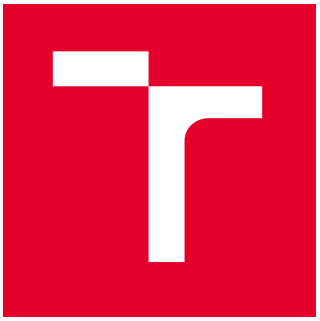


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

SONIFIKACE VIDEA POMOCÍ TECHNIK UMĚLÉ INTELIGENCE

ARTIFICIAL INTELLIGENCE FOR VIDEO SONIFICATION

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Filip Dobrocký**

AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Kamil Říha, Ph.D.**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Bc. Filip Dobrocký ***ID:*** 212553
Ročník: 2 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Sonifikace videa pomocí technik umělé inteligence

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je využití umělé inteligence pro automatickou tvorbu zvuků na základě obrazového obsahu videozáznamu a videa snímaného v reálném čase. V rámci diplomové práce naprogramujte detektor a klasifikátor základních objektů vyskytujících se v záběrech snímajících reálné nebo fotorealistické scény. Ve vhodném prostředí navrhněte pokročilé algoritmické propojování výsledků detektoru a vlastních zvukových modulů umožňujících přehrávání vhodných zvukových vzorků, zvukovou syntézu či generování hudebního materiálu na základě obrazových dat s cílem umělecké sonifikace videosekvence.

DOPORUČENÁ LITERATURA:

[1] GONZALEZ, R. C.; WOODS, R. E.: Digital Image Processing, Prentice Hall, New Jersey, 2002.
[2] BRADSKI, G.; KAEHLER, A.: Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, Inc. USA 2008, ISBN: 978-0-596-51613-0.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 19.5.2023

Vedoucí práce: doc. Ing. Kamil Říha, Ph.D.

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

ABSTRAKT

Táto práca sa zaoberá sonifikáciou videa – prevodom obrazu na zvuk. Jej cieľom je využitie moderných techník počítačového videnia založených na umelej inteligencii pre vytvorenie systému schopného algoritmickej tvorby zvuku použiteľného v umeleckom kontexte. Sústredí sa na oblasti sound artu, algoritmickej kompozície a generatívnej hudby. Súčasťou práce je implementácia modulárneho sonifikačného systému v jazyku Python využívajúceho moderný detektor objektov YOLOv7 spolu s algoritmom pre sledovanie viacerých objektov z knižnice Norfair. Princíp je založený na systematickom prideľovaní zvukových objektov sledovaným objektom vo videu. Zvuk je tvorený prostredníctvom platformy SuperCollider a jej API pre Python s názvom Supriya, využívajúc rozličné typy zvukovej syntézy spolu s automatizovane vytvorenou databankou zvukov.

KĽÚČOVÉ SLOVÁ

sonifikácia, interaktívna hudba, algoritmická kompozícia, sound art, počítačové videnie, umelá inteligencia, detekcia objektov, sledovanie objektov, YOLO, SuperCollider

ABSTRACT

This thesis deals with the topic of video sonification – the transformation of image into sound. It aims to use state-of-the-art techniques of computer vision based on artificial intelligence to create a system capable of algorithmic sound creation applicable in the art context. The focus is put on the fields of sound art, algorithmic composition and generative music. The thesis includes an implementation of a modular sonification system which utilizes the modern object detector YOLOv7 along with a multiple object tracking algorithm (implemented in the library Norfair), built using the programming language Python. The fundementals of the system lie in systematic assignment of sound objects to objects tracked in the video. The sound creation relies on the SuperCollider platform using the Python API Supriya, incorporating various methods of sound synthesis along with a programmatically created sound database.

KEYWORDS

sonification, interactive music, algorithmic composition, sound art, computer vision, artificial intelligence, object detection, multiple object tracking, YOLO, SuperCollider

DOBROCKÝ, Filip. *Sonifikace videa pomocí technik umělé inteligence*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 68 s. Diplomová práca. Vedúci práce: doc. Ing. Kamil Říha, Ph.D.

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora:	Bc. Filip Dobrocký
VUT ID autora:	212553
Typ práce:	Diplomová práca
Akademický rok:	2022/23
Téma závěrečnej práce:	Sonifikace videa pomocí technik umělé inteligence

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce doc. Ing. Kamilovi Říhovi, Ph.D. za ochotu viesť túto prácu, podnetné návrhy a otvorenosť mojim nápadom.
Vďaka patrí aj spolku SVITAVA – transmedia art lab, vďaka ktorému som si v rámci projektu Trychtýř v praxi vyskúšal tvorbu interaktívneho umenia, čo posunulo aj výsledky tejto práce.

Obsah

Zoznam obrázkov

Úvod

Transformácia obrazu na zvuk je dlho skúmanou problematikou – videnie a počutie sú úzko prepojené vnemy. Výpočtová technika umožňuje mnoho spôsobov algoritmickej interpretácie obrazu. Možnosti sú však výrazne rozšírené rozvojom počítačového videnia v posledných rokoch, ktorý je úzko spätý s umelou inteligenciou. Vďaka nej sú počítače schopné porozumieť vizuálnej scéne na veľmi vysokej úrovni. Informácie získané z obrazu môžu byť vyjadrené pomocou zvuku za rôznymi účelmi, medzi ktoré patrí napríklad pomoc zrakovo postihnutým. Sonifikácia však ponúka aj množstvo možností využitia v umení, v ktorom je umelá inteligencia mimoriadne aktuálnou témou (ostatne ako vo väčšine iných oblastiach).

V interaktívnom umení je využitie počítačového videnia už zaužívanou technikou. Obraz tu predstavuje vstup, s ktorým interaktívne dielo komunikuje – táto komunikácia prebieha často aj prostredníctvom média zvuku. Umelecká sonifikácia obrazu nájde množstvo aplikácií v algoritmickej kompozícii, generatívnej hudbe či už ťažšie definovateľných formách sound artu.

Táto práca skúma možnosti umeleckej sonifikácie videa a zameriava sa na objektové chápanie obrazu. Jej cieľom je vytvoriť variabilný systém pre algoritmickú kompozíciu zvukovej stopy na základe techník detekcie a sledovania objektov vo videu.

Teoretická časť sa venuje problematike sonifikácie, jej využitiu vo zvukovom umení, moderným technikám počítačového videnia a objektovej tvorbe zvuku s využitím programovacích prostredí pre zvukovú syntézu. Prvá podkapitola *Sonifikácia* si kladie za cieľ priblížiť motiváciu k vzniku celej práce, pohybujúc sa zväčša v umelecko-teoretickej rovine. Časť *Počítačové videnie* vysvetľuje princípy hlavných použitých algoritmov spracovania obrazu. Ďalšia časť *Tvorba zvuku* sa už viac prakticky venuje možným prostriedkom sonifikácie obrazu, sústrediac sa na prostredie SuperCollider, ktoré bolo vybrané ako vhodným nástrojom pre realizáciu praktickej časti.

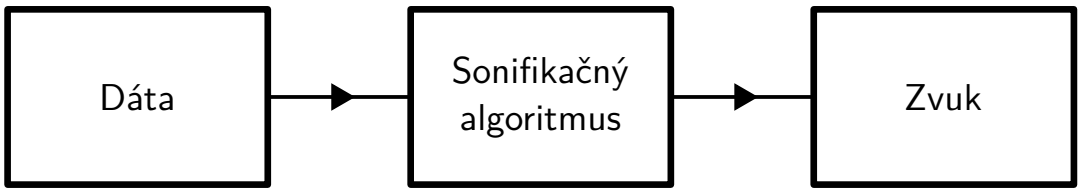
Praktická časť dokumentuje implementáciu modulárneho systému umožňujúceho sonifikáciu videa na základe objektového princípu. Je v nej vysvetlený celkový princíp systému a sú popísané aj jeho jednotlivé časti spolu so spôsobom ich implementácie. Na záver praktickej časti sú zvážené rôzne možnosti využitia vytvoreného systému.

1 Teoretická časť

1.1 Sonifikácia

Aj napriek tomu, že termín *sonifikácia* sa začal objavovať už v 90. rokoch 20. storočia, stále má ďaleko od zaužívaného pojmu a nie je jasná ani jeho definícia – za približne 30 rokov výskumu definícií vzniklo niekoľko, neraz si však vzájomne odporujú. Termín ale má svoje opodstatnenie a dá sa predpokladať, že sa stane podobne akceptovaným pojmom ako jeho obrazový ekvivalent *vizualizácia*. V rámci domácej akademickej obce sa tejto téme komplexne venoval Jiří Suchánek, ktorý po zohľadnení doterajšieho svetového výskumu navrhuje definíciu: *sonifikácia je systematická interpretácia skúmaného fenoménu pomocou zvuku.* [1]

Zdôrazňuje dôležitosť subjektivity interpretácie dát – tie môžu byť interpretované nespočetne mnoho spôsobmi a výsledný zvuk je prudko závislý na tvorcovi transformačného algoritmu, na jeho estetických preferenciách a zámere – *účele sonifikácie*. Tiež je potrebné zdôrazniť systematickosť interpretácie – výsledný zvuk a zdrojové dáta majú jednoznačne definovaný vzťah (ten však môže byť viac či menej komplexný a obtiažnosť jeho dekódovateľnosti sa môže líšiť). Autorský vstup spočíva vo vytvorení princípu transformácie dát. Samotné dáta sú objektívnym odrazom reality a autor do nich nezasahuje – potom by už nešlo o sonifikáciu.



Obr. 1.1: Reťazec sonifikácie

Suchánek navrhuje následujúce rozdelenie sonifikácie podľa jej účelu:

1. **Analytická / vedecká sonifikácia**
   Je využívaná pre vedecké účely a jej cieľom je analýza dát a skúmanie daného fenoménu prostredníctvom zvuku, ktorý umožňuje komplementárny pohľad k vizuálnej alebo textovej reprezentácii dát. Tento nástroj môže byť užitočný vďaka schopnosti sluchu vnímať drobné zmeny v opakujúcich sa vzorcoch. Niekedy tiež môže byť prostriedkom popularizácie vedy.
2. **Aplikovaná / navigačná sonifikácia**
   Spadá do kategórie designu a služieb. Slúži k orientácií a navigácii v technologických systémoch (auto, počítač, telefón, lietadlo...) – väčšinou ide o jedno-

duché signalizačné pípanie alebo krátke zvukové vzorky (*earcons*[1]).

3. **Umelecká sonifikácia**
   Cieľom je vyvolanie umeleckého dojmu. Prioritou nie je presná analýza dát, ale vytvorenie esteticky hodnotnej formy, využívanie konceptuálnych presahov, použitie zdrojových dát ako metafory, symbolu, apelu, či priameho posolstva. Systematickosť transformácie dát by však mala ostať zachovaná.

Táto práca sa zameriava na *umeleckú sonifikáciu*. Tá sa čiastočne prekrýva s pojmom *muzifikácia* (dátami riadená hudba), v prípade, že je jej cieľom hudobná forma. V opačnom prípade má umelecká sonifikácia bližšie k oblasti *sound artu* (časť 1.1.3).

Umelecká sonifikácia dovoľuje aj hybridné prístupy spojené s *algoritmickou kompozíciou*. Nemožno ju však s týmto pojmom zamieňať – pre sonifikáciu je podstatným rysom rešpektovanie zdrojových dát a reflexia reality. Algoritmická kompozícia naopak dovoľuje väčšiu slobodu narábania so vstupnými dátami aj výsledným zvukom – môže využívať sonifikáciu ako nástroj, avšak výsledná forma už sonifikáciou byť nemusí. V snahe pokryť čo najviac využití v umeleckej sfére sa táto práca venuje aj algoritmickej kompozícii a generatívnej hudbe.

1.1.1 Algoritmická kompozícia

S vývojom artificiálnej hudby 20. storočia sa začala značne komplikovať definícia hudobnej kompozície. Rodia sa skladby, ktoré by predtým vôbec neboli chápané ako hudba, ale sled zvláštnych hudobných či nehudobných zvukov. Vzniká neustála snaha oprostiť sa od tradičného chápania hudby hľadaním nových metód procesu vzniku skladby, práce s tónovým materiálom, rytmom alebo farbou.

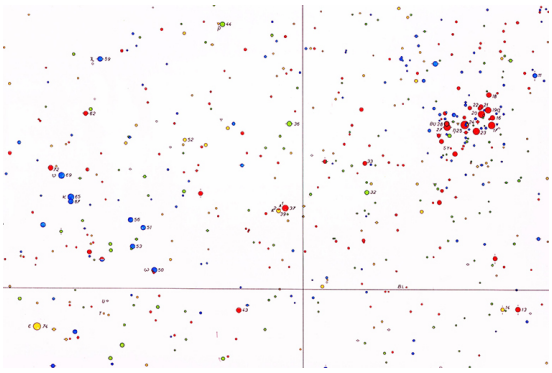
Mnoho týchto nových postupov úzko súvisí s *algoritmickou kompozíciou*, ktorá je definovaná ako tvorba hudby pomocou metodických procedúr – určitých predom definovaných postupov, ktoré môžu byť striktné (v extrémnom prípade bez ďalšieho zásahu skladateľa) alebo môžu byť zámerne ovplyvňované. Jej rysy môžeme nájsť do istej miery už v historickej hudbe – napr. kánony sú komponované pomocou opakujúceho sa melodického vzorca, alebo aj Bachove fugy sú mimoriadne systematickými operáciami.

V 20. storočí však vznikajú hudobné smery, o ktorých môžeme povedať, že sú založené na algoritmických prístupoch. *Dodekafónia*[2], ako reakcia na subjektívnu povahu romantizmu 19. storočia, prináša rád do (predtým voľnej) atonálnej hudby, používaním matematických radov ako tónového materiálu a vykonávaním kombinatorických operácií nad týmito radmi. Ešte viac algoritmický je *multiserializmus*,

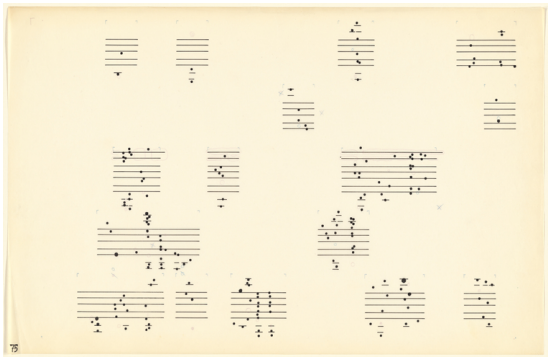
[1] Zvukový ekvivalent ikon grafických používateľských rozhraní.

