl označit je adjektivem postdigitální. Jedním z „motorů" postdigitality je pro něj, ve shodě s výše uvedenými citacemi, příchod informační techniky a nových způsobů komunikace. Díky internetu, snadnému přístupu k informacím a jejich výměně a filtraci v síti vznikly skupiny neakademicky vzdělaných umělců.[13] Jejich praxe byla poznamenána hlubokou zkušeností s digitální technikou.

Ohnisko zájmu postdigitálních umělců se ale posunulo od médií směrem k specifickým nástrojům[14] a netradičním technikám. Cascone navazuje na Marshalla McLuhana a tvrdí, že oním poselstvím již není médium, ale právě nářadí! Zmiňuje například německou skupinu Oval, která začala experimentovat s manipulací datových stop na CD (v našem kontextu pak asi nelze nezmínit Destruovanou hudbu od Milana Knížáka), do hudby se prolamují zvuky, které byly do té doby

[11] Jde o otevření a modifikaci v zapojení elektronického hudebního nástroje. Tím je dosaženo nové kvality ve zvuku, kterou původní nástroj neměl.

[12] CRAMER, Florian. What is 'Post-digital'? *Aprja.net* [online]. [cit. 2014-04-10]. Dostupné z: http://www.aprja.net/?p=1318.

[13] Zde je zajímavé, že Cascone již v té době předjímá to, co dnes již v náznaku lze vidět lépe. Jde o „naleptávání" tradičního modelu vzdělání a autority vzdělávacích institucí. Zdá se, že žijeme v době, kdy k tomu, aby ze sebe člověk udělal experta v dané oblasti, potřebuje jen přístup k internetu a odhodlání. Zdá se také, že některé z pružnějších univerzit na tento fakt již začaly reagovat.

[14] Proto Cascone patrně také jako předobraz postdigitálních tendencí zmiňuje futuristu Luigi Russola a jeho *intonarumori*.

jen na „okraji dění“ (klikání myši, hluk ventilátoru), Mika Vainio z Pan Sonic hraje na sadu doma postavených oscilátorů. Umělci otevírají obaly elektronických hudebních nástrojů, pod kterými nacházejí tištěné obvody, do kterých více či méně poučeně sahají za účelem ohnutí výrazu daného nástroje. Paralelně s tím probíhá vývoj nových softwarových nástrojů pro syntézu obrazu a zvuku (Max/MSP, Pure Data, SuperCollider, Processing), díky kterým se umělcům otevírají dříve obtížně přístupné oblasti (microsound, granulární syntéza). Chyba v interpretaci dat, neboli *glitch*, již není něčím nežádoucím, ale stává se krásotvorným prvkem. Cascone se domnívá, že pro postdigitální díla není charakteristická adorace techniky, ale spíš zaměření se na její selhání:[15]

> Ze selhání se na konci 20. století stala významná estetika, která připomíná, že naše kontrola nad technikou je jen iluzí. Selhání techniky odhaluje, že digitální nástroje jsou pouze tak dokonalé, precizní a výkonné, jako lidé, kteří je zkonstruovali.

Typická ukázka *glitche* v digitální fotografii.

Nezanedbatelná část Casconeho eseje se týká právě výkladu pojmu *glitch* – tedy chyby v interpretaci dat. Je lhostejné, zda jde o chybu ve zvuku, obraze nebo textu. Chyba, ať už jako pouhé krátké cvaknutí v reproduktoru nebo jako výsledek

[15] CASCONE, Kim. The Aesthetics of Failure. In: *Subsol.c3.hu* [online]. [cit. 2014-04-04]. Dostupné z: http://subsol.c3.hu/subsol_2/contributors3/casconetext.html.

programátorské neopatrnosti, která vede ke zhroucení systému, je momentem, který digitál zbavuje jeho iluzorní čistoty a dokonalosti.[16]

Pro postdigitální tendence je podle Kima Casconeho dále charakteristické zaostření se na „okraj dění“ a pozadí techniky – umělci nejsou již tolik fascinováni samotným médiem, ale principy, na kterých média a technika fungují. V hudební kompozici pak tento posun může znamenat opuštění tonatlity směrem k tomu, co hudbu „nese“: ticho, hluk, fyzikální zákony.[17]

> Při prověřování zrodu současného postdigitálního hnutí je třeba brát v úvahu pojmy jako „odpad“, „vedlejší produkt“, „pozadí“ nebo „horizont“. Když výtvarní umělci poprvé zaměřili svou pozornost z popředí na pozadí, napomohlo to rozšíření jejich percepčních hranic. Najednou dokázali zachytit tajemnou povahu pozadí. … Zvukoví umělci [jako Russolo a Cage] přesunuli svůj zájem z hudebních tónů na náhodné zvuky.

Domnívám se, že Cascone ve svém textu poskytuje dostatečné množství charakteristik a asociací, které pojem postdigitální vyplňují. Jeho „slabina“ spočívá v tom, že se nezabývá přesnějším vymezením pojmů digitální a postdigitální. To je na druhou stranu přijatelné, když *Estetiku selhání* pochopíme jako druh uměleckého manifestu, než jako odborný text. Casconemu jde spíš o vymezení se vůči minulým trendům a o popis těch tehdy současných.

5.2 | Postdigitální přesahy

Nyní se pokusíme odpovědět na otázky, týkající se příčiny, motivů a nevýslovných významů, které v sobě postdigitalita nese. Nejprve můžeme konstatovat, že vytváření nástrojů není nic nového. Rozdíl lze ale vidět v míře dostupnosti nástrojů a kvantity produkce. To, co dříve ve svých vybavených studiích mohli provozovat Karlheinz Stockhausen nebo Woody Vašulka, dnes dokáže každý jen s pomocí laptopu. Díky řadě poměrně jednoduchých a přístupných aplikací k vytváření hudby (Ableton, FLStudio, MAGIX Music Maker atd.) se nabízí otázka problematizující kreativitu.[18] Tyto aplikace se, díky svým propracovaným a intu-

[16] Florian Cramer zdůrazňuje, že pojmu digitál by v tomto případě nemělo být rozuměno v jeho vědecko-technickém smyslu, ale spíš tak, jak se mu rozumí obecně. Pro ilustraci tohoto obecného rozumění používá Cramer sérii „čistých a chladných“ obrázků, které zprostředkuje vyhledávač Google po zadání hesla digital. CRAMER, Florian. What is ‘Post-digital’? *Aprja.net* [online]. [cit. 2014-04-10]. Dostupné z: http://www.aprja.net/?p=1318.

[17] CASCONE, Kim. The Aesthetics of Failure. In: *Subsol.c3.hu* [online]. [cit. 2014-04-04]. Dostupné z: http://subsol.c3.hu/subsol_2/contributors3/casconetext.html.

[18] MCCORMAC, Jon. *Computers and Creativity*. New York: Springer, 2012. 440 s. ISBN 978-3642317262.

itivním uživatelským rozhraním, staly jednou z příčin masové produkce, v níž se jednotlivým hudebníkům více či méně tvůrčím způsobem daří využívat přednastavené funkce. Ve výsledku se však nelze ubránit omezením, která jsou apriorně a nevýslovně dána v rozhraních těchto programů. Při pohledu na sekvencer nebo piano-roll s dvanácti půltóny už nás skoro ani nenapadne, že by rozhraní a tedy i hudba, jež s jeho pomocí vzniká, mohla „vypadat" nějak jinak. Výše zmíněným programům chybí do určité míry modulárnost – tedy možnost jít v jejich struktuře hlouběji a přeorganizovat ji podle potřeb tvůrce.[19]

Přesto se dnes přednastavenost aplikací nejeví jako problematická, ale jako samozřejmá. Vždyť je to vlastně „přirozený" stav věcí – nůž, klarinet i boxovací pytel v sobě přeci také nesou charakteristický způsob užití. Celá problematika a její řešení se, jak se zdá, přesouvá do hlavy toho kterého tvůrce. Spokojí se s onou danností, nebo nechá svou hlavu fungovat jako generátor možností, díky nimž se přednastaveného nástroje chopí novým způsobem? Nebo ještě silněji: rozhodne se vytvářet své vlastní rozhraní a nástroje? Pro postdigitálního umělce hledajícího jedinečnost výrazu je problematizace přednastavenosti jednou z cest. Tvůrce se s přednastaveností nemůže spokojit, protože mu jde právě o respektování jedinečnosti a chce jí ve svém díle dostát. Navzdory tomu, že náš druh spojují jisté podobnosti (ve fyziognomii, v modelech myšlení), je každý z nás originálem a v kontaktu s přednastavením se tato originalita nevýslovně redukuje. Vyhrocená interpretace by dokonce přednastavení jakéhokoliv rozhraní mohla podávat jako něco, co je vždy založeno na zjednodušení a zobecnění, a tedy jako něco, čemu jedinečnost té které osoby vzdoruje. Jedinečnost může být zachována v modulárním prostředí nebo v prostředí s doširoka otevřenými možnostmi. Takovými jsou programovací jazyky jako Pd, Max/MSP, SuperCollider, Processing atd.[20]

Další podstatný rozdíl spočívá v kontextu, v němž se ono vytváření nástrojů děje. Casconeho větu o tom, že poselstvím je nástroj, je třeba číst prizmatem stavu, ve kterém se nachází dnešní společnost. Jednak je to společnost plná simulaker a jednak je to společnost radikálně přetechnizovaná.

Proti digitálně manipulované a „dokonalé" tváři modelky, která nikdy neexistovala, staví postdigitál *glitchový* portrét – obraz rozbitý a na hranici čitelnosti. A proti technice, kterou dnes většina uživatelů vnímá jako pandořinu skříňku nebo „black-box", pak postdigitál staví poučený a kritický postoj k nástrojům. Ten se demonstruje například tak, že umělec svůj nástrojový repozitář a metody vytvá-

[19]Na tento nedostatek ostatně Ableton již reagoval tím, že do svého prostředí implementoval možnost vkládání modulárního prostředí, jakým je Max/MSP.

[20]Tomuto odstavci není nutné rozumět tak, že se týká jen hardware, software a věcí. Přednastavenost je pojem, který má také společenské, kulturní a politické konotace.

ření učiní v díle zjevnými.[21] Radikálnější interpretace by pak mohla ve zkratce znít tak, že z onoho *jak* se dělá *co* – tedy že prostředky sloužící k vytvoření díla se samy stávají obsahem. Domnívám se, že v případě postdigitálních tendencí nejde o vyprázdněné a formální gesto, ale že v sobě, díky kontextu, nese význam.

V praxi postdigitálního umělce, například právě skrze konstrukci nástrojů, které pak v rámci audiovizuální jam session používá, se tak odehrává pomyslné odstranění vnímání nástroje jako „black-boxu". Předpokladem pro konstrukci nástroje je totiž existence schématu, a také určitého porozumění či znalosti. Bez těchto to prostě nejde. Praxe postdigitálního umělce je tedy současně autoedukativní činností.[22]

V souvislosti s nástrojem jako poselstvím bychom mohli uvažovat o „nové řemeslnosti", která v postdigitálních tendencích zaznívá. Vidět ji lze jako reakci na instantnost dnešních hardwarových i softwarových nástrojů. Pakliže umělec používá techniku, jíž nerozumí, je mezi nimi „vzdálenost". Umělec pracující se štětcem nebo dlátem má k těmto nástrojům „blízko" – intuitivně jim rozumí, ví jak fungují a co od nich může čekat. Ve vztahu k počítači je tento vztah mnohem komplikovanější. Jednou z cest k oné blízkosti a rozumění nástroji je naučit se používat programovací jazyk jako štětec – to má patrně asi na mysli i Wolfgang Welsch, když hovoří o tom, že by dnes každý umělec měl být tak trochu programátorem.[23]

Tendence, kterým jde o proměnu vnímání techniky (jakési pomyslné otevření a prosvětlení „black-boxu"), mají své antropologicko-filosofické konotace. Můžeme uvažovat například o umělci-programátorovi, který je ve vztahu ke svým nástrojům v odcizení a skrze svou praxi (tj. vytváření vlastních nástrojů) se toto odcizení pokouší redukovat. Romanticky řečeno: jako by se svou metodou dostával do vztahu, který má analogii v situaci malíře, který v ruce drží tak „jednoduchý" a „srozumitelný" nástroj jako štětec. Postdigitální umělec se digitálnímu „štětci" pokouší utopicky přiblížit tím, že si jej sám vytvoří. Takový nástroj pak respektuje jedinečnost toho kterého tvůrce a neredukuje ji jako nástroje již připravené.[24]

Další z možných antropologických významů postdigitálu vidím v již zmíněném

[21] Viz následující kapitola o *livecodingu*.

[22] Praxe postdigitálních umělců má také co do činění s přístupem hackerů v původním slova smyslu. Tedy ne kyberteroristů, ale spíš inovátorů a lidí, kteří mají potřebu rozumět nástrojům, se kterými zacházejí. To má svůj význam právě v přetechnizovaném světě, kdy k nástrojům zaujímáme spíš postoj důvěry.

[23] CSERES, Jozef a MURIN, Michal. *Od analógového k digitálnemu...* Banská Bystrica: Fakulta výtvarných umení, 2010. 219 s. ISBN 978-80-89078-78-3. s. 20.

[24] Ač by se umělci jako David Böhm a Jiří Franta za postdigitální umělce neoznačili, lze jejich hledání nových nástrojů a cest pro kresbu takto vidět.

přiznávání se, až adoraci chyby. Chyba je moment, který demytologizuje techniku a její spasitelnou roli. Chyba a selhání je něčím, co je nám jako kontingentním bytostem blíž než dokonalost. A je to právě chyba a nedokonalost, která je motivem pro evoluční proces k hledání nových cest a kvalit. Postdigitální umělci odmítají naivní představu pokroku a dokonalosti, jak je prezentována technooptimisty a transhumanisty, kteří techniku povyšují na prostředky spásy lidstva. Transhumanismus a technooptimismus můžeme vidět jako mutace gnoze, ve které byla spása chápána jako druh poznání – jako výkon, jehož důsledkem je osvobození.[25]

> Hans Moravec se domnívá, že bychom náš mozek měli – prostřednictvím downloadování – přenést na stroj zpracovávající data. Díky tomu by se naše myšlení mohlo osvobodit od jakékoliv stopy našeho původního těla a vůbec jakéhokoliv těla. Takto vzniklý netělesný duch by byl něčím zázračným, pokud jde o jasnost jeho myšlenek a hloubku jeho poznání.

Nemusíme uvádět protiargumenty, abychom si představili, že člověk není Baron Prášil, který sám sebe dokáže vytáhnout z bažiny za cop. Hezkou přehlídkou problémů spojených s tímto tématem je román *Město permutací* od Grega Egana.[26] Jedna ze situací, které kniha popisuje, je o tom, že chudší lidé si nemůžou dovolit rozběhnout svého digitálního ducha v reálném čase, protože nemají dost prostředků na zaplacení výpočetního potenciálu atd. Postdigitální postoj nadšení technooptimistů nesdílí, spíš má blíž k rozčarování a skepsi. Synopticky s postdigitálním postojem lze číst i esej *Umělé rajské zahrady?*[27] od Wolfganga Welsche, který se k technooptimistickým tendecím též vyjadřuje skepticky.

Akcentace momentu chyby, která se často v našich výtvorech objevuje, je prostředkem k rozvzpomenutí se na naši vlastní přirozenost a humanizaci techniky. *Glitch* může být také momentem extáze, vytržením ze stereotypu – ať už jde o hudební kompozici nebo společenské jevy.

Konečně poslední motiv, jenž bych v souvislosti s antropologickými přesahy postdigitální kultury zmínil, bychom mohli pojmenovat jako „návrat k hmotě“. Pokud je pro digitální kulturu a umění charakteristické odhmotnění, pak pro postdigitál je příznačné přiznání se k hmotě. Řada umělců v poslední dekádě „emigrovala“ z „dematerializované“ platformy k elektromechanickým prostřed-

[25] CSERES, Jozef a MURIN, Michal. *Od analógového k digitálnemu...* Banská Bystrica: Fakulta výtvarných umení, 2010. 219 s. ISBN 978-80-89078-78-3. s. 13.

[26] GREG, Egan. *Město permutací*. Brno: Návrat, 2002. 416 s. ISBN 80-7174-502-2.