[2] Tiež *dvanásťtónová hudba* – hudobný smer vytvorený Arnoldom Schönbergom na počiatku 20. storočia.

ktorý matematické rady aplikuje okrem tónovej výšky aj na ďalšie hudobné parametre, ako rytmus a farba. Naopak *aleatorická hudba* ponecháva isté parametre náhode. Je však potrebné definovať, ktoré prvky (a do akej miery) sú voľné alebo záväzné a teda ide tiež o utvorenie istých princípov / *systému*. John Cage vo svojej tvorbe dosahoval indeterminizmu mimoriadne premysleným postupom – napr. komponovanie pomocou Čínskej knihy premien (*I-ťing*), kde je „náhoda“ určená veštiacimi tabuľkami (hexagrammi). [2]



(a) Časť atlasu *1950.0*



(b) Skica partu skladby

Obr. 1.2: John Cage – Atlas Eclipticalis [3]

Niektoré Cageove skladby majú aj prvky sonifikácie – napr. skladba *Atlas Eclipticalis* (1961), ktorá vznikala prekladaním notovej osnovy cez hviezdny atlas *1950.0* Antonína Bečváře (obr. 1.2a). Proces vzniku skladby teda spočíval v transformácii atlasu na hudobnú partitúru istým systematickým postupom, avšak s podstatným autorským vstupom (selekcia a usporiadnanie hviezdnych útvarov v partitúre). Skladba je pre variabilný počet nástrojov, pričom plný počet partov je 86. Je využitý špeciálny notačný systém, ktorý dovoľuje značnú vôľu v spôsobe interpretácie, a teda ide najmä o aleatorickú skladbu. [3]

Dnes sa pod pojmom algoritmická kompozícia pochopiteľne väčšinou myslí skladba do istej miery realizovaná pomocou počítačového programu (rovnako ako pojem algoritmus sa najčastejšie spája s počítačmi). Počítače umožňujú automatizáciu metodických procedúr a teda aj generovanie hudobného materiálu – tvorbu *generatívnej hudby*.

1.1.2 Generatívna hudba

Termín *generatívna hudba* sa veľmi často pripisuje Brianovi Enovi (ktorý sa okrem iného označuje za priekopníka *ambientu*). Eno popisuje generatívnu hudbu ako hudbu, ktorá je výstupom istého systému, „nikdy sa presne neopakuje“ a „trvá večne“.

Generatívny systém nemusí nutne znamenať počítačový program – niekedy sa za generatívnu hudbu označuje aj napr. skladba *Pendulum Music* od Steva Reicha, prípadne skladba *In C* od Terryho Rileyho. Pendulum music využíva mikrofóny zavesené nad reproduktormi, ktoré sa správajú ako kyvadlá a vďaka spätnej väzbe sú tak generované tóny (v rytmoch, ktoré určuje kmitavý pohyb kyvadiel a ich vzájomné fázové posuny). In C je skladba určená pre neurčitý počet hudobníkov, ktorí si jednotlivé hudobné frázy v partitúre môžu sami poskladať (za uplatnenia určitých pravidiel) – skladba má aleatorické prvky (aleatorická a generatívna hudba ostatne toho majú mnoho spoločného). Tieto skladby však v ich bežných interpretáciách nesplňujú Enovo kritérium „večného trvania“ hudby, avšak využívajú generatívne metódy. Príklad generatívnej skladby so skoršej Enovej tvorby je *2/1*, vytvorená pomocou systému, ktorý zahŕňal rôzne dlhé slučky magnetofónovej pásky („*tape loops*“). Vďaka rozdielnej dĺžke jednotlivých slučiek by sa hudba zopakovala až za 27 dní (čo je príliš dlho na to, aby si to akýkoľvek poslucháč všimol, a teda to je možné považovať za „večnosť“). [4]

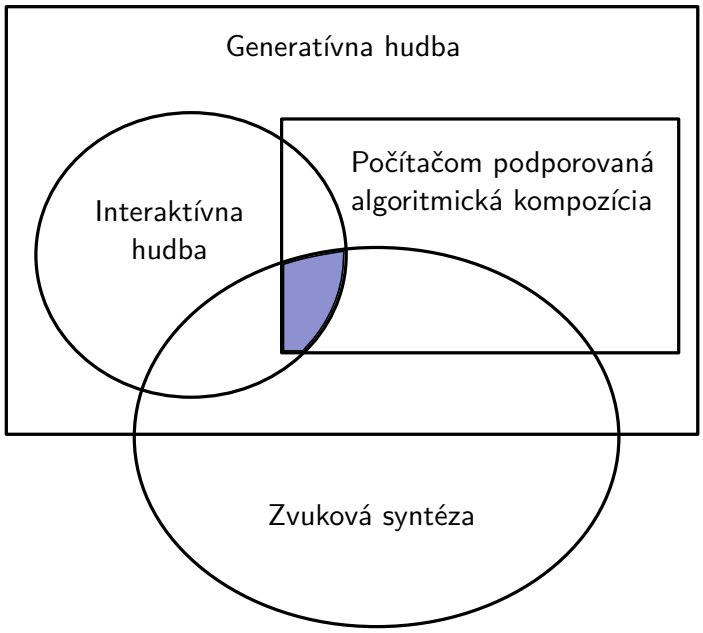
Eno poukazoval aj na limitácie hudobných médií, ktoré ho nútili hudbu tvorenú pomocou generatívnych systémov nahrávať na fixné médiá namiesto toho, aby poslucháčom poskytoval samotné systémy. Jeho album *Reflection* z roku 2017 má okrem štandardných formátov aj formu iPhone aplikácie, v ktorej je hudba priamo generovaná – aplikácia je teda jediný plnohodnotný formát albumu, zatiaľ čo štandardné nosiče obsahujú len statický úryvok celého diela. Ďalším príkladom poskytovania hudby v nestatickom, generatívnom formáte je streamovacia služba *generative.fm* ponúkajúca mnoho generatívnych hudobných systémov rôzneho charakteru a „nálady“. [5]

Generatívne stratégie môžu byť využité aj napr. na imitáciu zaužívaných žánrov populárnej hudby a často tak môžu byť výhodne aplikovateľné v komerčnej sfére. Generatívna hudba je tiež mnohokrát vhodným formátom pre hudobné stopy počítačových hier (kde býva spravidla využívaná aspoň *adaptívna* hudba a sound design).

V artificiálnej elektroakustickej hudbe môže slúžiť ako prostriedok tvorby inak len ťažko realizovateľných hudobných štruktúr. Ďalšou prednosťou môže byť konceptuálna zaujímavosť danej aplikácie generatívnej hudby:

„Soudobý skladatel je dnes často (v souladu např. s principem koláže ve výtvarném i hudebním umění) záměrně jen organizátorem do jisté míry přejatých struktur, nebo dokonce pouze „poukazovatelem“ na zajímavou ideu (viz konceptuální umění, drobné intervence do okolního prostoru, změna významy díla pouhou změnou kontextu atd.).“ [2]

Generatívna hudba je tiež predpokladom *interaktívnej* hudby – umožňuje tvorbu



Obr. 1.3: Vzťah generatívnej hudby, počítačom podporovanej algoritmickej kompozície, interaktívnej hudby a zvukovej syntézy [2]

rôznych interaktívnych zvukových inštalácií[3] alebo hudobných vystúpení.

Ako už bolo naznačené, princípy generatívnej hudby môžu byť aplikované v rámci *počítačom podporovanej algoritmickej kompozície*, využitím rôznych kompozičných programov, prípadne programovacích prostredí. Tie môžu generovať hudobné informácie resp. notový zápis (často využitím MIDI) alebo priamo produkovať samotný zvuk – realizovať *zvukovú syntézu*. Hudba môže byť generovaná v reálnom čase na základe vstupu interagujúceho subjektu (používateľa, interpreta, diváka alebo aj prostredia...) – vtedy hovoríme o *interaktívnej hudbe*.

Vzťah vyššie kurzívou vyznačených pojmov (podľa Dana Dlouhého) je ilustrovaný na obrázku 1.3. Vyznačený prienik je oblasť, kde spadajú ciele tejto práce – umelecká sonifikácia videa. Aplikácia prevádzajúca vizuálne dáta na hudbu (pomocou zvukovej syntézy) môže byť využitá ako prostriedok algoritmickej kompozície a za predpokladu, že je video vstup transformovaný v reálnom čase, aj ako prostriedok interaktívnej hudby.

Perspektívy generatívnej hudby

Na generatívnu hudbu je možné prihliadať z nasledujúcich perspektív [2][6]:

1. **Lingvistické/štrukturálne hľadisko**

[3]Tie sa však častejšie spájajú so *sound artom*, viď časť 1.1.3

Hudba vzniká na základe analytických východísk dostatočne jednoznačných, aby boli použiteľné pre vytvorenie súdržnej štruktúry. Vychádza z teórií generatívnej gramatiky a hudby, ktoré generujú materiál na základe rekurzívnych stromových štruktúr.

2. **Interaktívne/behaviorálne hľadisko**
   Hudba je generovaná predom určeným systémom bez akéhokoľvek hudobného vstupu a ďalšieho zásahu.
3. **Kreatívne/procedurálne hľadisko**
   Hudba generovaná procesom, ktorý inicioval skladateľ (proces je jeho dielom – viď *Pendulum Music* a *In C* na začiatku tejto podkapitoly).
4. **Biologické/„vývojové“ hľadisko**
   Indeterministická hudba, ktorá je jedinečná a neopakovateľná, čo súvisí s hlavnou myšlienkou generatívneho umenia – autor je iniciátorom nekonečného počtu možných výsledných variantov vychádzajúcich z počiatočných podmienok.

Pre sonifikáciu je signifikantná najmä druhá a tretia perspektíva. Predom určeným systémom z druhej perspektívy môže byť sonifikačný algoritmus (ktorého vstup je spravidla nehudobný). Spustenie algoritmu je tiež možné vnímať ako iniciáciu procesu z tretej perspektívy. V rámci sonifikačného algoritmu je však možné využívať aj princípy súvisiace s prvou a štvrtou perspektívou.

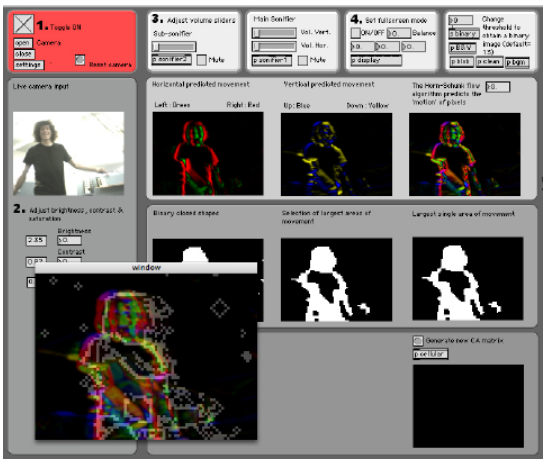
1.1.3 Sound art

Doteraz boli prezentované najmä možnosti sonifikácie v spojitosti s hudbou. Sonifikačné umelecké projekty však častokrát nadobúdajú nečakané hraničné formy spadajúce do kategórie sound artu: *umenia, ktorého primárnym vyjadrovacím médiom je zvuk*. V súlade s touto širokou definíciou je vzťah hudby a sound artu následovný: *nie každý sound art je hudba, ale každá hudba je sound art*.

Sound art sa však nesústredí na vytvorenie hudobného diela, ale často je predmetom jeho záujmu aj samotný nástroj, objekt alebo priestor, v ktorom zvuk vzniká. Jeho dôležitým rysom je časová neohraničenosť a nestatická forma (podobne ako u generatívnej hudby). Namiesto performatívnych situácií sound art patrí skôr do situácií exhibičných – typicky má podobu umeleckých inštalácií v galériách či iných priestoroch.

Sonifikácia je v sound arte často využívanou metódou, a to najmä v prípade interaktívnych senzorických inštalácií – ide o sonifikáciu v reálnom čase. Ak je sonifikačný algoritmus časovo invariantný systém, nedá sa hovoriť o interaktivite, ale iba o reaktivite. Interaktívna sonifikácia vzniká, ak sa vnútorná štruktúra systému mení na základe vstupu systému (používateľa / sonifikovaných dát). Dochádza k zmysluplnej komunikácii medzi subjektom a interaktívnym systémom. [1]

(a) J. Suchánek – *Soil Choir* [7]



(b) J. Jakovich – *Sonic Tai Chi* [8]

Obr. 1.4: Príklady sonifikácie v sound arte

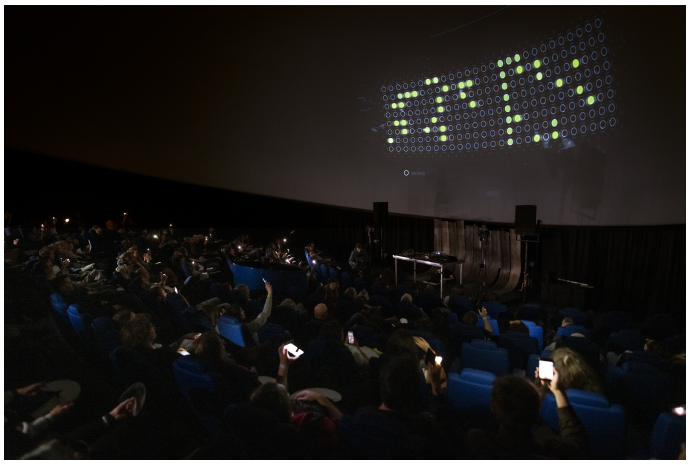
Príkladom interaktívnej sound-artovej inštalácie využívajúcej sonifikáciu môže byť *Soil Choir* [7] Jiřího Suchánka (obr. 1.4a) z roku 2019. Táto inštalácia sleduje veľmi pomalé zmeny v pôde snímaním jej vlhkosti a prevádzaním týchto dát na zvuk – časové merítko je tak oveľa väčšie ako pri bežnej hudobnej skladbe. Interakcia v tomto prípade prebieha medzi pôdou a sonifikačným systémom (oproti bežnejšiemu modelu návštevník–systém).