[27] CSERES, Jozef a MURIN, Michal. *Od analógového k digitálnemu...* Banská Bystrica: Fakulta výtvarných umení, 2010. 219 s. ISBN 978-80-89078-78-3. s. 13-26.

kům. Svou roli v tom sehrála jistě i dostupná a uživatelsky přívětivá platforma Arduino.[28] S její pomocí lze na jedné straně stavět alternativní rozhraní pro čtení vstupních signálů počítačem a na druhé straně je s ní možné jednoduše řídit elektromechanické komponenty. Arduino umělcům, kteří nejsou inženýři, zpřístupnilo oblast technicky pokročilejších aplikací, jež jim dříve mohly být zprostředkovány pouze odborníkem.

Některé z aspektů, o nichž jsme v souvislosti s postdigitalitou hovořili jsou ilustrovatelné na díle *Machines that Almost Fall Over*[29] od Michaela Kontopoulose. Kritický potenciál postdigitálu naposled vyzvedl Florian Cramer na jednom z panelů letošního Transmediale, když razantně prohlásil, že postdigitalita sebou přináší nutnost vybrat si stranu.[30]

Machines that almost fall over, Michael Kontopoulos, 2008.

5.3 | Postdigitální „avantgarda“?

Hned z počátku by bylo možné následující úvahy znemožnit poukazem na anachronismus, které slovo avantgarda asociuje. Jako by pro ni v dnešním rozbitém a tekutém postmoderním světě nebylo místo. Pokud je ale platné naše dřívější vymezení postmoderny, je v jejím rámci určitě možné vidět „modernistické“ ostrovy, vůči kterým se různá avantgardní hnutí vymezují. Avantgardu zde zjednodušeně chápu jako určité hnutí, které se vymezuje vůči etablované praxi. Mezi atributy avantgardy bychom mohli zařadit radikalitu, fascinaci novým, potřebu formulovat své postoje formou manifestu, ambivalentní vztah fascinace-kritika techniky, a snad i určitý utopismus a naivitu. Domnívám se, že konkrétním případem, na kterém je možné avantgardnost postdigitálních tendencí ilustrovat, je praxe tzv. *livecodingu*. Předběžně jej vymezíme takto: *Livecoding* je druh vystou-

[28] *Arduino.cc* [online]. [cit. 2014-04-10]. Dostupné z: http://www.arduino.cc/. Viz také http://cs.wikipedia.org/wiki/Arduino.

[29] Machines that Almost Fall Over. In: *Vimeo* [online]. [cit. 2014-04-10]. Dostupné z: http://vimeo.com/1609126.. Kanál uživatele Michael Kontopoulos.

[30] STRNAD, Matěj. Média, umění a meze hacktivismu / Transmediale. *Cinepur*. 2014. č. 92. ISSN 1213-516X.

pení, v němž performer vytváří a modifikuje kód, jehož výstupem jsou audiovizuální data.

Livecoding je programátorsko-performativní praxe, která má svůj předobraz v akcích švédského umělce Clicka Nilsona[31] a jeho inovativním způsobu interpretace hudební partitury. Před Nilsonem existovaly přístupy, které interpretovi skladby dávaly určitou volnost ve hře (aleatorická hudba), ale nevztahovaly se na možnost modifikace partitury během hry. Nilson působnost interpreta posunul dál právě tím, že mu umožnil modifikovat partituru během interpretace. Pravidla pro tento postup shrnul v krátkém textu s názevem *Instruktážní hra pro jednoho a více hudebníků.*[32]

Dalším předobrazem je performance skupiny The Hub. Toto těleso sestávalo z několika mikropočítačů navzájem propojených do sítě, po které si hráči posílali informace, které pak ovlivňovaly chování soundsystému na tom kterém počítači. Diváci během performance mohli procházet mezi počítači a pozorovat, co se na jednotlivých monitorech odehrává.

Nyní se přesuňme do přítomnosti, k popisu standardní audiovizuální performance. Hráči obvykle sedí za svými notebooky, jsou „ponoření" do obrazovky, jejich prsty se občas dotýkají klávesnice nebo kontrolerů. Jinými slovy: performativní aspekt události a pocit z živého hraní se někam nenápadně vytrácí. Na tuto situaci reagovala specifickým manifestem skupina performerů-programátorů, která se kolem roku 2000 zformovala do hnutí TOPLAP.[33] Praxi živého audiovizuálního vystoupení inovovali o určité postupy.

Na enigmatičnost, zahalenost situace, kdy performer sedí za svým počítačem a divák nevidí, co se děje, reaguje manifest TOPLAPu[34] dvěma požadavky: *Dejte nám přístup k performerově mysli – k celému lidskému instrumentu. Tmářství je nebezpečné – ukažte nám vaše obrazovky.*[35] Součástí *livecoding* performance je tedy i projekce, na které je vidět, co hráč dělá. To můžeme interpretovat jako pokus o znovunabytí bezprostřednosti. Přeneseně řečeno jde o to, vidět kytaristu, jak hraje, nebo resp. že vůbec hraje. To, že nerozumíme kódu, který vidíme, nevadí, podobně jako nemusíme rozumět nauce o harmonii či akustice, když poslouchá-

[31] *Geocities.ws* [online]. [cit. 2014-04-11]. Dostupné z: http://www.geocities.ws/clicknilson/index.html.

[32] *Toplap.org* [online]. [cit. 2014-04-11]. Dostupné z: http://toplap.org/wiki/Click_Nilson%27s_text_piece.

[33] Jde o mnohovýznamový akronym: (Temporary | Transnational | Terrestrial | Transdimensional) Organisation for the (Promotion | Proliferation | Permanence | Purity) of Live (Algorithm | Audio | Art | Artistic) Programming).

[34] *Toplap.org* [online]. [cit. 2014-04-11]. Dostupné z: http://toplap.org/wiki/ManifestoDraft.

[35] Show Us Your Screens. In: *Vimeo* [online]. [cit. 2014-04-09]. Dostupné z: http://vimeo.com/20241649. Kanál uživatele louis mccallum.

me kytarový koncert. I navzdory tomu můžeme kvalitu hry docenit a vychutnat.

V manifestu dále čteme následující body: *Programy jsou nástroje, které mohou měnit samy sebe. Kód by měl být viditelný a slyšitelný. Livecoding není o nářadí. Algoritmy jsou myšlenky. Motorové pily jsou nářadí. Proto je někdy obtížnější si všimnout algoritmů než motorových pil.*

Druhý pražský *livecoding* v klubu Neone, 2014. Foto: Dita Havránková.

Předpřipravený nástrojový repozitář – tedy již hotové syntetizéry a software – jsou v rámci *livecoding* performance nahrazeny pouhým programovacím prostředím.[36] Proti instantnosti se zde staví počáteční „nicota“. Pokud je v *livecodingu* něco apriorně dáno, je to jen vztah umělce a možností jazyka, který používá.[37] Na počátku je projekce prázdná, a teprve s postupem času, kdy performer rozvíjí kód, se rozvíjí i … Tři tečky zde naznačují selhání pojmového aparátu. O co zde vlastně jde? Má kód v rámci *livecodingu* statut hudebního nástroje, nebo partitury? Je performer autorem nástroje, kompozice, nebo je jejím interpretem? Tento moment považuji za novum. Zdá se, že zde role autora nástroje, kompozice i interpreta spadají v jedno. Další novum spočívá v tom, že v rámci *livecoding* performance je možné kód sdílet po síti, událost tak může mít povahu rozhovoru, nebo jam-session, kdy každý z hráčů může sahat do části kódu druhého.

Zmíněnou performativně-programátorskou praxi lze vidět také v souvislosti

[36] Livecoding je realizovatelný např. v jazycích Pure Data, SuperCollider, VVVV, ChucK, Fluxus, Haskell a řadě dalších.

[37] Podobnost bychom mohli vidět například v tom, jak probíhá *slam poetry*.

s hnutím Fluxus a hudebními performancemi, jejichž jádrem byla postupná preparace nástrojů. Jen místo klavíru dnes hudebníci podrobují dekonstrukci a ohýbání kódy a programovací jazyky.

Proti statičnosti kódu staví livecoding dynamičnost, procesualitu a pomíjivost kódu. Programy vidí manifest TOPLAPu jako nástroje, které mohou samy sebe měnit. Kód je něco, co je v *livecoding* performancích podrobeno permanentní možnosti proměny až sebedestrukce. Jeden z požadavků manifestu TOPLAP, který ovšem řada členů porušuje, zní „žádné zálohy".

Již jsme zmínili tvrzení, že livetime coding není o nářadí, ale o myšlení. Zde by bylo možné vřazení *livecodingu* do postdigitálních tendencí napadnout tvrzením, že pozice TOPLAP manifestu je již dál než to, co Cascone popisuje pojmem postdigitální. Přesto se domnívám, že *livecodingu* lze v souvislosti s postdigitálními tendencemi rozumět – jednak kvůli motivu projasnění nástrojů/black-boxů s nimiž umělec zachází, a kritickému přístupu k nim a jednak kvůli motivu vytváření si svých vlastních nástrojů.

Řadu *livecoding* performancí a událostí s ním spojených lze dohledat přímo na domovské stránce sdružení TOPLAP. Živá je zejména scéna kolem jazyka SuperCollider a programovacího prostředí Fluxus. Za zmínku stojí jistě série performancí Gabora Pappa[38] a také excelence umělce Chun Leeho, kterou prokazuje při *livecodingu* s matematickými výrazy.[39] Pro zajímavost dodejme, že *livecoding* nemusí být nutně vázán na konkrétní programovací jazyk a počítač. Alex McLean vidí člověka jako krásně zkonstruovanou funkci, jež provádí *livecoding* na sobě samé. Též určité typy promlouvání, které jsou zároveň konáním (např. když říkáme: „slibuji") mají k *livecodingu* velmi blízko. Za *livecoding* performanci by bylo lze považovat například i přístup japonského umělce Tetsuo Kogawy, který během svých hudebních vystoupení vytvoří a modifikuje jednoduchý FM radiopřijímač. V budoucnu se třeba dočkáme toho, že *livecoding* programátoři budou standardní součástí symfonických orchestrů.

Snad je z předešlého čtenáři zřejmé, že téma postdigitální umělecké praxe reprezentované *livecodingem* má svou relevanci a význam. Minimálně tím, že je tuto praxi možno vidět jako druh postoje, jímž se problematizuje zdánlivá samozřejmost v přístupu k technice. Nebo v tom, že se díky postdigitální umělecké hře nevýslovně ozývají ony antropologické kostanty jako chyba a selhání, na které rádi zapomínáme.

---

[38]Hackpact. In: *Vimeo* [online]. [cit. 2014-04-10]. Dostupné z: http://vimeo.com/album/2425774. Kanál uživatele gabor papp.

[39]Pd live coding/patching. In: *YouTube* [online]. 8. 11. 2008. [cit. 2014-04-10]. Dostupné z: http://www.youtube.com/watch?v=9zKzxqN5mUI. Kanál uživatele chun lee.

5.4 | DIY a FLOSS

To, čemu říkáme umění, se obvykle neodehrává v inkubátoru nebo separované laboratoři, ale probíhá na pozadí společenských, politických a ekonomických dějů. Pokusíme se zde jen krátce nastínit širší kontext, který přesahuje hranice umění a ve kterém můžeme dění na postdigitální umělecké scéně vidět.

DIY[40] kultura je založena na dispozicích každého kterého člověka a na principu soběstačnosti v produkci výrobků. DIY lze vidět v podobném duchu jako tendence, o nichž jsme dříve hovořili v souvislosti s postdigitálem: obojí je reakcí na přetechnizovaný svět, ve kterém je člověk ve vztahu k nástrojům a věcem v (nereflektovaném) odcizení a jako pokus o utopický návrat do světa, v němž bylo vše jaksi „blízko".

Na rozdíl od postdigitálu se DIY netýká jen nástrojů a techniky. Široká definice DIY by mohla hovořit o kultuře, pro kterou je charakteristická autonomie, vymezování se vůči konzumerismu, sebevzdělávání, dobrovolná skromnost, recyklace, řemeslná zručnost a sebeprodukce jdoucí od potravin (OpenCola) přes ziny, architekturu až k hudebním nosičům.

DIY kultura tím, že v jejím zájmu stojí „jedinečný" výrobek, reprezentuje alternativu vůči ekonomickému establishmentu, který trh nasycuje danými komoditami a podmiňuje tak obecný vkus. Současně je akcentací principu recyklace gestem proti neetickým praktikám, jako je neohleduplný přístup velkých firem k životnímu prostředí.

Jedním z nosných pilířů DIY etiky je tvrzení, že každý je s to, na základě svých dispozicí a vzdělání, vykonávat danou škálu činností, jejíž výsledek bude podobný tomu, který jinak zprostředkovávají placení specialisté. Domnívám se, že toto je poněkud utopistická vize, nicméně v řadě oblastí je přeci jen naplnitelná.

Zhotovení nástroje nebo výrobku svépomocí (tím nemyslíme pouhou replikaci) předpokládá pochopení jeho principů, a to je díky studiu volně sdílených informací v řadě případů dost dobře možné, ale též časově náročné. Právě náročnost je jedním z důvodů, proč se DIY nedostává patřičného ohlasu – v souboji o přízeň stále prohrává s nenáročnou instantností výrobků dostupných na trhu. Oproti nim má však stále co nabídnout ve sféře neviditelné: určitou poezii jedinečnosti, s kterou souvisí též vnímání hodnoty věcí.

V poslední době lze ve spojitosti s DIY vypozorovat zejména rozruch kolem 3D tiskáren a možných změn, které přinesou ve změně vztahu výrobce a spotřebi-

[40] „Do It Yourself", česky kutilství – v anglickém znění však postihující širší škálu, než na kterou jsme v našem kontextu zvyklí. RATTO, Matt (ed.). *DIY Citizenship: Critical Making and Social Media*. Cambridge: MIT, 2014. 464 s. ISBN 978-0262525527.

tele.[41] Kromě základních technických součástek se 3D tisk týká potravin, architektury a také hudebních nástrojů.[42]

Druhý širší kontext, se kterým postdigitální tendence souvisí, se obecně týká vztahu software a kultury. Ze software se dnes stala zásadní vrstva, která prostupuje všechy společenské oblasti – jsou zde skupiny vytvářející určitý software a tento pak zpětně vytváří společnost.[43]

> Přidáním software do kultury byla proměněna identita toho, co kulturu tvoří.

Sociální média proměnila vnímání kategorií jako je diskuse, přátelé nebo společenská angažovanost. Stala se mocenským nástrojem, který je efektivně využíván ke sledování a kontrole mas. Google dokáže na základě analýzy vyhledávání daných hesel předpovídat chřipkovou epidemii s několikadenním předstihem před zdravotnickými organizacemi. To, že kromě farmaceutického průmyslu je zde i řada jiných oblastí, ze kterých se dají vytěžovat relevantní a zpeněžitelné informace, snad není třeba nijak zvlášť zdůrazňovat. Zdá se, že techniky data-miningu dnes v rozvržení mocenských pozic a strategií pro budoucnost hrají svou roli.[44]

Na data dennodenně protékající sítí lze aplikovat matematicko-lingvistické analýzy a na výsledcích pak stavět politické nebo ekonomické strategie. Malému množství uživatelů pomalu začíná docházet, že komunikace je druh energie, již zdarma poskytují daným společnostem. Jedním z poukazů na toto uvědomění je například manifest *Mzdy za Facebook*.[45] V kombinaci s technikami sociálního inženýrství se nám tak pomalu rýsuje vize budoucnosti, která není daleko od představ reprezentovaných v kyberpunkových románech.

Na druhou stranu jsou sociální média nástrojem v rukou uživatelů, díky kterému se dokáží efektivně organizovat a establishment kritizovat. Ze software se stalo transparentní politikum. Neexistuje něco jako neutrální software – každý nástroj v sobě nese určité stigma, způsoby užívání, ideologii – je otiskem myšlení a hodnotového systému svého tvůrce. Detailním prozkoumáním vztahu kultury a software se nebudeme zabývat, pouze zde odkážeme na relevatní studie od

[41] SHADDACK. Revoluce na stole. *Živel*. 2009, č. 30. s. 68-74. ISSN 1212-5644.

[42] *3dprintedinstruments.wikidot.com* [online]. [cit. 2014-04-11]. Dostupné z: http://3dprintedinstruments.wikidot.com/printed-instruments.

[43] *Software is the Message* [online]. [cit. 2014-04-04]. Dostupné z: http://lab.softwarestudies.com/2013/12/software-is-message-new-mini-article.html.

[44] MÜLLER, Claudio. Velký bratr tě hlídá. *Chip*. [online]. 2013, č. 11 [cit. 2014-04-10]. Dostupné z: http://www.chip.cz/dokumenty/velky-bratr-te-hlida/.