Inou ukážkou môže byť inštalácia *Sonic Tai Chi* [8] (obr. 1.4b) austrálskej umelkyne Joanne Jakovich z roku 2005, ktorá využíva techniky počítačového videnia (konkrétne optický tok) na sledovanie pohybov tela návštevníkov, ktoré sú transformované na zvuk využitím granulárnej syntézy a generatívneho modelu celulárnych automatov (spadajúc do štvrtej perspektívy generatívnej hudby – viď časť 1.1.2).

Ukážkou z vlastnej tvorby, ktorá vznikla počas písania tejto práce a bola prezentovaná na festivale SONDA 2022 [9], je projekt *Phone Orchestra* (F. Dobrocký, M. Stenko) na pomedzí inštalácie a performance, sonifikujúci svetlo mobilov divákov pomocou segmentácie obrazu, tvoriac zvuk v prostredí Max – obr. 1.5.

1.1.4 Vzťah zvuku a obrazu

Na vzťah zvuku a obrazu vzhľadom ku sonifikácii je možné prihliadať z viacerých perspektív. Pri sonifikácii statického obrazu sa ponúka príklad grafických partitúr, neštandardných spôsobov hudobnej notácie, ktoré využívajú grafické symboly a majú presah do výtvarného umenia – v niektorých prípadoch môže mať grafická zložka dokonca väčšiu váhu ako tá hudobná – každopádne ale ide o multimediálne dielo. Na základe tejto perspektívy sa sonifikáciou vizuálnych dát obraz stáva grafickou



Obr. 1.5: Phone Orchestra (foto: Karolina Raimund)

partitúrou. V prekrútenom chápaní sonifikácie je aj každá interpretácia štandardnej notácie sonifikáciou notografických symbolov[4].

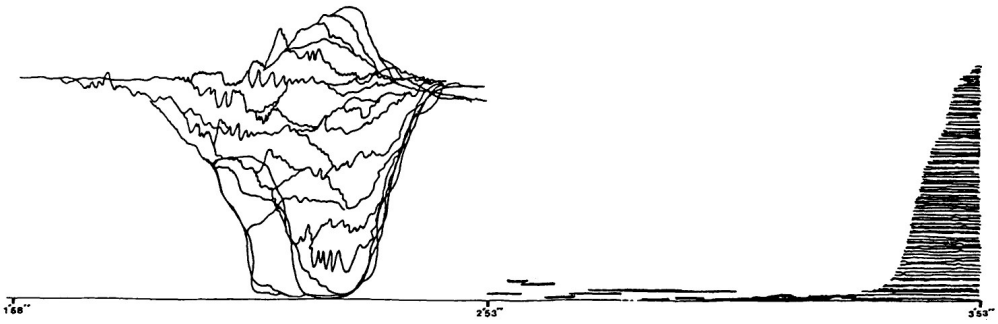
Vzhľadom k pohyblivému obrazu je prirodzenejšie sonifikáciu prirovnať k tvorbe zvukovej zložky (*sound designu*) audiovizuálneho diela. V kinematografii je nesmierne dôležitý súlad zvukovej a obrazovej zložky, či už ide o ruchové stopy, dialóg alebo hudbu. Človek vníma jeden jav viacerými zmyslami, resp. jeden jav môže vyvolať viacero súvisiacich vzruchov. Aj pri živých hudobných vystúpeniach je okrem zvukovej zložky dôležitá tá obrazová – vnímaný zvuk je dôsledkom pohybov interpretov, ktoré človek vníma vizuálne; konkrétny zvuk vzniká na konkrétnom nástroji, ktorý divák vidí. Zvuk je súčasťou „audiovizuálneho komplexu“ [11]. Na základe tejto prepojenosti obrazového a zvukového vnemu je možné usúdiť, že sonifikácia obrazových dát je najprirodzenejšou formou sonifikácie.

Zložky zvukovej stopy filmu (ruchy či hudba) sú najčastejšie dotvárané na základe už hotového obrazu, teda ide o prístup blízky sonifikácii. Doplnenie obrazu je však v rukách sound designera či hudobného skladateľa a (väčšinou) nejde o systematickú interpretáciu obrazu pomocou transformačného algoritmu.

Algoritmická tvorba zvukovej zložky audiovizuálneho diela (tzv. *adaptívny sound design*) býva aplikovaná v počítačových hrách. Sound designer netvorí statickú zvukovú stopu, ale navrhuje princípy, na základe ktorých je zvuk tvorený. Zvuk však nevzniká na základe obrazu; obraz aj zvuk vznikajú asynchrónne a súvisiace zvukové a obrazové deje sú riadené tými istými udalosťami (prislúchajú tým istým objektom). Každopádne je možné podobné postupy uplatniť pri sonifikácii videa, kde sú dáta (objekty), na základe ktorých zvuk vzniká, získavané z obrazu (viac v časti 1.3.1).

Vzťah medzi obrazom a zvukom pri sonifikácii však nemusí byť vždy taký priamy.

[4]Nemá však význam to sonifikáciou nazývať.



Obr. 1.6: Príklad grafickej partitúry: *Iannis Xenakis – Mycenae-Alpha* [10]

Pomocou techník *počítačového videnia* je možné z obrazu získať rôznorodé informácie. Je možné zamerať sa na konkrétny fenomén získaný z obrazu a tento fenomén systematicky interpretovať pomocou zvuku (viď definíciu sonifikácie – časť 1.1).

Medzi možnosti využitia umeleckej sonifikácie videa teda patrí:

- algoritmická kompozícia,
- generatívna hudba riadená obrazom,
- algoritmický sound design audiovizuálneho diela,
- interaktívne zvukové inštalácie,
- iné formy sound artu.

1.2 Počítačové videnie

Cieľom počítačového videnia je využívanie kamery pre analýzu a porozumenie reálnym scénam. Táto disciplína sa zameriava na metodologické a algoritmické problémy, ale aj problematiku spojenú s implementáciou konkrétnych riešení. [12]

Človek vníma trojdimenzionálnu štruktúru sveta okolo neho so zdanlivou ľahkosťou. Vie bez námahy identifikovať tvar objektu a odlíšiť ho od pozadia, určiť počet ľudí v skupinovom portréte alebo dokonca odhadnúť ich emócie. Docieliť podobného chápania scény pomocou počítača nie je ani z ďaleka triviálnym problémom. Je to sčasti preto, že ide o *inverzný problém*, v ktorom je cieľom získanie neznámych z neúplných informácií pre úplné určenie riešenia. Preto je potrebné uchýliť sa k fyzikálnym alebo pravdepodobnostným modelom, prípadne strojovému učeniu. [13]

Vývoj počítačového videnia bol však v posledných dvoch dekádach veľmi rýchly a dnes sa stretneme s množstvom reálnych aplikácii v oblastiach ako napríklad [13]:

- **optické rozpoznávanie textu** – napr. čítanie ručne písaných smerovacích čísel, rozpoznávanie poznávacích značiek áut,

- **strojová inšpekcia** – automatická kontrola kvality výrobkov,
- **maloobchod** – využitie rozpoznávania objektov pre automatické pokladne aj plne automatizované obchody,
- **logistika** – autonómne doručovanie zásielok,
- **motion capture** – záznam pohybov hercov pre účely počítačovej animácie,
- **match move** – automatické spájanie počítačom generovaného obrazu a živého záznamu,
- **sledovanie** – bezpečnostné kamery, monitorovanie premávky,
- **biometrika** – rozpoznávanie odtlačkov prstov, tváre atď.,
- **zobrazovacie metódy v lekárstve**,
- **autonómne vozidlá**.

Pre uvedené aplikácie sú využívané rozličné techniky spadajúce do oblastí ako

- rozpoznávanie,
- analýza pohybu,
- analýza hĺbky,
- rekonštrukcia scény.

Pre túto prácu je dôležitá predovšetkým oblasť *rozpoznávania*, kde spadá detekcia a klasifikácia objektov, a oblasť *analýzy pohybu*, ktorej súčasťou je sledovanie objektov (*object tracking*).

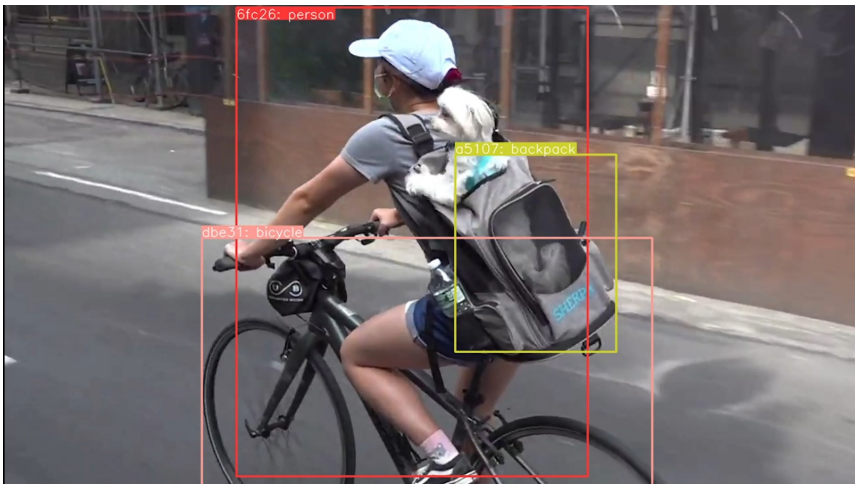
1.2.1 Rozpoznávanie objektov

Rozpoznávanie objektov ako také je možné rozdeliť do dvoch kategórií: *rozpoznávanie inštancie* a *rozpoznávanie tried*. Prvá kategória sa zaoberá hľadaním konkrétneho známeho objektu v reálnej scéne, zatiaľ čo druhá kategória, tiež známa ako rozpoznávanie *na úrovni kategórie* alebo *generické* rozpoznávanie[5], je omnoho zložitejším problémom, ktorého cieľ je rozpoznanie akejkoľvek inštancie danej triedy (napr. osoba, mačka, bicykel...).

Tradičnejšie metódy rozpoznávania sú založené na detekcii tzv. príznakov (*features*) a sú stále často preferovanými metódami rozpoznávania inštancie. Pre rozpoznávanie tried sa však ukázali efektívnejšími techniky založené na *hlbokom učení* (*deep learning*), ktoré v posledných rokoch značne posunuli oblasť počítačového videnia.

Spočiatku bol výskum generického rozpoznávania venovaný hlavne detekcii tváre (*face detection*) a detekcii chodcov (*pederstrian detection*), či celkovej klasifikácii scény, avšak počítačové videnie malo odjakživa úmysel vyriešiť všeobecný problém detekcie a označovania objektov v obraze – tzn. určiť, v akej oblasti sa v obraze nachádzajú objekty rozličných tried. Toto dnes umožňujú moderné detektory objektov založené na konvolučných neurónových sieťach.

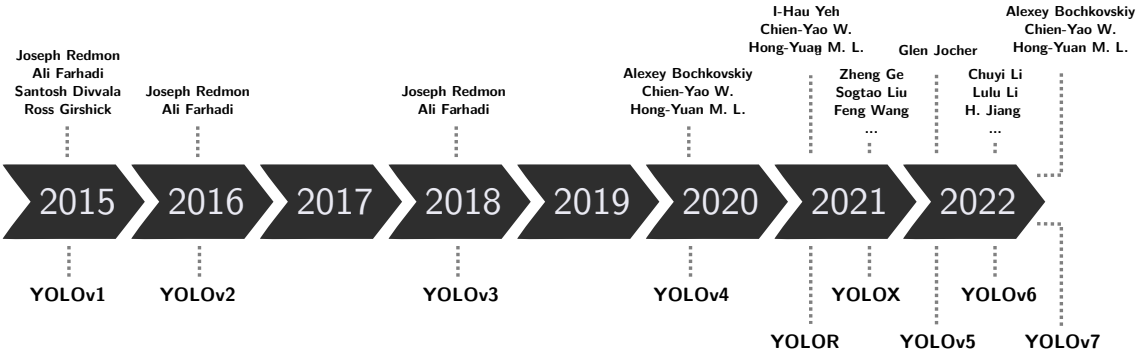
[5]Category-level / generic object recognition. [13]



Obr. 1.7: Zobrazenie výstupu detektoru YOLOv5

Jeden z prvých takýchto detektorov je R-CNN[6] z roku 2014. Ide o dvojštádiový detektor – v prvom štádiu sú hľadané možné oblasti objektov v obraze pomocou algoritmu selektívneho vyhľadávania, druhé štádium spočíva v klasifikovaní týchto oblastí (určení triedy objektu) prostredníctvom konvolučnej neurónovej siete. Okolo roku 2016 sa začali objavovať jednoštádiové detektory, používajúce jedinú neurónovú sieť pre detekciu a klasifikáciu objektov v rozličných oblastiach obrazu. Jednoštádiové detektory umožňujú rýchlejšiu detekciu a klasifikáciu a sú teda vhodné pre aplikácie pracujúce v reálnom čase, zatiaľ čo prednosťou viacštádiových detektorov môže byť väčšia presnosť. [13]

YOLO – *You Only Look Once*



Obr. 1.8: Modely YOLO, ich autori a roky vzniku [14]

[6]Region-based Convolutional Network

V súčasnosti sú pre detekciu objektov v reálnom čase najvhodnejšie a najčastejšie používané jednoštádiové detektory z rodiny *YOLO*. Pôvodný algoritmus bol predstavený v roku 2015 vo vedeckom článku, na ktorom sa podieľali Joseph Redmon a Ali Farhadi, tiež autori ďalších dvoch vylepšených modelov YOLOv2/9000 (2016) a YOLOv3 (2018). Odvtedy vzniklo množstvo príbuzných modelov nesúcich meno YOLO od rôznych autorov (obrázok 1.8), so snahou zlepšenia rýchlosti a presnosti detekcie.