[45] STEJSKALOVÁ, Tereza. Mzdy za Facebook. *A2* [online]. 2014, č. 5 [cit. 2014-04-04]. Dostupné z: http://www.advojka.cz/archiv/2014/5/mzdy-za-facebook. ISSN 1803-6635.

kolektivu autorů, které byly shrnuty v rozsáhlé monografii *New Media Reader* a *FLOSS + Art*[46], nebo na publikace od Matthew Fullera.[47]

V souvislosti s naším tématem má smysl zmínit se i o FLOSS, neboli o *Free / Libre / Open-Source Software*, který se jeví jako alternativa k proprietálnímu software, tedy k software s uzavřeným kódem, který někdo vlastní a nelze v něm svobodně dělat úpravy. V angličtině je většinou termín *free* v názvu špatně chápán tak, že jde o software, který je zadarmo. V definici na stánkách GNU projektu se termín *free* vysvětluje takto:[48]

> Svobodný software je věcí svobody, nikoli ceny. Pro porozumění tomuto konceptu je důležité chápat „free" ve smyslu „free speech" a ne ve smyslu „free beer" … Svobodný software spočívá v tom, že uživatel je svobodný ve spouštění, kopírování, distribuování, studování, upravování a vylepšování softwaru.

Open-source software je distribuován jako otevřený zdrojový kód, do kterého je možné nahlížet a modifikovat ho, což s sebou nese řadu výhod. Pro programátory je open-source „čitelný", takže je zřejmé, co a jak daný software dělá. V open-source software se obvykle rychleji implementují bezpečnostní záplaty, což ve svém důsledku vede k větší stabilitě a bezpečnosti. Příkladem této praxe může být například operační systém Linux a jmenovitě jím je jeho distribuce Debian. V takových systémech v podstatě nenarazíte na počítačový vir nebo malware, a též riziko úspěšného útoku na server, jenž běží na Debianu, je v případě aktuálních záplat minimální. V případě proprietálního software je uživatel, co se bezpečnostních záplat a implementace nových vlastností týče, odkázán na danou společnost, jejíž reakční doba je obvykle delší, než je tomu v případě open-source komunity.

Podstatnou roli zde ale hraje také celkový étos open-source komunity, který bychom mohli přirovnat k otevřené akademii. Její členové se od sebe navzájem stále učí a sdílejí své poznání.[49] Idealisticky viděno jde o prostředí, kde jsou sdílení informací, spolupráce a vývoj těmi nejzásadnějšími hodnotami. V open-sourcové komunitě žije stejný duch, který je charakteristický i pro DIY kulturu. Vznik a vývoj nástrojů zde není motivován ekonomickým kalkulem, ale vychází z nadšení a přirozené potřeby tvůrců-uživatelů. Je zde zachována otevřenost,

[46] MANSOUX, Aymeric. *FLOSS+Art*. London: OpenMute, 2008. 320 s. ISBN 978-1906496180.

[47] FULLER, Matthew (ed.). *Software Studies: A Lexicon*. Cambridge: MIT, 2008. 400 s. ISBN 978-0262062749.

[48] *Gnu.org* [online]. [cit. 2010-04-11]. Dostupné z: https://www.gnu.org/.

[49] *Github.com* [online]. [cit. 2014-04-11]. Dostupné z: https://github.com/.

průhlednost a dostupnost kódů-nástrojů. V obou těchto světech narazíte na určitý druh náročnosti, nutnosti časové investice a studia, ale též na komunitní solidaritu tyto potíže ulehčující. A konečně: ani v jednom z těchto světů se nevyhnete setkání s celou řadou chyb.

Domnívám se, že doposud nedoceněný potenciál open-source spočívá v jeho využití při vyúce, a to na jakékoliv úrovni vzdělání. Pro uvedení do programování na úrovni základní školy je například vhodný programovací jazyk Scratch.[50] Pro výtvarné a „novo-mediální" umělce je dostupná celá škála software, která tvoří alternativu ke komerčním aplikacím. Pole rastrové a vektorové grafiky pokrývají aplikace Gimp a Inkscape; 3D grafiku, animaci, střih videa a postprodukci lze realizovat v aplikaci Blender; k editování a míchání zvuku zase poslouží software Audacity nebo Ardour. Potenciál Pure Dat je ve vyúce poměrně široký: s jejich pomocí lze projít oblasti týkající se algoritmizace, základů DSP, akustiky, nauky o harmonii atd. V případě jmenovaných software lze dobře aplikovat princip *schola ludus*, nebo jak se dnes říká: vzdělání lze pojmout jako edutainment.

Nasazení open-source software ve vzdělávání má také zcela pragmatické přesahy týkající se ušetření prostředků, které vzdělávací instituce musí jinak investovat k nákupu licencí komerčního software. Firmy distribuující operační systémy a komerční aplikace si samozřejmě dobře uvědomují, že školství je pro ně velmi dobrou živnou půdou a proto školám nabízejí výhodné EDU licence. Tím vytváří určitou závislost na svých produktech. Po opuštění vzdělávací instituce student nárok na EDU licenci ztrácí a je postaven do poněkud „tvrdší" situace – cena regulérní licence se totiž obvykle pohybuje v jiných řádech. Nad celou touto situací dělá open-source s operačním systémem Linux a adekvátním balíčkem aplikací pro danou oblast vzdělání pomyslné „undo".

Pakliže přijmeme tvrzení Kima Casconeho o tom, že poselstvím je nářadí, které umělec používá, je používání open-source v umění zároveň poukazem na určitý étos, pro který je charakteristické svobodné užívání software. Pro postdigitálního umělce, který potřebuje mít k nástrojům „blízko" a který se potřebuje v operačním systému cítit jako „doma"[51], je používání open-source jednou z optimálních variant. O relevanci a významu rozhodnutí se pro open-source v oblasti umění svědčí festivaly jako Piksel[52] nebo MakeArt[53] a okrajově také větší události typu Ars Electronica a Transmediale. Preferenci pro open-source nemusíme samozřejmě rozumět jen na softwarové úrovni, ale také obecněji, jako aktu spole-

[50] *Scratch.mid.edu* [online]. [cit. 2014-04-11]. Dostupné z: http://scratch.mit.edu/.

[51] To znamená, že si může například sám navrhnout podobu a vybavení „interiéru", ale také že si sám dokáže podle plánu navrhnout nebo opravit dané zařízení.

[52] *Piksel.no* [online]. [cit. 2014-04-11]. Dostupné z: http://piksel.no/.

[53] *Makeart.goto10.org* [online]. [cit. 2014-04-11]. Dostupné z: http://makeart.goto10.org/main/.

čenskému, ve kterém se demonstruje kritika uzavřených systémů a zdůrazňuje se v něm požadavek otevřenosti.

Na závěr této kapitoly bych rád zmínil souvislost open-source a konceptu otevřeného uměleckého díla Umberta Eca. Eco vymezuje otevřené dílo takto:

> Viděli jsme tedy, že (1) „otevřená“ díla (i ta v pohybu) jsou spjata s autorem a že (2) na vyšší úrovni existují díla, která svou organickou kompletností jsou otevřena neustálému generování vnitřních vztahů, které musí adresát odkrýt a vybrat si z nich při aktu percepce totality přicházejících stimulů. (3) Každé umělecké dílo, ať už je produktem explicitní nebo implicitní poetiky nutnosti, je účinně otevřeno potenciálně nekonečnému počtu různých čtení, v nichž získává vždy novou vitalitu a perspektivu.

Eco pojednává o otevřeném díle především v souvislosti s recipientem – otevřenost se týká možnosti a mnohosti interpretací. Ze strany autora v jeho představě však jde v podstatě o řízenou záležitost. Eco hovoří o tom, že otevřené dílo nabízí vnímateli možnost vložit se do světa, který *autor zamýšlel*.

Zdá se, že vznik uměleckého díla/kódu v prostředí open-source komunity rozšiřuje Ecův pojem otevřenosti i ze strany autora. V součinnosti řady umělců-programátorů není konec vývoje díla, ani to, že se během vývoje objeví nové světy, jasně ohraničeno.