Hlavnými prednosťami modelov YOLO je vysoká rýchlosť, veľká presnosť detekcie, široká škála využitia a otvorený zdrojový kód.

Hrubý princíp algoritmu je možné rozdeliť do štyroch blokov:

1. **Residual blocks**
   Obraz je rozdelený do rovnomernej $N \times N$ mriežky. Každá bunka má na starosti lokalizáciu a odhad triedy objektu, ktorý sa v nej nachádza a tiež určenie významnosti predikcie (*confidence value*).
2. **Bounding box regression**
   Ďalším krokom je určenie ohraničujúcich rámčekov (*bounding boxes*) objektov pomocou modelu regresie. Ohraničujúci rámček je reprezentovaný vektorom $\mathbf{y} = (p_c, b_x, b_y, b_w, b_h)$, kde p_c značí pravdepodobnosť, že sa v rámčeku nachádza objekt, b_x, b_y sú súradnice stredu rámčeku relatívne k bunke mriežky a b_w, b_h je šírka a výška rámčeku relatívna k celej mriežke.
3. **Intersection Over Unions (IOU)**
   Jeden objekt môže mať niekoľko ohraničujúcich rámčekov, pričom nie všetky z nich musia byť relevantné. IOU značí pomer plochy prieniku prekrývajúcich sa rámčekov ku ploche ich zjednotenia. Stanovením IOU *thresholdu* sú odstránené tie duplicitné rámčeky toho istého objektu, ktorých $IOU > threshold$.
4. **Non Max Suppression (NMS)**
   Definovanie IOU thresholdu nemusí stačiť pre odstránenie všetkých duplicitných rámčekov, preto je ešte aplikované tzv. potlačenie nemaximálnych hodnôt (NMS), vďaka ktorému je pre daný objekt uchovaný len ohraničujúci rámček s najväčšou hodnotou pravdepodobnosti p_c. [14]

Každá bunka $N \times N$ mriežky predpovedá B ohraničujúcich rámčekov[7] a C hodnôt pravdepodobnosti jednotlivých tried objektu. Predikcie sú zakódované ako $N \times N \times (5B + C)$ *tenzor*. [15]

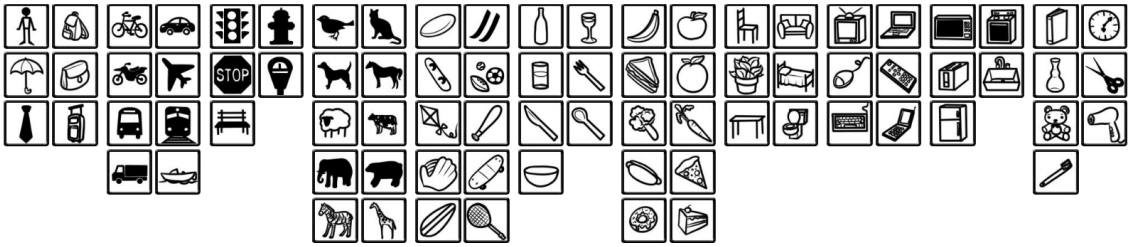
V dostupných implementáciách ([16], [17]) je spracovaným výstupom algoritmu zoznam detekcií, pričom detekcia môže mať formát `(xyxy, conf, cls)`, kde `xyxy` sú súradnice rohových bodov rámčeku, `conf` je významnosť predikcie a `cls` je identifikátor triedy objektu. Výstup v takomto formáte možno jednoducho zobraziť alebo

[7] Ohraničujúci rámček reprezentuje vektor $\mathbf{y}$ dĺžky 5.

ďalej spracovávať.

COCO Dataset

Pre porovnávanie účinnosti algoritmov YOLO je najčastejšie využívaný dataset COCO – *Common Objects in Context*. Dostupné implementácie YOLO ponúkajú modely trénované týmto datasetom, ktoré sú schopné rozpoznať 80 bežných objektov v reálnych scénach. Pre rôzne špecifické aplikácie je možné trénovať vlastné modely využitím vlastných špecifických datasetov. Východzie modely však umožňujú jednoduché a univerzálne využitie detektorov.



Obr. 1.9: 80 tried objektov datasetu COCO [18]

Dataset COCO obsahuje až 330 tisíc obrázkov (z čoho je viac ako 200 tisíc anotovaných), na ktorých je až 1,5 milióna inštancií objektov v rôznom kontexte. Pre anotácie je využitých 80 kategórií objektov (*COCO classes*) a 91 kategórií kontextu (*COCO stuff* – materiály alebo objekty bez konkrétneho ohraničenia, napr. obloha, ulica, tráva...). Dataset je okrem modelov pre detekciu objektov v reálnom čase možné využiť aj pre celkovú klasifikáciu obrazu, sémantickú segmentáciu[8], prípadne aj pre odhad držania tela (*pose estimation*). [18]

1.2.2 Sledovanie objektov

Pri real-time detekcii objektov vo videu sú v každej snímke hľadané inštancie objektov, reprezentované ako ohraničujúce rámčeky. Z pohľadu týchto algoritmov sú inštancie v rozdielnych snímkach videa na sebe nezávislé aj vtedy, keď ide o ten istý objekt. Zavedením sledovania objektov je možné asociovať detekcie naprieč snímkami a získať tak informácie o pohybe jednotlivých objektov vo videu. Táto problematika sa označuje ako *Multiple Object Tracking* – MOT.

[8]Odlíšenie tried alebo inštancií na úrovni pixelov – ohraničenie objektov je presnejšie ako jednoduchý rámček.

SORT – Simple Realtime Online Tracking

MOT algoritmy využívajú rozličné metódy modelovania pohybu a vzhľadu objektov v scéne. Algoritmy sú však často výpočtovo náročné a málokedy použiteľné pre real-time aplikácie. Jeden z algoritmov, ktorý cieli na čo najmenšiu časovú zložitosť a real-time aplikovateľnosť je SORT – *Simple Realtime Online Tracking*. Tento algoritmus nevyužíva vizuálne dáta a vstupom sú len detekcie reprezentované ako ohraničujúce rámčeky. Algoritmus priraďuje detekcie z novej snímky k existujúcim objektom pomocou predikcie geometrie ohraničujúcich rámčekov existujúcich objektov v novej snímke.

Pre predikciu pohybu je využitý *Kalmanov filter* a priraďovanie detekcií je vyriešené optimálne pomocou *maďarskej metódy*[9]. Matica cien priradenia je počítaná ako IOU každej detekcie a všetkých predikovaných rámčekov existujúcich objektov. Je stanovené minimálne IOU pre vyradenie priradení nepresahujúcich IOU_{min}.

Keď objekt vojde do obrazu, resp. z neho odíde, musí byť vytvorená nová identita, resp. zrušená stará. Detekcia s prekrytím menším ako IOU_{min} značí existenciu nesledovaného objektu. Nová identita je potvrdená až pri niekoľkých priradeniach pre potlačenie falošných detekcií. Identita je zrušená, ak jej nie je priradená žiadna detekcia po T_{Lost} snímok. [19]

Deep SORT

Deep SORT vylepšuje pôvodný algoritmus SORT integrovaním informácií o vzhľade pomocou *hlbokej asociačnej metriky*. Vďaka tomu je možné sledovať objekty aj pri dlhšej dobe prekrytia a v konečnom dôsledku je tak zmenšená miera zámeny identity. Asociačná metrika pôvodného algoritmu SORT založená na IOU nie je spoľahlivá pri vyššej miere neistoty predikcie pohybu, a teda má nedostatok pri oklúziách. Preto bola v asociačnej metrike algoritmu Deep SORT zavedená konvolučná neurónová sieť trénovaná pre rozlíšenie chodcov. Algoritmus značne zväčšuje svoju presnosť, no kvôli zložitejšej asociačnej metrike je znížená rýchlosť. Real-time aplikovateľnosť však zostáva zachovaná. [20]

1.3 Tvorba zvuku

Uvedené techniky počítačového videnia sú založené na objektovom chápaní obrazu. Ponúka sa preto v rámci sonifikácie obrazu pomocou týchto techník nazerať objektovo aj na zvuk. Zvuková zložka teda môže byť algoritmicky komponovaná pomocou *zvukových objektov*, ktoré sú systematicky prepojené s vizuálnymi objektmi.

[9] Hungarian algorithm

1.3.1 Zvukový objekt

Zvukový objekt je termín zavedený Pierrom Schaefferom v roku 1966 v spojitosti s *akuzmatickou* hudbou. Akuzmatické počúvanie je opakom priameho počúvania, teda situácie, v ktorej sú zdroje zvuku prítomné a viditeľné. Akuzmatické situácie sú už dnes veľmi bežné – patrí sem reprodukovaná hudba, akékoľvek zvukové nahrávky, telefón, rádio atď. Takéto situácie menia prirodzenú percepciu zvuku – zvuk je separovaný od audiovizuálneho komplexu a stáva sa autonómnym – zvuk oddelený od svojho zdroja je vnímaný ako *zvukový objekt*. Materiálny zdroj, z ktorého zvuk pochádza, Schaeefer nazýva *zvukovým telom* – oproti nemu je zvukový objekt niečo nemateriálne a založené na percepcii. Jedno zvukové telo môže byť zdrojom rôznorodých zvukových objektov.

Táto koncepcia umožňuje organizáciu zvukov komplexného charakteru. Podobne ako nota v klasickom chápaní hudby je zvukový objekt základnou zvukovou jednotkou. Nejde ale o ekvivalent – napr. glissando na harfe je v hudobnom zápise zložené z niekoľkých nôt, no pre poslucháča je jedným zvukovým objektom. Zvukový objekt sa môže skladať z viacerých mikroudalostí vzájomne zviazaných formou. [11] Zvukový objekt v ponímaní Curtisa Roadsa generalizuje koncept noty a rozširuje ho o komplexné a obmieňajúce sa zvuky na časovej škále od zlomku sekundy po niekoľko sekúnd. [21]

Zvukové objekty teda dovoľujú rôzne prístupy k organizácii zvuku do väčších štruktúr [22]:

- **inštrumentálny prístup** využívajúci štandardnú notáciu (krátke zvuky danej výšky a farby reprezentovateľné ako noty),
- **musique concrète**[10] používajúca širokú škálu zvukových objektov rôzneho charakteru kategorizovaných pomocou fenomenologického[11] popisu ich vlastností,
- **zvuková syntéza** umožňujúca dlhotrvajúce a obmieňajúce sa „prúdy zvuku“.

Zvukový objekt v priestore – zvukové krajiny

Anglický skladateľ Trevor Wishart zdôrazňuje aj priestorové hľadisko zvukových objektov tvoriacich „zvukové krajiny“. Pri ustanovení koherentného sluchového obrazu reálneho akustického priestoru je možné určiť pozíciu zvukových objektov v tomto priestore – šírka je reprezentovaná rozložením v stereu, hĺbku predstavuje znižujúca sa intenzita zvuku, utlmené vysokofrekvenčné zložky a zväčšujúca sa reverberácia.

[10] Tiež *konkrétna hudba* – hudobný smer založený Pierrom Schaefferom, používajúci nahrané zvuky ako primárny materiál.

[11] Prístup spojený s filozofiou fenomenológie; popis zameraný na skúsenosť a povedomie pozorovateľa namiesto samotného predmetu pozorovania.

Objekty môžu byť statické alebo sa pohybovať v priestore – pohyb objektu môže v zvukovej krajine predstavovať Dopplerov jav. Wishart popisuje tri typy *imaginárnych zvukových krajín* [22]:

- **nereálne objekty / reálna krajina** – rozloženie objektov zodpovedá realistickému akustickému priestoru, ale zdroje zvuku reálne nie sú;
- **reálne objekty / nereálna krajina** – reálne zvukové objekty sú nereálne rozložené v priestore mixážou;
- **reálne objekty / reálna krajina (surrealizmus)** – reálne sú zvukové objekty aj ich rozloženie, ale vzťah jednotlivých objektov reálny nie je.

Tento priestorový prístup je blízky oblasti sound designu audiovizuálneho diela, kde sú zvukové objekty priradené viditeľným zdrojom. Zvuková krajina je tvorená tak, aby zodpovedala vizuálnej krajine – prípadne aby, so zámerom vytvorenia audiovizuálnej disonancie, bola s obrazom vo výraznom kontraste. Každopádne sú pri produkcii zvukovej stopy audiovizuálneho diela vytvárané imaginárne zvukové krajiny.

Technologické ponímanie zvukového objektu

Objektový prístup je uplatňovaný predovšetkým vo zvuku počítačových hier, v priestorových zvukových systémoch (napr. Dolby Atmos), distribučných štandardoch, ako MPEG-H, alebo v *binaurálnej virtualizácii* zvuku, využívanej o. i. v systémoch virtuálnej reality. Vtedy hovoríme o *objektovo založenom audiu*, ktoré je alternatívou zvukových formátov založených na fixnom počte kanálov. Zvukový objekt je v tomto prípade monofónny signál, ktorému prislúchajú 3D súradnice relatívne k poslucháčovi. Zvukové objekty tak môžu byť behom produkcie precízne rozmiestnené nezávisle na cieľovom zvukovom systéme a výsledné kanály sú pri prehrávaní tvorené v reálnom čase procesom adaptívneho renderovania. V priestorových zvukových systémoch tak môžu byť objekty renderované do usporiadaní s veľkým počtom kanálov. Pri binaurálnej virtualizácii je v procese renderovania na objekty aplikovaný HRTF filter, ktorého frekvenčná charakteristika záleží na polohe objektu. [23]

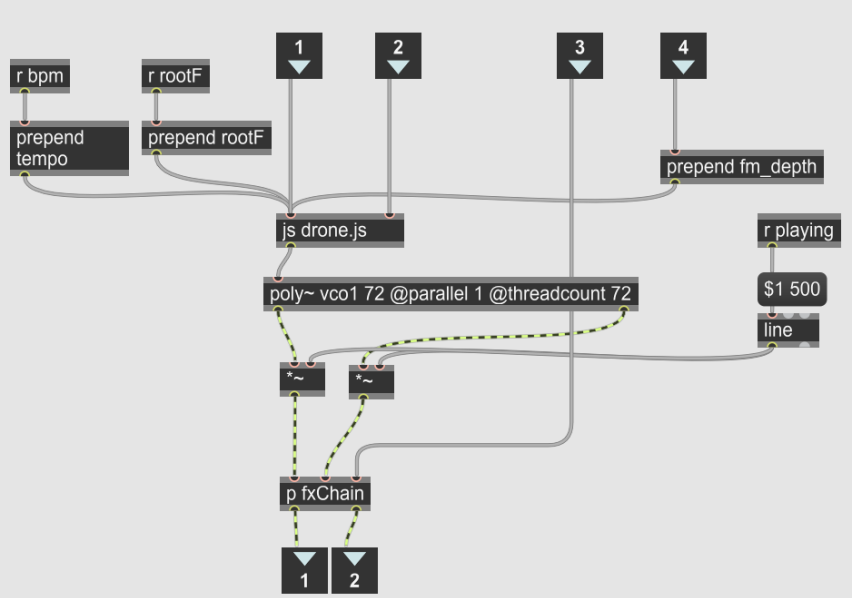
Okrem spomenutých sofistikovaných spôsobov 3D reprodukcie a virtualizácie zvuku je možné objektový vzor (aj technologicky menej náročným spôsobom) aplikovať v algoritmickej tvorbe zvukových krajín v reálnom čase, zohľadnením parametrov určujúcich polohu zvukového objektu popisovaných Trevorom Wishartom.

1.3.2 Programovacie prostredia

Pre algoritmickú tvorbu zvuku je výhodné využiť niektoré z dostupných programovacích prostredí umožňujúcich zvukovú syntézu v reálnom čase. Tieto prostredia typicky ponúkajú vysokoúrovňový (často aj grafický) programovací jazyk, ktorým

je možné ovládať istý zvukový engine a abstrahovane tak pracovať s algoritmami digitálneho spracovania signálov. Pomocou protokolov ako MIDI či OSC umožňujú jednoduché prepojenie s externými dátami (typicky z rôznych kontrolérov alebo senzorov) a teda aj interaktivitu.

Max



Obr. 1.10: Ukážka časti patcheru v prostredí Max 8

Softvér *Max* je prostredie vizuálneho programovania pre zvuk a multimédiá pôvodne vytvorený Millerom Pucketteom v 80. rokoch. V súčasnosti je to jeden s najpoužívanejších nástrojov pre tvorbu interaktívnej hudby a audiovizuálneho umenia. Puckette má na svedomí aj softvér *Pure Data (Pd)*[12], ktorý je principiálne veľmi podobnou open-source alternatívou proprietárneho Maxu teraz spravovaného spoločnosťou Cycling '74.

Softvér, niekedy označovaný aj ako *Max/MSP/Jitter*, umožňuje pracovať s audio signálmi (časť označovaná ako *MSP*) aj s obrazom (*Jitter*) prostredníctvom prepájania modulov (*objektov*) a názorným spôsobom tak riadiť tok signálov. Jedná sa teda o tzv. *flow-based* jazyk, ktorý definuje aplikácie (v Maxe označované ako *patchers*) ako siete black-box procesov, ktoré si vzájomne predávajú správy (*messages*) pomocou externe definovaných konexií (*patch cords*). Objekt môže mať niekoľko vstupov

[12]Prostredie Pure Data pre jeho podobnosť s prostredím Max v práci nie je podrobnejšie popísané, aj keď v niektorých aplikáciách môže byť vhodnejšie (napr. pre menšiu výpočtovú náročnosť alebo otvorený zdrojový kód).

(*inlets*) a výstupov (*outlets*). Max rozlišuje medzi objektmi pracujúcimi s *udalosťami* a objektmi, ktoré pracujú so signálmi – tie označuje znakom `~` (napr. `cycle~` je harmonický oscilátor). Nadobudnutím základnej znalosti funkcionality objektov Maxu je tak možné zostavovať patchery, ktoré komplexne riadia dáta rôzneho typu spolu s generovaním či spracovaním zvuku alebo obrazu.

Max umožňuje zapúzdrenie vnútorných častí patcheru do tzv. *subpatch*ov, prípadne patcher uložiť ako abstrakciu a použiť ho v inom patcheri. Priamo v prostredí je tiež možné programovať vlastné objekty v jazyku JavaScript. Max ponúka aj API, pomocou ktorého je možné vyvíjať externé objekty v jazyku C. Max v jeho súčasnej podobe je mimoriadne robustným nástrojom s množstvom rozšírení. Okrem iného umožňuje aj integráciu do DAW Ableton Live. [24] [25]

SuperCollider

Prostredie SuperCollider, pôvodne vydané v roku 1996, je oproti Maxu zamerané výhradne na zvuk. Zásadným rozdielom je využitie objektovo orientovaného skriptovacieho jazyka namiesto vizuálneho programovania. Medzi časté využitia patrí živé programovanie hudby (*live coding*), tvorba generatívnej hudby, realizácia interaktívnych zvukových inštalácií alebo tiež akustický výskum. Prostredie používa klient–server architektúru a skladá sa z troch častí:

`sclang` — interpretovaný programovací jazyk,

`scserver` — real-time audio server,

`scide` — editor / vývojové prostredie pre `sclang`.

Komunikácia medzi klientom a serverom je realizovaná pomocou protokolu OSC, čo umožňuje použitie aj iných klientov ako `sclang`, a teda je možné ovládať `scserver` z rôznych aplikácií a jazykov, pričom jeden server môže mať viacero klientov. Je tak možné okrem iného prepojiť generovaný zvuk s vizuálom alebo senzorickým vstupom. [26]

`sclang` je objektovo orientovaný funkcionálny jazyk. Funkcia je definovaná ako kód obklopený zloženými zátvorkami `{ }` a je tiež objektom. Jazyk umožňuje definovať komplexné správanie minimom kódu. Napr. následujúci kód prehráva sínusový signál amplitúdovo modulovaný nízkofrekvenčným oscilátorom:

```
{ var amp;
    amp = SinOsc.kr(0.5, 1.5pi, 0.5, 0.5);
    SinOsc.ar(440, 0, amp);
}.play;
```

Syntax `.play` znamená, že je objektu (ktorý je v tomto prípade funkcia) poslaná *správa* `play`, ktorá spustí prehrávanie signálov definovaných v tomto objekte. SuperCollider rozlišuje medzi dvoma frekvenciami kalkulácie signálov: *audio rate* pre signály v počuteľnom frekvenčnom pásme (správa `ar`) a *control rate* pre nízkofrekvenčné, riadiace signály (správa `kr`). Vzorky audio signálov sú spracovávané v segmentoch („blokoch“). Pre jeden blok je počítaná len jedna vzorka control rate signálu, zatiaľ čo audio rate signál má jednu hodnotu pre každú vzorkovaciu periódu. Objekty pre generovanie signálov, schopné odpovedať na správy `ar` a `kr` sú potomkami abstraktnej triedy `UGen` (*unit generator*) – to je aj generátor sínusového signálu `SinOsc`. Správy `ar` a `kr` sú volaniami metód potomkov `UGen`, ktoré môžu byť volané bez ohľadu na to, o aký `UGen` ide – jazyk teda dovoľuje *polymorfizmus*.

SuperCollider umožňuje definovať serveru vlastné syntetizátory v `sclang` syntaxou `SynthDef`. Potom je možné v programe ľubovoľne vytvárať inštancie daného syntetizátoru:

```
SynthDef(\synth, { |out, freq = 800, amp = 0.1|
    Out.ar(out, SinOsc.ar(freq, 0, amp))
}).add;
x = Synth.new(\synth, {[\freq, 300]});
y = Synth.new(\synth, {[\freq, 800]});
z = Synth.new(\synth, {[\freq, 850]});
```

`Synth` v ponímaní SuperCollideru nemusí byť vyslovene syntetizátor, ale môže napr. plniť funkciu efektového procesoru. `Synth` je tzv. serverovou abstrakciou. Ďalšími dôležitými abstrakciami sú `Bus` (zbernica), `Group`, umožňujúca skupinové ovládanie syntetizátorov, alebo `Buffer` pre uchovávanie audio nahrávok či syntetizovaných signálov. [27]

Objektový charakter SuperCollideru je výhodou, ak je cieľom algoritmická tvorba zvukových objektov. Typ zvukového objektu môže byť definovaný ako `SynthDef` SuperCollideru a v programe potom môžu byť systematicky tvorené inštancie týchto objektov (v našom prípade na základe detekovaných objektov v obraze). Preto bolo toto prostredie zvolené za vhodný prostriedok pre realizáciu praktickej časti práce.

1.3.3 Základné stavebné prvky zvukovej syntézy

Predchádzajúca časť sa ľahko dotkla digitálnej syntézy zvuku. Tá je hlavným účelom zahrnutia programovacích prostredí do tejto práce a preto je vhodné zhrnúť jej základné princípy.

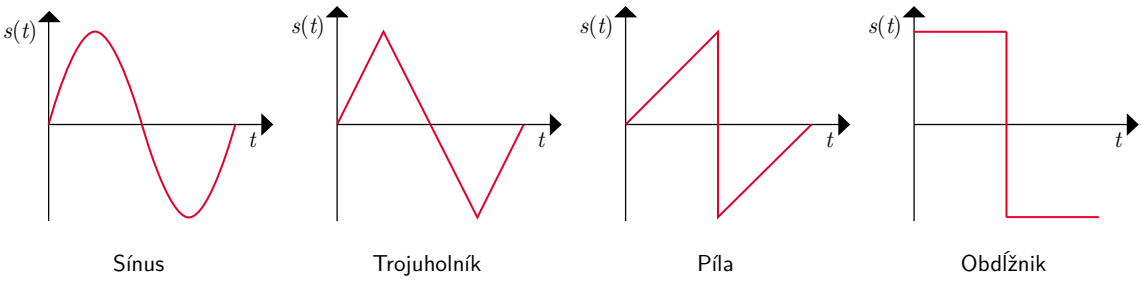
Stavebné prvky digitálnej syntézy sa v kontexte zvukových programovacích jazykov označujú ako *unit generators*[13] (v SuperCollideri *UGens*). Sú ekvivalentom

[13]Termín zaviedol Max Mathews, autor jazyka MUSIC vyvinutého v roku 1957 v Bell Labs.

modulov (obvodov) analógového syntetizátoru a reprezentujú algoritmy zvukovej syntézy a spracovania signálov. Prepájaním týchto blokov je možné realizovať konkrétne nástroje zvukovej syntézy – *patche* generujúce zvukové signály. Unit generator môže byť buď generátorom (oscilátor / LFO, šumový generátor, sampler, generátor obálky) alebo modifikátorom signálu (filter, efektový procesor, mixer). [28]

Oscilátor

Oscilátor generuje periodický priebeh zvukového signálu – najčastejšie sínusový, obdĺžnikový, trojuholníkový alebo pílový. V analógovej syntéze sa označuje ako VCO (*Voltage Controlled Oscillator*), čo značí, že je možné ovládať jeho základnú frekvenciu (a prípadne iné parametre) napätím – ekvivalentom v hardvérových digitálnych syntetizátoroch je DCO.



Obr. 1.11: Bežné tvary priebehov generovaných oscilátorom

V digitálnej syntéze je možné signál generovať rôznorodými algoritmami, čo môže byť spojené aj s *typom* syntézy (časť 1.3.4). Napríklad pomocou *wavetable* syntézy môže byť okrem spomenutých bežných priebehov generovaný aj zložitejší signál uložený vo vyhľadávacej tabuľke.

Priebeh generovaný oscilátorom tvorí základ zvukovej farby. Jeho frekvenčné spektrum môže byť ďalej upravované napr. filtráciou (*subtraktívna syntéza*), saturáciou (*waveshaping* syntéza) alebo moduláciou amplitúdy či frekvencie (*modulačné* syntézy).

Príklady *UGen*ov SuperCollideru: `SinOsc`, `LFTri`, `LFSaw`, `VarSaw`, `LFPulse`. [27]

Šumový generátor

Tiež ide o generátor signálu, nie však periodického, ale náhodného. V kombinácii s oscilátormi môže slúžiť pre pridanie šumových zložiek do harmonického spektra.

Tento jazyk je nepriamym predchodcom prostredí SuperCollider a Max (ktoré bolo po Mathewsovi aj pomenované).

Príklady *UGen*ov SuperCollideru: `WhiteNoise, PinkNoise, BrownNoise`. [27]

Sampler

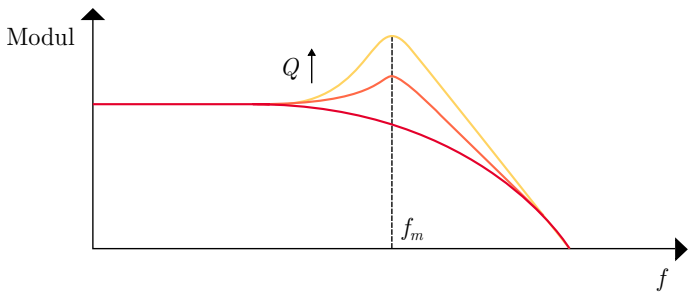
Sampler prehráva (spravidla kratšie) prednahraté zvuky. Pomenovanie vychádza z anglického slova pre vzorkovanie – *sampling*. Pojmom *sample* sa však v tomto prípade nemyslí jedna vzorka digitálneho signálu, ale krátka nahrávka, ktorú sampler prehráva. Samplery väčšinou umožňujú ovládanie parametrov ako je rýchlosť a smer prehrávania alebo opakované prehrávanie (*loop*).

UGen SuperCollideru s funkcionalitou sampleru je `PlayBuf`, ktorý pracuje so serverovou abstrakciou `Buffer`. Príbuzným *UGen*om je aj `GrainBuf`, ktorý realizuje granulárnu syntézu. [27]

Filter

Filter tvaruje frekvenčné spektrum zvukového signálu tak, že zoslabuje určité časti spektra, zatiaľ čo iné časti prepúšťa. Najbežnejšie typy filtrov sú dolnopriepustný filter (*Low Pass Filter* – LPF), hornopriepustný filter (High Pass Filter – HPF), pásmovopriepustný filter (*Band Pass Filter* – BPF) alebo filter typu pásmová zádrž (*Band Stop Filter* – BSF).