5.5 | Ohlédnutí

Ohlédnout se je možné za tím, co máme již za sebou. Ohlédnutí se je také příležitostí k vidění daných věcí z odstupu a v kritickém světle. Pokud bychom se zde o něco takového měli pokusit, měly by zaznít následující motivy:

Je otázkou, zda je ještě vůbec aktuální hovořit o postdigitálu a jeho souvislosti s postmodernou. Někteří teoretikové vidí postmodernu již jako minulý myšlenkový směr. O jejím pomyslném konci by mohla svědčit výstava s názvem *Posmoderna: Styl a subverze 1970-1990*, která proběhla na přelomu roku 2011/12 v Londýnském V&A muzeu. Stejně tak jako Cascone svůj esej otevřel větou o konci digitální éry, měla by nyní v textu zaznít proklamace o konci postdigitálu a nová vize vytvářející jasnou hranici. Tak jednoduché to ale není, protože způsoby dřívějšího myšlení a jednání se promítají i do přítomnosti. Byla by též škoda opouštět něco, čeho význam ještě nebyl patřičně zhodnocen.

Navzdory úsilí teoretiků o vydobytí pojmu, jímž by přítomné dění bylo zastřešeno, se zdá, že svého Lyotarda naše doba doposud neobjevila. Pojmy jako digimoderna nebo post-postmoderna se nedočkaly obecně sdíleného užívání. Pakli-

že bych se měl k této problematice vyjádřit, hovořil bych o informační totalitě[54] a interpretoval bych postdigitál prizmatem Puckettova tvrzení, že poslední dvě dekády patřily tvůrcům systémů (můžeme číst: nástrojů) – a to nejen hudebních, ale i komunikačních. Co z jejich užívání vzniká, se již pomalu ukazuje. Postdigitál byl zaostřen kriticky na nástroj jako takový. Demytologizoval bezchybnost nástrojů a rozkryl možnost zaujetí nového a střízlivějšího postoje. Jestliže v sobě každý nástroj nese určité způsoby užívání a jiné vylučuje – jestliže v sobě nese nějakou otevřenost a uzavřenost – nějaká přednastavení a apriori, pak postdigitál na tento banální a zapomenutý fakt znovu poukázal.

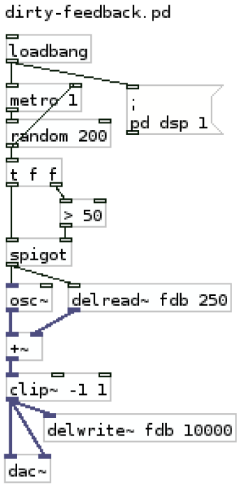
Tady a teď – kdy boti vytěžují data z komunikace na sociálních sítích, kdy jsou tisíce zombie-počítačů součástí dalšího DDoS útoku, kdy je internetová síť rozparcelována vládami a nadnárodními společnostmi – může postdigitální náhled přispět k zvažování, pro jaké nástroje se rozhodnout a jak je používat.

Kritické ohlédnutí za postdigitálem by se mělo dotknout také problematiky exkluzivnosti – jakési sebeuzavřenosti a esoteričnosti této scény, do které je pro vnějšího pozorovatele obtížné vstoupit a vůbec pochopit problémy a fascinace tématy, která se zde řeší. Textů, které by zhodnocovaly a rozvíjely postdigitální přesahy v širších společensko-kulturních souvislostech je poskrovnu. Aspoň částečně jsme se o to pokusili v předešlých kapitolách. Celé téma by si ale zasloužilo mnohem detailnější průzkum. Předchozí text snad poskytuje dostatečné základní pilíře, na nichž tento průzkum může stát.

Dále by bylo lze podotknout, že sama postdigitální praxe je vlastně již etablovaná – a to dokonce na akademické půdě – nic nového aktuálně neřeší a její avanatgardnost je tedy passé. Postdigitál si v rámci fylogeneze prošel fází od infantilní fascinace nástrojem k „dospělé“ fascinaci kódem a významem. Tento proces je ostatně často vypozorovatelný i v rámci ontogeneze.

Pokud je postdigitální éra již za námi, lze její aktuální a smyslupně prodloužené linie vidět například v budování dočasných autonomních zón na síti, které jsou za anonymizačním systémem TOR, v přechodu na ne-inflační a decentralizované virtuální měny nebo v projektu Hackerspace Global Grid, kterému jde o vybudování sítě nad internetem, jež by byla mimo dosah dnešních trendů, jakými jsou sociální inženýrství, data-mining, big-data analysis, nebo deep packet inspection. Jinou prodlouženou linii lze vidět v proměně postdigitálních tendencí do ekodigitálních. V nich se akcentuje prozkoumávání vztahu techniky k životnímu prostředí a hledání nových cest pro nové polidštění mechanizované duše.

[54] V informační totalitě již autoritou, která se snaží dohlížet a kontrolovat aspekty veřejného i soukromého života, není stát, ale nadnárodní společnost vlastnící data uživatelů a účinné prostředky k jejich analýze.


dirty-feedback.pd
loadbang
;
pd dsp 1
metro 1
random 200
t f f
>= 50
spigot
osc~
delread~ fdb 250
+~
clip~ -1 1
delwrite~ fdb 10000
dac~

6 | Outro

Pictures came and broke your heart, we can't rewind we've gone too far…

Video Killed the Radio Star

Dospěli jsme v rukověti na její konec – na místo, ze kterého lze její jednotlivé kapitoly nahlédnout a vidět je v souvislostech nebo nad jejich výběrem a řazením vznášet otázky.

Kapitola *Předpoklady* je zjednodušeným přehledem vývoje techniky, která předcházela vzniku osobního počítače. Techniku jsme zde chápali v širším slova smyslu, takže kromě hmotných nástrojů jsme se zmínili také o formalizaci, algoritmizaci a jejich možném užití v umění. Kapitola končí u vzniku konceptu počítače, jak ho známe dnes. Navzdory redukci je tato kapitola dostačující propedeutikou, která otevírá rozumění počítači jako nástroji v širším historickém kontextu. Pokusili jsme se v ní zohlednit postdigitální perspektivu, v níž je nástroj poselstvím a ve které se k nástrojům stavíme kriticky a též s požadavkem porozumění.

Kapitola *Rukověť* je v úvodu „časovým i prostorovým“ zúžením předchozí kapitoly. Sledovali jsme v ní již pouze bezprostřední předpoklady vývoje programovacího jazyka Pure Data, abychom posléze přešli k ryze praktické a „řemeslné“ části. Tato kapitola slouží jako úvod do programování v Pure Datech, a je tedy rukovětí v pravém slova smyslu, po které může čtenář/student vždy v případě potřeby sáhnout. Kromě obecných základů jsou v ní představeny i postupy vytváření audiovizuálních nástrojů. Pure Data jsme pracovně vymezili jako nářadí k vytváření nástrojů, které může postdigitálnímu umělci dobře posloužit v realizaci jeho vizí. Dodejme, že zmíněné vymezení Pure Dat i celá rukověť souvisí s vědomým zohledněním postdigitální perspektivy, ve které hraje „nástrojařina“ a sebevzdělávání v duchu *Schola ludus* svou nezanedbatelnou roli. Rukověť je označena jako *0.1a*, takže je otevřena dalším rozšířením.

V navazující kapitole s názvem *Aplikace* jsme představili jedenáct autorů, jejich umělecká díla, osobní pohledy i hodnocení možností a významu Pure Dat. Smyslem této kapitoly bylo jednak demonstrovat škálu výrazů, které Pure Da-

ta zprostředkovávají, a jednak byla příležitostí k ohlednutí se za nedávným děním na světové i československé audiovizuální scéně. Ani zdaleka jsme nezmínili všechna díla, která by si to zasloužila – pestrost „rejstříků“ Pure Dat jsme snad ale představili dostatečně.

Konečně poslední kapitola, jak napovídá už její název, je uvedením do souvislostí postdigitality. Nejprve jsme pojem postdigitálu podali tak, jak je prezentován v eseji *Estetika selhání* od Kima Casconeho, a poté jsme se zamýšleli nad příčinami, motivy a významy, které v sobě postdigitalita nese. Ohniskem úvah byly antropologické pojmy jako chyba a blízkost. Pokusili jsme se vystihnout étos, který je pro postdigitál charakteristický, a proto jsme se zmiňovali zejména o DIY, FLOSS a kritickém postoji k technice.

Ve všech kapitolách – čtenář nechť posoudí, zda více nebo méně úspěšně – jsme se pokusili zohlednit postdigitální perspektivu. Tím, že se textem prochází od historických předpokladů, přes programování až k zamyšlení se nad předkládanou problematikou, se vytváří určitý typ blízkosti a zároveň možnost kritického odstupu od tématu. Pakliže si je čtenář po přečtení rukověti obojího vědom, splnila tato svůj účel.

Zároveň musíme dodat, že podtitul rukověti není nijak exkluzivní – svou funkci může dobře sehrát nejen v rukou postdigitálního umělce, ale může posloužit komukoliv jako propedeutika, která uvádí do jedné z oblastí audiovizuálního umění, jeho charakteru a témat, jež byla aktuální během posledních dvou dekád.

Doufám, že kromě uvedení do problematiky Pure Dat, zprostředkovala takto rukověť čtenáři také dostatečné množství odkazů a asociací, které mu mohou dobře posloužit v rozehrání vlastních úvah o funkci nástrojů v umění a současné roli umělců-programátorů. Přeji čtenáři na cestě v prodloužených myšlenkových liniích naší práce, která tady končí, hodně zdaru!

Bibliografie

Tištěné zdroje:

ABELSON, Harold. *Structure and Interpretation of Computer Programs*. Cambridge: MIT, 1996. 684 s. ISBN 978-0262510875.

ALTENA, Arie. *The Aelectrosonic*. Amsterdam: Sonic Acts Press, 2011. 62 s. ISBN 978-90-810470-0-5.

AUMONT, Jacques. *Obraz*, Praha: Akademie múzických umění, 2005. 328 s. ISBN 80-7331-045-7.

BENSON, Dave. *Music: A Mathematical Offering*. Cambridge: Cambridge University Press, 2006. 423 s. ISBN 978-0521619998.

BOULANGER, Richard. *The Audio Programming Book*. Cambridge: MIT, 2011. 889 s. ISBN 978-0-262-01446-5.

CAGE, John. *Silence*. Praha: Tranzit, 2010. 275 s. ISBN 978-80-87259-07-8.

COLLINS, Nicolas. *Handmade Electronic Music. The Art of Hardware Hacking*. London: Routledge, 2009. 360 s. ISBN 978-0415998734.

COXX, Geoff. *Speaking Code: Coding as Aesthetic and Political Expression*. Cambridge: MIT, 2012. 168 s. ISBN 978-0262018364.

CSERES, Jozef a MURIN, Michal. *Od analógového k digitálnému...*. Bánská Bystrica: Fakulta výtvarných umení, 2010. 219 s. ISBN 978-80-89078-78-3.

COPE, David. *Techniques of the Contemporary Composer*. New York: Schrimer Books, 1997. 250 s. ISBN 0-02-864737-8.

CROMBIE, David. *New Complete Synthesizer*. London: Omnibus Press, 1986. 112 s. ISBN 978-0711907010.

DEVLIN, Keith. *Jazyk matematiky*. Praha: Argo, 2002. 343 s. ISBN 80-7203-470-7.

DVOŘÁK, Tomáš (ed.). *Kapitoly z dějin a teorie médií*. Praha: Akademie výtvarných umění v Praze, 2010. 349 s. ISBN 978-80-87108-16-1.

ERICKSON, Jon. *Hacking – umění exploitace*. Brno: Zoner Press, 2009. 544 s. ISBN 978-80-7413-022-9.

FARNELL, Andy. *Designing Sound*. Cambridge: MIT, 2010. 688 s. ISBN 978-0262014410.

FULLER, Matthew (ed.). *Software Studies: A Lexicon*. Cambridge: MIT, 2008. 400 s. ISBN 978-0262062749.

FULLER, Matthew. *Evil Media*. Cambridge: MIT, 2012. 232 s. ISBN 978-0262017855.

GLEICK, Jaems. *Informace*. Praha: Argo, 2013. 397 s. ISBN 978-80-257-0901-6.

GODDARD, Michael (ed.). *Reverberations: The Philosophy, Aesthetics and Politics of Noise*. London: Bloomsbury Publishing, 2012. 287 s. ISBN 978-1-4411-6065-2.

GOLDSMITH, James a WU, Tim. *Kdo řídí internet?*. Praha: Dokořán, 2008. 271 s. ISBN 978-80-7363-184-0.

GOODMAN, Nelson. *Jazyky umění*. Praha: Academia, 2007. 212 s. ISBN 978-80-200-1519-8.

GUŠTAR, Milan. *Elektrofony I*. Praha: Uvnitř, 2007. 397 s. ISBN 978-80-239-8446-0.

GUŠTAR, Milan. *Elektrofony II*. Praha: Uvnitř, 2008. 518 s. ISBN 978-80-239-8447-7.

GRAHAM, Paul. *Hackers & Painters: Big Ideas from the Computer Age*. Sebastopol: O'Reilly Media, 2010. 272 s. ISBN 978-1449389550.

GRAYSON, John. *Sound Sculpture*. Vancouver: A.R.C., 1975. 196 s. ISBN NB1272.G739.

HAYLES, Katherine. *My Mother Was a Computer*. Chicago: The University of Chicago Press, 2005. 278 s. ISBN 0-226-32148-7.

HEIDEGGER, Martin. *Básnicky bydlí člověk*. Praha: OIKOYMENH, 1993. 187 s. ISBN 80-85241-40-4.

HRUŠKA, Bohuslav. *Pod babylónskou věží*. Praha: Práce, 1987. 374 s. ISBN 24-048-87.

HUSSEY, Edward. *Presokratici*. Praha: Rezek, 1997. 213 s. ISBN 80-86027-07-4.

CHOCHOLOVÁ, Lucie (ed.). *ITC a současné umění ve výuce.* Praha: Národní Galerie, 2008. 156 s. ISBN 978-80-7035-378-3.

IGOE, Tom. *Making Thinks Talk*. Sebastopol: O'Reilly Media, 2007. 426 s. ISBN 978-0-596-51051-0.

KAKU, Michio. *Hyperprostor*. Praha: Argo, 2008. 324 s. ISBN 978-80-257-0013-6.

KANDINSKY, Wassily. *O duchovnosti v umění*. Praha: Triáda, 2009. 134 s. ISBN 978-80-87256-08-4.

KOFROŇ, Petr a SMOLKA, Martin. *Grafické partitury a koncepty*. Olomouc: Votobia, 1996.

KOTRUBENKO, Viktor. *Tajemství syntezátorů.*. Praha: Editio Suprahon, 1987. ISBN 09/22 02-044-88.

KNUTH, Donald. *Umění programování, 1. díl*. Brno: Computer Press, 2008. 648 s. ISBN 978-80-251-2025-5.

KOMÁREK, Stanislav. *Příroda a kultura*. Praha: Vesmír, 2000. 180 s. ISBN 80-85977-33-8.

KREIDLER, Johannes. *Loadbang*. Hofheim am Taunus: Wolke Verlag, 2009. 280 s. ISBN 978-3-95593-055-4.

LEAVITT, David. *Muž, který věděl příliš mnoho*. Praha: Argo, 2007. 270 s. ISBN 978-80-7363-086-7.

LÉVY, Pierre. *Kyberkultra*. Praha: Karolinum, 2000. 229 s. ISBN 80-246-0109-5.

LIVIO, Mario. *Zlatý řez*. Praha: Argo, 2006. 255 s. ISBN 80-7203-808-7.

LOY, Gareth. *Musimathics I.*. Cambridge: MIT, 2006. 482 s. ISBN 978-0-262-12282-5.

LOY, Gareth. *Musimathics II.*. Cambridge: MIT, 2007. 562 s. ISBN 978-0-262-12285-6.

LYON, Eric. *Designing Audio Objects for Max/MSP and Pd* . Middleton: A-R Editions, 2012. 340 s. ISBN 978-0895797155.

MCCORMAC, Jon. *Computers and Creativity*. New York: Springer, 2012. 440 s. ISBN 978-3642317262.

MANSOUX, Aymeric (ed.). *FLOSS + Art*. London: OpenMute, 2008. 320 s. ISBN 1906496188.

MAREK, Rudolf. *Assembler pro PC*. Brno: Computer Press, 2007. 228 s. ISBN 80-7226-843-0.

MAREŠ, Milan. *Slova, která se hodí*. Praha: Academia, 2006. 348 s. ISBN 80-200-1445-5.

NAUMANN, Friedrich. *Dějiny informatiky*. Praha: Academia, 2009. 422 s. ISBN 978-80-200-1730-7.