Rád filtru (ktorý určuje počet jeho *pólov*) ovplyvňuje strmosť prechodu do nepriepustného pásma – filter I. rádu má strmosť 6 dB/okt, filter II. rádu 12 dB/okt... [29]



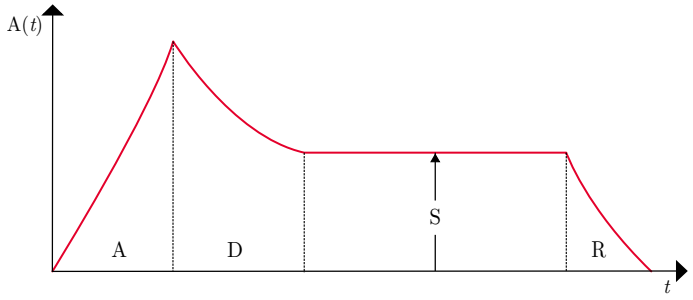
Obr. 1.12: Náčrt modulovej frekvenčnej charakteristiky rezonančného LPF

Vo zvukovej syntéze sa najčastejšie používa dolnopriepustný filter II. rádu. Bežné parametre filtru syntetizátoru sú *cutoff* (medzná frekvencia filtru f_m) a *resonance* (parameter ovláda kvalitu filtru Q). Rezonancia znamená zosilnenie pásma okolo medznej frekvencie filtru a vzniká vďaka spätnej väzbe v štruktúre filtru. Pri veľmi vysokom nastavení rezonancie dochádza k *samooscilácii* filtru – filter sa stáva oscilátorom. [28]

SuperCollider má mnoho *UGen*ov pre rôzne typy filtrov, ktoré sú potomkami abstraktnej triedy `Filter`. Príkladmi pre bežné filtre II. rádu sú: `BLowPass, BHiPass, BBandPass, BBandStop`. [27]

Generátor obálky (Envelope generator)

Tento modul generuje riadiaci signál typu obálka. Obálka, ako aj iné riadiace signály, môže ovládať (modulovať) rôzne parametre iných modulov syntetizátoru. Najčastejšie sa používa na riadenie *dynamickej obálky* zvukového signálu – jeho dynamického priebehu v čase. V syntetizátoroch často ovláda aj *cutoff* filtru.



Obr. 1.13: Obálka ADSR

Najbežnejším typom obálky je ADSR (*Attack, Decay, Sustain, Release*). *Attack* je čas, za ktorý signál dosiahne maximálnu hodnotu, *decay* je čas, za ktorý sa signál dostane na ustálenú úroveň, ktorú značí parameter *sustain*, a *release* je čas poklesu z tejto úrovne na nulu.

Istým nastavením parametrov ADSR je možné dostať aj iné zjednodušené typy obálky – nastavením úrovne *sustain* na nulu vznikne AD obálka používaná pre perkusné zvuky, prípadne nastavením parametru *decay* na nulu a *sustain* na maximálnu hodnotu vznikne obálka AR bez fázy ustálenia sa na danú úroveň.

UGen SuperCollideru s funkciou generátoru obálky je `EnvGen`. Generuje obálku definovanú pomocou objektu `Env`, ktorý dovoľuje aj viac kriviek ako bežná obálka ADSR, pričom je možné definovať aj ich individuálnu krivosť. [27]

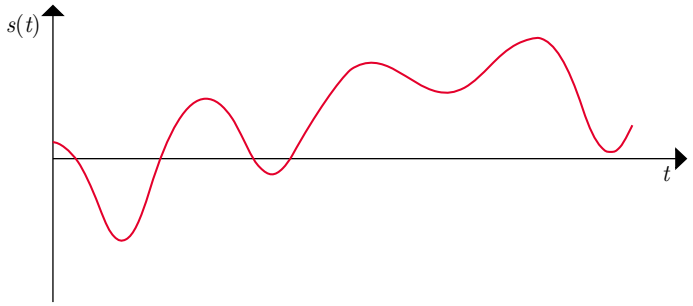
Nízkofrekvenčný oscilátor (LFO)

Nízkofrekvenčný oscilátor rovnako ako obálka generuje riadiaci signál, a teda slúži pre moduláciu rôznych zvukových parametrov. Principiálne je podobný oscilátoru, avšak generuje periodické signály o frekvenciách pod počuteľným pásmom (pod 20 Hz).

Niektoré LFO však dovoľujú nastavenie frekvencie aj do počuteľného pásma a umožňujú tak rozmazávať hranicu medzi rytmom a výškou tónu – napr. LFO

s frekvenciou 1 Hz modulujúce amplitúdu signálu vytvára pomalý rytmický pattern, zvyšovaním frekvencie sa pattern zrýchľuje, a keď presiahne do počuteľného pásma, mení frekvenčné zloženie tónu (AM syntéza).

Bežné tvary priebehu signálu generované LFO sú podobné ako pri štandardnom oscilátore – sínus, obdĺžnik, trojuholník, píla. Často sú však využívané aj „náhodné“ LFO generujúce (pseudo)náhodné hodnoty o určitej frekvencii. Vzorky medzi týmito hodnotami môžu byť interpolované a namiesto skokového signálu je tak dosiahnutý plynulejší priebeh. Je mnoho spôsobov implementácie náhodného LFO využitím tzv. *stochastických* generátorov. Príkladom je *UGen* `LFNoise2` SuperCollideru, ktorý využíva kvadratickú interpoláciu náhodných hodnôt.



Obr. 1.14: Možný priebeh interpolačného náhodného LFO

V SuperCollideri pre LFO môžu byť využité rovnaké *UGeny* ako pre normálne oscilátory. Pre zmenšenie výpočtovej náročnosti môže byť namiesto správy `ar` použitá správa `kr` (*control rate*). Naopak aj stochastické generátory môžu byť využité pre generovanie zvukového signálu. [27]

Efektové procesory

Modifikujú zvukový signál rôznorodými technikami digitálneho spracovania signálov za účelom ozvláštnenia, obohatenia, či tvarovania zvuku. Základné efekty je možné kategorizovať podľa princípu napríklad nasledujúcim spôsobom [29]:

1. Efekty založené na oneskorení signálu a modulačné efekty
    - **Delay** (Echo) – vytvára (opakujúce sa) ozveny signálu,
    - **Chorus** – „zahusťuje“ / rozlaďuje pôvodný signál miešaním viacerých kópií signálu s modulovaným oneskorením,
    - **Phaser**, **Flanger** – spektrum signálu je tvarované modulovaným hrebeňovým filtrom,
    - **Reverb**[14] – simuluje akustický priestor, pridáva zvuku „hĺbku“,

[14]Niektoré špecializované reverb algoritmy môžu presahovať kategóriu efektov založených na oneskorení signálu.

- **Tremolo** – nízkofrekvenčná modulácia zosilnenia signálu,
- **Vibrato** – modulácia výšky tónu zmenou doby oneskorenia signálu.

2. Saturačné efekty
   - **Overdrive**, **Distortion**, **Fuzz** – skreslenie signálu systémami s nelineárnou prevodovou charakteristikou (pridávajú vyššie harmonické zložky),
   - **Exciter**, **Enhancer** – kmitočtovo závislé nelineárne skreslenie.
3. Efekty pre úpravu dynamiky
   - **Kompresor**, **Limiter** – znižujú dynamický rozsah zoslabovaním signálu pri presiahnutí určitej úrovne,
   - **Expander**, **Gate** – zvyšujú dynamický rozsah zoslabovaním signálu pod určitou úrovňou.

Určité špeciálnejšie efekty môžu spadať do viacerých z týchto kategórií alebo ich presahovať.

SuperCollider má implementované *UGeny* len pre niektoré zo spomenutých efektov – napr. rôzne reverb algoritmy, ako `FreeVerb`; všetky efekty pre úpravu dynamiky sú zahrnuté v *UGene* `Compander`. Každopádne je možné každý z efektov (nie príliš zložitým spôsobom) vytvoriť pomocou rôznych *UGen*ov pre spracovanie signálov. [27]

Mixer

Kombinuje viacero zdrojov zvukového signálu do menšieho počtu kanálov. Umožňuje vyvažovať úrovne jednotlivých signálov a polohovať (*panorámovať*) ich do sterea (či viackanálového systému).

UGen `Mix` umožňuje automatizovane sčítať pole signálov tak, aby bolo zamedzené digitálnemu skresleniu. SuperCollider má aj množstvo *UGen*ov pre panorámovanie – bežne používaným pre stereo je `Pan2` využívajúci panorámovanie so zachovaním výkonu (*equal power panning*). Miešanie signálov je v SuperCollideri možné aj bežnými matematickými operátormi, ako sú operátory násobenia a sčítania. [27]

1.3.4 Vybrané typy zvukovej syntézy

Cieľom tejto časti je priblížiť niektoré princípy zvukovej syntézy, ktoré budú použité pre vytvorenie vlastných zvukových modulov v praktickej časti práce – isté bežné typy syntézy sú teda úmyselne vynechané.

Subtraktívna syntéza

Je jedným z najzákladnejších princípov zvukovej syntézy. Základom je frekvenčne bohatý zdrojový signál (pílový, obdĺžnikový či pulzný), ktorého frekvenčné zložky

sú zoslabované (subtrahované) jedným alebo viacerými filtrami. Parametre filtrov (f_m, Q) môžu byť ovládané riadiacimi signálmi (obálkami, LFO).

Subtraktívnu syntézu je možné kombinovať aj s ostatnými typmi syntézy. Napr. je možné využiť *aditívny* princíp, ak sú pred filtráciou (prípadne aj po nej) sčítané signály z rôznych zdrojov – viacero oscilátorov rôznych typov či s rôznymi základnými frekvenciami, prípadne v kombinácii so šumovými generátormi.

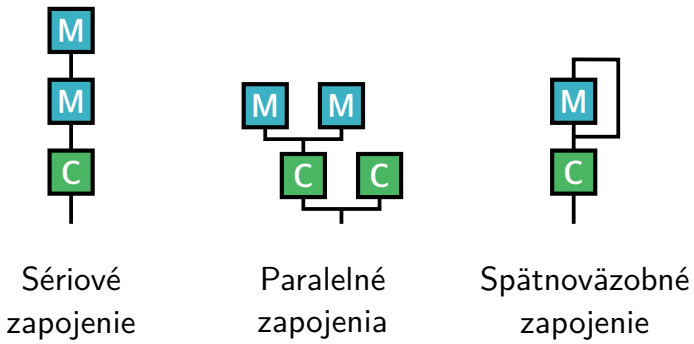
FM syntéza

Pre FM syntézu sú typické kovové, zvonovité zvukové farby. Frekvenčne bohaté spektrum je tvorené pomocou frekvenčnej modulácie – frekvencia *nosného* signálu je modulovaná iným, *modulačným*, zvukovým signálom, vďaka čomu vznikajú okolo základnej frekvencie nosného signálu *postranné* frekvenčné zložky. Inharmonicitu výsledného signálu určuje pomer nosnej a modulačnej frekvencie – celočíselné pomery vytvárajú harmonické farby, zatiaľ čo neceločíselné násobky nosnej frekvencie tvoria inharmonické farby.

Nosný harmonický signál frekvenčne modulovaný druhým harmonickým signálom sa dá vyjadriť nasledujúcou rovnicou:

$$s(t) = A_c \cos(2\pi f_c t + d_m \cos(2\pi f_m t)) \,, \tag{1.1}$$

kde A_c je amplitúda nosného signálu, f_c je nosná frekvencia, f_m je modulačná frekvencia a d_m je hĺbka modulácie. Toto je najjednoduchšia forma FM syntézy.



Obr. 1.15: Algoritmy FM syntézy (C – Carrier, M – Modulator)

Nosný signál sa označuje ako *carrier*, modulačný ako *modulator*. Tieto signály sú generované modulmi označovanými ako *operátor* – ide o oscilátory (rôznych priebehov) s vlastnou dynamickou obálkou. Spôsob vzájomného prepojenia operátorov určuje *algoritmus*. Algoritmus môže zahŕňať rôzne množstvo operátorov, ktoré môžu byť prepojené sériovo (modulator–carrier), paralelne (súčet viacerých operátorov do

jedného výstupu) alebo v spätnej väzbe (operátor moduluje svoju vlastnú frekvenciu) – obrázok 1.15. Rozličnými algoritmami FM syntézy je teda možné vytvárať aj omnoho zložitejšie signály ako ten definovaný rovnicou 1.1.

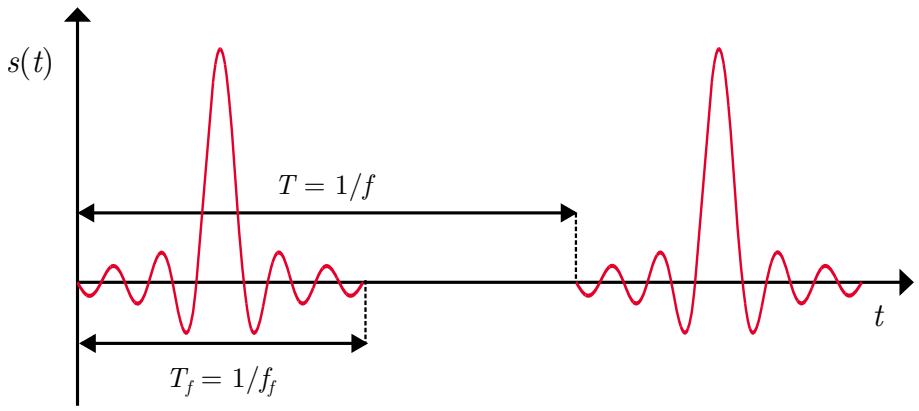
Granulárna syntéza

Granulárna syntéza tvorí zvuk pomocou krátkych zvukových úsekov, ktoré označuje jednotka nazývaná *grain* (zrno). Jeden grain je krátka *mikroakustická* udalosť trvajúca od 1 do 100 ms, obsahujúca zvukový signál tvarovaný amplitúdovou obálkou. Grain je základnou stavebnou jednotkou pre tvorbu zvukových objektov. Kombináciou tisícov zŕn je možné vytvoriť premenlivé sonické atmosféry. [21]

V praxi sú zdrojom zvukových signálov pre granulárnu syntézu často zvukové nahrávky – sample. Zrná sú zo samplu vyberané prostredníctvom parametrov ako pozícia v sampli a dĺžka zrna, ktoré môžu byť modulované (často náhodnými) riadiacimi signálmi. Tiež je možné modulovať parametre amplitúdovej obálky zrna alebo jeho panorámu a tvoriť tak amorfné „oblaky“ zvuku.