NIERHAUS, Gerhard. *Algorithmic Composition*. Wien New York: Springer, 2009. 287 s. ISBN 978-3-211-77539-6.

NOBLE, Joshua. *Programming Interactivity: A Designer's Guide to Processing, Arduino, and Openframeworks*. Sebastopol: O'Reilly Media, 2009. 736 s. 978-0596154141.

OPPENHEIM, Alan. *Discrete-time Signal Processing*. Upper Saddle River: Prentice Hall, 1999. 870 s. ISBN 0-13-754920-2.

PEREGRIN, Jaroslav. *Člověk a pravidla*. Praha: Dokořán, 2011. 166 s. ISBN 978-80-7363-347-9.

PEKELIS, Viktor. *Malá encyklopédia kybernetiky*. Bratislava: Mladé letá, 1981. 309 s. ISBN 66-226-81.

PEŠEK, Kryštof. *Processing Beta*. Praha: Akademie múzických umění, 2013. 152 s. ISBN 978-80-7331-224-4.

PINKER, Steven. *Slova a pravidla*. Praha: Academia, 2008. 455 s. ISBN 978-80-200-1641-6.

PUCKETTE, Miller. *The Theory and Technique of Electronic Music*. Singapore: World Scientific Publishing Company, 2007. ISBN 978-9812700773.

ROADS, Curtis. *Microsound*. Cabridge: MIT, 2004. 392 s. ISBN 978-0262681544.

ROADS, Curtis. *The Computer Music Tutorial*. Cambridge: MIT, 1996. ISBN 978-0262680820.

RUSNÁKOVÁ, Katarína. *V toku pohyblivých obrazov*. Bratislava: Vysoká škola výtvarných umení v Bratislave, 2005. 193 s. ISBN 80-88675-97-9.

RUSS, Martin. *Sound Synthesis and Sampling*. Waltham: Focal Press, 2008. 568 s. ISBN 978-0240521053.

SHIFFMAN, Daniel. *The Nature of Code*. Daniel Shiffman, 2012. 498 s. ISBN 0985930802.

SMITH, Steven. *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego: California Technical Publishing, 1999. 650 s. ISBN 0-9660176-7-6.

SOKOL, Jan. *Jak počítá počítač*. Praha: SNTL, 1977. 110 s. ISBN L26-A-IV-31f/52254.

SOUSEDÍK, Prokop. *Logika pro studenty humanitních oborů*. Praha: Vyšehrad, 2001. 224 s. ISBN 80-7021-509-7.

SVOBODA, Vladimít. *Páni inženýři*. Praha: SNDK, 1965. 221 s. ISBN 10-50159.

SYROVÝ, Václav. *Hudební akustika*. Praha: Akademie múzických umění, 2008. 440 s. ISBN 978-80-7331-127-8.

TÖPFER, Pavel. *Algoritmy a programovací techniky*. Praha: Prometeus, 2007. 300 s. ISBN 978-80-7196-350-9.

TUGENDHAT, Ernst a WOLF, Ursula. *Logicko-sémantická propedeutika*. Praha: Rezek, 1997. 237 s. ISBN 80-86027-02-3.

WARDRIP-FRUIN, Noah. *The New Media Reader*. Cambridge: MIT, 2003. 840 s. ISBN 978-0262232272.

WELSCH, Wolfgang. *Naše postmoderní moderna*. Praha: Zvon, 1994. 200 s. 80-7113-104-0.

WISHART, Trevor. *Audible Design: A Plain and Easy Introduction to Sound Composition*. York: Orpheus The Pantomime, 1994. 138 s. ISBN 978-0951031315.

WITTGENSTEIN, Ludwig. *Tractatus logico-philosophicus*. Praha: OIKOYMENH, 2007. 87 s. ISBN 978-80-7298-284-4.

WRÓBLEWSKI, Piotr, Algoritmy. Brno: Computer Press, 2004. 351 s. ISBN 80-251-0343-9

XENAKIS, Iannis. *Formalized Music: Thought and Mathematics in Composition* . New York: Pendragon Press, 1992. 490 s. ISBN 978-1576470794.

ZIMMER, Fränk (ed.). *Bang Pure Data*. Hofheim am Taunus: Wolke Verlag, 2005. 176 s. ISBN 978-3-936000-37-5.

ŽEGIN, Lev Fjodorovič. *Jazyk malířského díla*. Praha: Odeon, 1980.

Elektronické zdroje:

Aprja.net [online]. [cit. 2014-04-10]. Dostupné z: http://www.aprja.net/.

Ccrma.stanford.edu [online]. [cit. 2014-04-10]. Dostupné z: https://ccrma.stanford.edu/.

Codehop.com [online]. [cit. 2014-04-10]. Dostupné z: http://codehop.com/.

Criticalartware.net [online]. [cit. 2014-04-10]. Dostupné z: http://criticalartware.net/.

Dangermouse.net [online]. [cit. 2014-04-10]. Dostupné z: http://www.dangermouse.net/.

Danomatika.com [online]. [cit. 2014-04-10]. Dostupné z: http://danomatika.com/code/.

Designingsound.org [online]. [cit. 2014-04-10]. Dostupné z: http://designingsound.org/.

Faust.grame.fr [online]. [cit. 2014-04-10]. Dostupné z: http://faust.grame.fr/.

Flossmanuals.net [online]. [cit. 2014-04-10]. Dostupné z: http://en.flossmanuals.net.

Generative.net [online]. [cit. 2014-04-10]. Dostupné z: http://generative.net/.

Gridflow.ca [online]. [cit. 2014-04-10]. Dostupné z: http://gridflow.ca/.

Hackaday.com [online]. [cit. 2014-04-10]. Dostupné z: http://hackaday.com/.

Iim.cz [online]. [cit. 2014-04-10]. Dostupné z: https://www.iim.cz/.

Ircam.fr [online]. [cit. 2014-04-10]. Dostupné z: http://www.ircam.fr/.

Ixi-audio.net [online]. [cit. 2014-04-10]. Dostupné z: http://www.ixi-audio.net/.

Katjaas.nl [online]. [cit. 2014-04-10]. Dostupné z: http://www.katjaas.nl/.

Makeart.goto10.org [online]. [cit. 2014-04-10]. Dostupné z: http://makeart.goto10.org/.

Metamute.org/ [online]. [cit. 2014-04-10]. Dostupné z: http://www.metamute.org/.

Microsound.org [online]. [cit. 2014-04-10]. Dostupné z: http://microsound.org/.

Monoskop.org [online]. [cit. 2014-04-10]. Dostupné z: http://monoskop.org.
Msp.ucsd.edu [online]. [cit. 2014-04-10]. Dostupné z: http://msp.ucsd.edu/.
Noizehack.com [online]. [cit. 2014-04-10]. Dostupné z: http://noizehack.com/.
Pawfal.org [online]. [cit. 2014-04-10]. Dostupné z: http://www.pawfal.org.
Pd-graz.mur.at [online]. [cit. 2014-04-10]. Dostupné z: http://pd-graz.mur.at/.
Pd-tutorial.com [online]. [cit. 2014-04-10]. Dostupné z: http://www.pd-tutorial.com/.
Puredata.hurleur.com [online]. [cit. 2014-04-10]. Dostupné z: http://puredata.hurleur.com.
Puredata.info [online]. [cit. 2014-04-10]. Dostupné z: http://puredata.info/.
Runme.org [online]. [cit. 2014-04-10]. Dostupné z: http://runme.org/.
Steim.org [online]. [cit. 2014-04-10]. Dostupné z: http://steim.org/.
Synthtopia.com [online]. [cit. 2014-04-10]. Dostupné z: http://www.synthtopia.com/.
Toplap.org [online]. [cit. 2014-04-10]. Dostupné z: http://toplap.org/.
Ubuntustudio.org [online]. [cit. 2014-04-10]. Dostupné z: http://ubuntustudio.org/.
Uvnitr.cz [online]. [cit. 2014-04-10]. Dostupné z: http://www.uvnitr.cz/.
Wired.com [online]. [cit. 2014-04-10]. Dostupné z: http://www.wired.com/.

Michal Cáb

Pure Data: Rukověť postdigitálního umělce, ver. 0.1a

Sazbu programem XeLaTeX připravil autor.

S pomocí open-source aplikací Gedit, Gimp, Inkscape, MuseScore, Pure Data a operačního systém GNU/Linux.