Pulsarová syntéza

Pulsarová syntéza je technika príbuzná granulárnej syntéze, pretože tiež výsledný signál vytvára z mikroudalostí, ktorými sú v tomto prípade tzv. *pulsar*y. Pulsar obsahuje *pulsaret* (podľa pojmu *wavelet*), čo je ľubovoľný priebeh nasledovaný určitým trvaním ticha. Priebeh pulsaretu môže byť napr. jedna alebo niekoľko periód funkcie sínus váhovaných oknom, prípadne pásmovo ohraničený obdĺžnikový impulz (funkcia sinc).



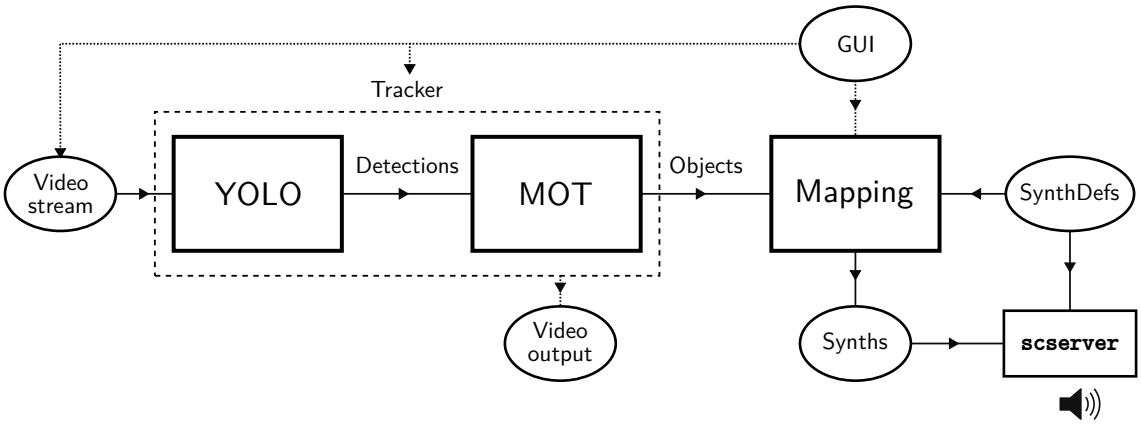
Obr. 1.16: Príklad pulsaru

Hlavnými parametrami pulsarovej syntézy sú základná frekvencia f a formantová frekvencia f_f. Základná frekvencia značí frekvenciu celkového priebehu *pulsaru* (vrátane ticha). Formantová frekvencia je obrátenou hodnotou časovej dĺžky *pulsaretu*

– viď obrázok 1.16. Formantovú frekvenciu je možné ovládať nezávisle na základnej frekvencii a meniť tak časové merítko pulsaretu spolu s trvaním ticha – pri $f \geq f_f$ je trvanie nulové.

Dôležitou vlastnosťou tejto syntézy je, že základná frekvencia môže klesnúť pod dolnú hranicu počuteľného spektra a vytvárať tak sled počuteľných pulzov, rozmazávajúc hranicu medzi rytmom a výškou tónu. Pokročilejším variantom pulsarovej syntézy môže byť konvolúcia jej výstupu so zvukovou nahrávkou. [21][30]

2 Praktická časť



Obr. 2.1: Bloková schéma základného princípu sonifikačného systému

V rámci praktickej časti práce bol navrhnutý a implementovaný pomerne komplexný a variabilný systém sonifikácie videa na základe sledovania objektov. Algoritmus zahŕňa sledovač objektov založený na detekcii, ktorého výstup je základom pre systematickú tvorbu zvukových objektov definovaných pomocou SuperCollideru – bloková schéma na obr. 2.1. Blok *Mapping* priraďuje sledovaným objektom definície zvukových objektov (*SynthDefs*) a na základe toho tvorí konkrétne zvukové objekty (*Synths*), ktorých parametre sú závislé na parametroch sledovaných objektov. Vďaka grafickému používateľskému rozhraniu (GUI) využívajúcemu uzlový editor je možné vytvoriť nespočetne mnoho rozdielnych spôsobov prepojenia obrazových a zvukových objektov a ich parametrov. Sonifikačný algoritmus pracuje v reálnom čase a je interaktívny – dovoľuje za behu meniť jeho štruktúru, použité zvukové moduly, spôsoby prepojenia a pod.

2.1 Použité technológie

Program bol implementovaný vo vysokoúrovňovom interpretovanom jazyku Python, ktorý je v súčasnosti najpoužívanejším jazykom pre prácu s modelmi umelej inteligencie a počítačovým videním. Výhodou jazyka je jednoduchá a čitateľná syntax a tiež obrovské množstvo externých modulov (dostupných cez PyPI alebo napr. GitHub), čo vytvára priaznivé podmienky pre softvérové prototypovanie.

Prvý prototyp systému používal pre detekciu objektov modely YOLOv5 [16] a pre sledovanie objektov knižnicu `motpy` [31]. Model pre detekciu bol neskôr nahradený modelom YOLOv7 [17] pre jeho efektivitu a namiesto `motpy` bola využitá univerzálnejšia a prepracovanejšia knižnica Norfair [32]. Vďaka tejto knižnici je možné

model YOLOv7 v budúcnosti jednoducho nahradiť ktorýmkoľvek iným modelom pre detekciu objektov (ktorých každým rokom pribúda).

V sledovači je okrem samotného sledovania objektov implementované aj rozpoznávanie a analýza tváre pomocou frameworku `deepface` [33]. Táto knižnica umožňuje klasifikáciu ľudských emócií, ktoré sonifikačný systém dovoľuje prepojiť so zvukom.

Komunikácia so SuperCollider serverom prebieha cez API Supriya [34], ktoré efektívne a elegantne integruje princípy jazyka `sclang` do Pythonu. Definície syntetizátorov tak v zdrojovom kóde majú podobu Python funkcií.

V neposlednom rade je implementované grafické rozhranie na frameworku Qt, využívajúc *Qt for Python* modul PySide2 [35]. Základom uzlového editoru je knižnica NodeGraphQt [36] založená takisto na PySide2.

2.2 Sledovač objektov

Sledovač objektov predstavuje trieda `Tracker` v module `tracker.py`. Tento modul zabezpečuje načítanie, spracovanie a zobrazovanie videa. V triede `Tracker` je možné definovať zdroj videa (webkameru alebo video súbor) v konštruktore alebo neskôr po vytvorení objektu. Takisto je možné nastaviť parametre detektoru - *confidence threshold*, *IOU threshold* (vysvetlené v časti 1.2.1), *image size* (veľkosť obrazu vstupujúceho do CNN). V konštruktore je inicializovaný model detektoru (prostredníctvom modulu `yolo.py`) a sledovač objektov (`norfair.Tracker`). Je nastavený parameter *distance threshold* sledovača, ktorý udáva maximálnu hodnotu vzdialenostnej funkcie (v našom prípade IOU, teda je braná inverzná hodnota), ktorá je tolerovaná u konkrétneho objektu medzi dvoma iteráciami sledovača. Použitý MOT algoritmus je podobný algoritmu SORT a využíva Kalmanov filter.

Trieda obsahuje metódu `load_video()` pre načítanie videa zo súboru, prípadne nastavenie webkamery ako zdroj videa. O načítanie video dát sa stará trieda `Video` knižnice Norfair. Metóda `read_video()` číta snímky videa a naplňuje nimi dve vyrovnávacie pamäte (dátové štruktúry *queue*) – jedna pre zobrazovanie, jedna pre spracovanie (podrobnejšie vysvetlené v časti 2.3). Metóda `show_video()` zobrazí aktuálnu snímku videa (snímku na konci zobrazovacej vyrovnávacej pamäte) spolu s ohraničujúcimi rámčekmi sledovaných objektov, prípadne aj s klasifikovanými emóciami (prostredníctvom emotikonov OpenMoji [37]).

V metóde `track()` je analyzovaná aktuálna snímka videa (snímka na konci vyrovnávacej pamäte pre spracovanie). Je vykonaná lokalizácia a klasifikácia objektov (*inferencia*) pomocou načítaného modelu, v ktorom je interne aplikované potlačenie nemaximálnych hodnôt (NMS) podľa nastavených parametrov. Ďalej sú detekcie prevedené do formátu pre sledovač Norfair a je vykonaná iterácia sledovača.

Výstupom sledovača Norfair sú aktuálne sledované („živé“) objekty pre danú snímku. Živé objekty sú prevedené na objekty vlastnej triedy `Object` a sú ukladané do slovníka (dátová štruktúra `dict`). V prípade, že objekt s daným *ID* už existuje (nejedná sa o nový objekt), je vypočítaná jeho okamžitá rýchlosť (rozdiel pozície voči predchodzej snímke) a skopírované ďalšie prípadné parametre (emócia a pod.).

Ak je sledovaný objekt osoba (trieda s identifikátorom `cls` = 0), je región obrazu určený ohraničujúcim rámčekom objektu predaný rozpoznávaču tváre, ktorého výstupom sú parametre `gender`, `emotion` a `face_reg` (ohraničujúci rámček tváre). Tvár je analyzovaná v samostatnom vlákne na pozadí v intervale určenom konštantou `FACE_PERIOD`.

Metóda `track()` vracia množiny (dátová štruktúra `set`) *aktuálnych* objektov, *nových* objektov a *zaniknutých* objektov – nové a zaniknuté objekty sú určené ako rozdiely množín aktuálnych a predchodzích objektov. Je to preto, aby bolo možné jednoducho priraďovať syntetizátory všetkým novým objektom alebo naopak uvoľniť syntetizátory patriace zaniknutým objektom.

Výpis 2.1: Spracovanie sledovaných objektov

```
for track in self.tracks:
    active_obj = Object(track)
    active_obj.frame_num = frame_num
    id = active_obj.id

    # test predošlej existencie objektu,
    # výpočet rýchlosti a kopírovanie parametrov
    if id in self.all_objs:
        ex_obj = self.all_objs[id]
        # výpočet okamžitej rýchlosti objektu
        active_obj.speed = active_obj.get_distance(ex_obj) / ( ↪
  ↪ active_obj.frame_num - ex_obj.frame_num)
        active_obj.copy_attr(ex_obj)
        self.all_objs[id].__dict__ = active_obj.__dict__.copy()
    else:
        self.all_objs[id] = active_obj

    # analýza tváre
    if self.all_objs[id].can_process_face:
        self.all_objs[id].process_face(frame)

    self.curr_objs.add(self.all_objs[id])

self.new_objs = self.curr_objs - self.__prev_objs
self.del_objs = self.__prev_objs - self.curr_objs
self.__prev_objs = self.curr_objs
for o in self.del_objs:
```

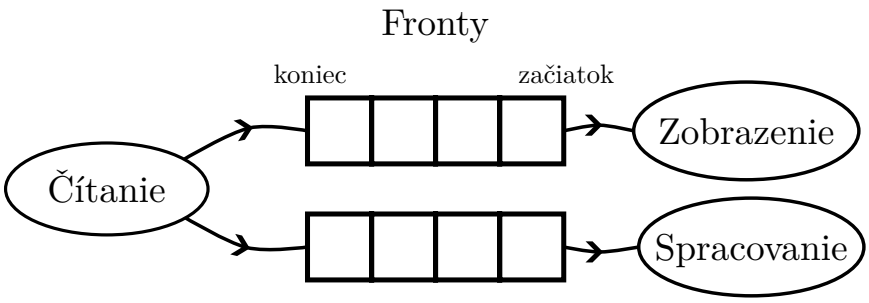
```
            self.all_objs.pop(o.id)
    return self.curr_objs, self.new_objs, self.del_objs
```

2.3 Multitasking

Spracovanie videa (detekcia a sledovanie objektov) v sonifikačnom systéme je výpočtovo náročnou operáciou. Na systémoch, ktoré to podporujú, môže byť detekcia hardvérovo akcelerovaná pomocou technológie Nvidia CUDA – spracovanie vstupu (inferencia) konvolučnou neurónovou sieťou je paralelizované pomocou množstva CUDA jadier grafickej karty.

Pri spracovaní videa v reálnom čase je prioritou plynulosť výstupu – ak je spracovanie a zobrazovanie synchrónne (výstup je zobrazený až po dokončení spracovania vstupu), môže pri náročnejšom spracovaní systém pôsobiť sekane. Pri väčšom množstve detekcii spracovanie trvá dlhšie, pretože všetky detekcie musí spracovať MOT algoritmus. Preto je v systéme čítanie, spracovanie a zobrazovanie videa rozdelené do samostatných vlákien, pričom je video zobrazované pri (viac-menej) konštantnom FPS a na spracovanie nečaká. Zobrazené sú vždy najaktuálnejšie výsledky spracovania – ak spracovanie trvá dlhšie ako čas zodpovedajúci jednej snímke, zobrazené výsledky nemusia zodpovedať zobrazovanej snímke (tzn. latencia v zobrazovaných detekciách). To je kompenzované využitím dvoch vyrovnávacích pamätí a tiež variabilným nastavovaním periódy sledovača (čo Norfair dovoľuje).

2.3.1 Vyrovnávacia pamäť



Obr. 2.2: Princíp vyrovnávacích pamätí

Pri viacvláknovom spracovaní signálov je vhodné využiť vyrovnávaciu pamäť (buffer). Typická je FIFO (first in, first out) štruktúra *queue* (front). V jazyku Python je táto dátová štruktúra dostupná pomocou dátového typu `deque`. V implementovanom systéme sú zavedené dva fronty (osobitne pre spracovanie a zobrazovanie),

pričom oba majú rovnakú dĺžku. Pri čítaní je snímka pridaná na koniec oboch frontov. Ak je zobrazovanie (a tým pádom aj čítanie) rýchlejšie ako spracovanie, čítaná snímka sa aj tak pridáva na koniec frontu pre spracovanie a snímka zo začiatku frontu musí byť zahodená – front pre spracovanie má nastavenú maximálnu dĺžku. V tom prípade vstup nie je spracovaný každú snímku a perióda sledovača je nastavená na počet snímok vyšší ako 1.

2.3.2 Vlákna

Vlákna sú implementované ako triedy `Thread` vstavaného modulu `threading`. Táto abstrakcia predstavuje vlákno operačného systému – o predávanie riadenia konkrétnemu vláknu sa stará operačný systém (tzv. *preemptívny multitasking*). [38]

Čítanie, spracovanie a zobrazenie videa je implementované pomocou troch vlákien – `read`, `process` a `display`. Operačný systém môže ktorékoľvek z týchto vlákien prerušiť a predať riadenie inému z nich (resp. úplne inému vláknu). Keď konkrétne vlákno má v istý moment vykonanú svoju úlohu (napr. `read` naplnilo vyrovnávacie pamäte), je „uspané“ metódou `time.sleep()` a operačný systém môže predať riadenie inému vláknu. Vlákno `display` vykonáva svoju úlohu v intervale určenom časom jednej snímky (1/FPS), vlákno `read` je vykonávané v intervale menšom ako tento čas (určenom parametrom `FPS_SMOOTHNESS`), vlákno `process` je uspané, len keď je front pre spracovanie prázdna alebo je sledovač zastavený – viď výpis 2.2.

Výpis 2.2: Vlákna

```
def read():
    while threads_running:
        available = tracker.running and tracker.read_video()
        if not available:
            time.sleep(0.01)
            continue
        fps = tracker.video_fps
        time.sleep(1 / (FPS_SMOOTHNESS * fps) if fps > 0 else
    ↪ 0.002)


def display():
    while threads_running:
        available = tracker.running and tracker.show_video()
        if not available:
            time.sleep(0.01)
            continue
        fps = tracker.video_fps
        time.sleep(1 / fps if fps > 0 else 0.002)
```

```
def process():
    while threads_running:
        if not tracker.running:
            time.sleep(0.01)
            continue

        result = tracker.track()
        if result is None:
            time.sleep(0.01)
            continue

        all, new, deleted = result
        ...
```

2.4 Zvukové moduly

V module `synths.py` je definovaných niekoľko syntetizátorov pomocou dekorátoru `@synthdef` API Supriya. Definície tak majú podobu funkcií, ktorých argumenty značia ovládateľné parametre syntetizátorov. Je využité množstvo *UGen*ov SuperColliderу, ktoré majú v API podobu Python tried s dekorátorom `@ugen`. Supriya zabezpečuje kompiláciu týchto konštrukcií a ponúka tak natívny spôsob komunikácie so SuperCollider serverom.

Každý z vytvorených syntetizátorov využíva globálne definovanú dynamickú obálku, ktorá zaručí fade-in a fade-out pri vytváraní alebo rušení objektov. Daný SynthDef môže mať vlastné špecifické parametre a tiež sú definované následovné všeobecné parametre, ktoré obsahuje definícia každého syntetizátoru:

`level` — hlasitosť $\langle 0, 1 \rangle$,

`pan` — panoráma $\langle -1, 1 \rangle$,

`depth` — hĺbka / pomer signálu v efektovej a výstupnej zbernici $\langle 0, 1 \rangle$,

`gate` — riadi obálku $\{0, 1\}$,

`out_bus` — ID výstupnej zbernice,

`fx_bus` — ID efektovej zbernice.

Najjednoduchší zvukový modul `beeper` je jednoduché pípanie vytvorené pomocou sínusového oscilátoru a nízkofrekvenčného pílového oscilátoru, ktorý je jeho amplitúdovou obálkou. Ďalším jednoduchým zvukovým modulom je `duster`, ktorý tvorí

náhodné pásmovo ohraničené impulzy. O niečo komplexnejší syntetizátor `droner` obsahuje 5 pulzných oscilátorov, ktoré môžu byť vzájomne rozladďované, 3 LFO modulujúce zvukové parametre (napr. striedu oscilátorov – PWM), pričom výsledný zložený signál je filtrovaný LPF II. rádu.

Ďalej SynthDef `operator` implementuje FM syntézu s 3 operátormi zapojenými v sérii, modul `pulsar` realizuje pulsarovú syntézu, modul `drummer` generuje rytmy pomocou zvukov elektronických bicích vytvorených subtraktívnou syntézou.

Zvukové moduly `player` (jednoduchý sampler) a `grainer` (granulárny syntetizátor) tvoria zvuk pomocou samplov z databanky využitím abstrakcie `Buffer`.

Stručná dokumentácia všetkých vytvorených zvukových modulov obsahujúca popis ich parametrov je uvedená v prílohe A.

Okrem spomenutých syntetizátorov je definovaný aj efektový procesor `reverb`, ktorý využíva implementáciu algoritmu FreeVerb obsiahnutú v SuperCollideri. Tento procesor je zaradený za efektovú zbernicu, takže signály jednotlivých syntetizátorov posielané do reverbu ovplyvňuje ich parameter `depth`. Použitý je aj *UGen* `Limiter`, ktorý je zaradený za hlavnú zbernicu, aby pri veľkom počte zvukových objektov bolo zamedzené digitálnemu skresleniu (*clipping*).

Výpis 2.3: Implementácia zvukového modulu `pulsar`

```
@synthdef()
def pulsar(fx_bus, gate=1, out_bus=0, level=0.5, pan=0, depth
    ↪ =0.4, freq=1, formant_freq=50, sine_cycles=2, window_curve
    ↪ =2):
    # globálna dynamická obálka
    envelope = EnvGen.kr(envelope=global_env, gate=gate,
    ↪ done_action=2)
    # fáza pulsaretu vytvorená pílovým generátorom
    phase = LinLin.ar(LFSaw.ar(freq, 1), -1, 1, 0, 1) *
    ↪ formant_freq / freq
    # okno vytvorené inverziou rozsahu a umocnením signálu fázy
    window = LinLin.ar(phase, 0, 1, 1, 0) ** window_curve
    # signál pulsaretu vypočítaný pomocou funkcie sin
    sine = UnaryOpUGen(
           source=phase*2*math.pi*sine_cycles.floor(),
           special_index=supriya.UnaryOperator.SIN,
           calculation_rate=supriya.CalculationRate.AUDIO)
    # váhovanie oknom a zabezpečenie fázy ticha v pulsare
    sig = sine * window * (phase < 1)
    out_sig = level * envelope * Pan2.ar(sig, pan)
    Out.ar(fx_bus, out_sig * depth)
    Out.ar(out_bus, out_sig * (1 - depth))
```

2.4.1 Zvuková databanka

Pre priradenie konkrétnych zvukových vzoriek (samplov) sledovaným objektom bola automatizovane vytvorená zvuková databanka pomocou API otvorenej online databázy Freesound [39]. Zvuky boli hľadané pre každú triedu objektov datasetu COCO a tiež pre každú triedu emócie + pohlavia z modelov frameworku `deepface`. Požiadavky vyhľadávania obsahovali len názvy tried a obmedzenie dĺžky do 15 sekúnd. Bola tak vytvorená databanka s približne 1200 zvukovými súbormi.

2.5 Priraďovanie zvukových objektov

Systém prepojenia obrazových a zvukových objektov je realizovaný v module `mapping`, v ktorom sú definované dve abstrakcie – `SynthMapping` a `ParameterMapping` s nasledujúcimi konštruktormi:

```
SynthMapping(object_classes: List[int],
             synthdef: supriya.SynthDef)

ParameterMapping(synthdef: supriya.SynthDef,
                 synth_attr: str,
                 obj_attr: str=None,
                 scaling: Callable[[float], float]=(lambda x: x))
```

Pomocou abstrakcie `SynthMapping` je možné definovať priradenie konkrétneho typu zvukového modulu (SynthDef) konkrétnym triedam obrazových objektov, definovaných pomocou zoznamu `object_classes`.

Abstrakcia `ParameterMapping` prepojuje parameter daného typu zvukového modulu s parametrom obrazového objektu. Spravidla normalizovaný obrazový parameter je pred priradením škálovaný danou funkciou `scaling`. Sú to funkcie jednej reálnej premennej a v programe sú väčšinou implementované ako tzv. `lambda` funkcie. Vďaka nim je možné previesť napr. normalizovanú horizontálnu polohu objektu x v rozsahu $\langle 0, 1\rangle$ na frekvenciu oscilátoru.

Modul obsahuje funkcie pre pridávanie, odstraňovanie a modifikovanie inštancií `SynthMapping` a `ParameterMapping`, ktoré sú uchovávané v globálnych zoznamoch tohto modulu. Funkcie `scaling` je tiež možné reťaziť pomocou funkcie `chain()`.

Je možné vytvoriť aj prepojenia parametrov, ktoré sú aplikované pre všetky typy zvukových modulov (užitočné pre globálne nastavovanie parametrov `level`, `pan` a `depth`), ak je atribút `synthdef` abstrakcie `ParameterMapping` nastavený na `None` – generické prepojenia. Generické prepojenia sú prepísané prepojeniami, ktoré sú definované pre konkrétny typ zvukového modulu.

To, či sa konkrétne prepojenie parametrov aplikuje pre konkrétny syntetizátor, rozhoduje funkcia `mapping_applies()` uvedená vo výpise 2.4.

Výpis 2.4: Rozhodovanie realizácie prepojenia pre konkrétny syntetizátor

```
def mapping_applies(mapping: ParameterMapping, synth: supriya.
    ↪ Synth):
    # vzťahuje sa negenerické prepojenie k tomuto syntetizátoru?
    if mapping.synthdef is not None:
        return synth.synthdef.name == mapping.synthdef.name
    # vráti False, ak je toto generické prepojenie prepísané
    return not any(m.synthdef.name == synth.synthdef.name and
    ↪ mapping.synth_attr == m.synth_attr
                   for m in param_mappings if m.synthdef is not
    ↪ None)
```

Pridávanie konkrétnych syntetizátorov a nastavovanie parametrov je potom vykonávané vo vlákne `process`. Pre všetky nové obrazové objekty, pre ktoré sa aplikujú vytvorené prepojenia sú vytvorené nové syntetizátory. Slovník `synth_map` drží informácie o aktuálnom priradení jednotlivých syntetizátorov k sledovaným objektom – položka s kľúčom `id` obsahuje zoznam syntetizátorov, ktoré sú priradené objektu s daným identifikátorom. Syntetizátory, ktoré sú priradené v `synth_map` už zaniknutým objektom, sú odstránené. Pre prípad, že bolo prepojenie vytvorené až po vzniku obrazového objektu (objekt nie je nový), sú vytvorené syntetizátory pre objekty, ktoré nie sú v `synth_map`, ale existuje `SynthMapping`, ktorý sa pre nich aplikuje. Podobne sú odstránené syntetizátory, ktorých prepojenia boli odstránené, ale objekt ešte nezanikol. Teda prepojenia nie je problém vytvárať a odstraňovať v reálnom čase počas prehrávania videa. Prepojenia parametrov sú aplikované pomocou metódy `ParameterMapping.apply()`, v ktorej je nastavený príslušný parameter syntetizátoru.

S rôznymi parametrami syntetizátorov je možné prepojiť následovné parametre obrazových objektov:

`x` — horizontálna pozícia $\langle 0, 1 \rangle$,

`y` — vertikálna pozícia $\langle 0, 1 \rangle$,

`area` — plocha $\langle 0, 1 \rangle$,

`speed` — okamžitá rýchlosť $\langle 0, 1 \rangle$,

`class_id` — identifikátor triedy $\{0, ..., 79\}$,

`sex_id` — identifikátor pohlavia $\{0, 1\}$,

`emo_id` — identifikátor emócie $\{0, ..., 6\}$.

Parametre `sex_id` a `emo_id` môžu byť prepojené s parametrom `buffer_id` príslušných syntetizátorov – je potom nastavovaná príslušná zvuková vzorka podľa

emócie či pohlavia. Parametre, ktoré sú prepojené s parametrom `pan`, nemusia byť škálované a sú automaticky prevedené do rozsahu $\langle -1, 1 \rangle$.

Zoznam všetkých tried objektov, pohlavia a emócií spolu s ich identifikátormi je uvedený v prílohe B.

2.6 Kvantizácia výšky tónu

Špeciálnou škálovaciou funkciou je metóda `snap()` triedy `Quantizer` z modulu `pitch_quantization.py`. Trieda `Quantizer` umožňuje už naškálovaný parameter kvantovať do rôznych tónových terénov a je tak možné v sonifikácii dosahovať aj muzikálnejších výsledkov.

Trieda využíva externé moduly `musical_scales` a `pitch_tools` dostupné z platformy PyPI. Modul `musical_scales` obsahuje definície množstva bežných aj exotických hudobných stupníc v dvanásťtónovom teréne. Výstupom sú názvy tónov v stupnici, ktoré musia byť prevedené na frekvencie, k čomu pomáha modul `pitch_tools`. Pri prevode je možné nastaviť referenčný tón ladenia (východzia hodnota je A4 = 440 Hz). Prevedené frekvencie sú posunuté do vyšších a nižších oktáv pomocou vlastnej funkcie `expand_f_scale()` – výpis 2.5.

Výpis 2.5: Funkcia pre rozšírenie zoznamu frekvencií do oktáv

```
def expand_f_scale(freqs):
    min_f = 0.5
    max_f = 20_000

    expanded = freqs.copy()
    shifted = freqs

    while len(upper := [2*x for x in shifted if 2*x <= max_f]):
        # pridanie vyššej oktávy na koniec zoznamu
        expanded = expanded + upper
        shifted = upper
    shifted = freqs
    while len(lower := [x/2 for x in shifted if x/2 >= min_f]):
        # pridanie nižšej oktávy na začiatok zoznamu
        expanded = lower + expanded
        shifted = lower

    return expanded
```

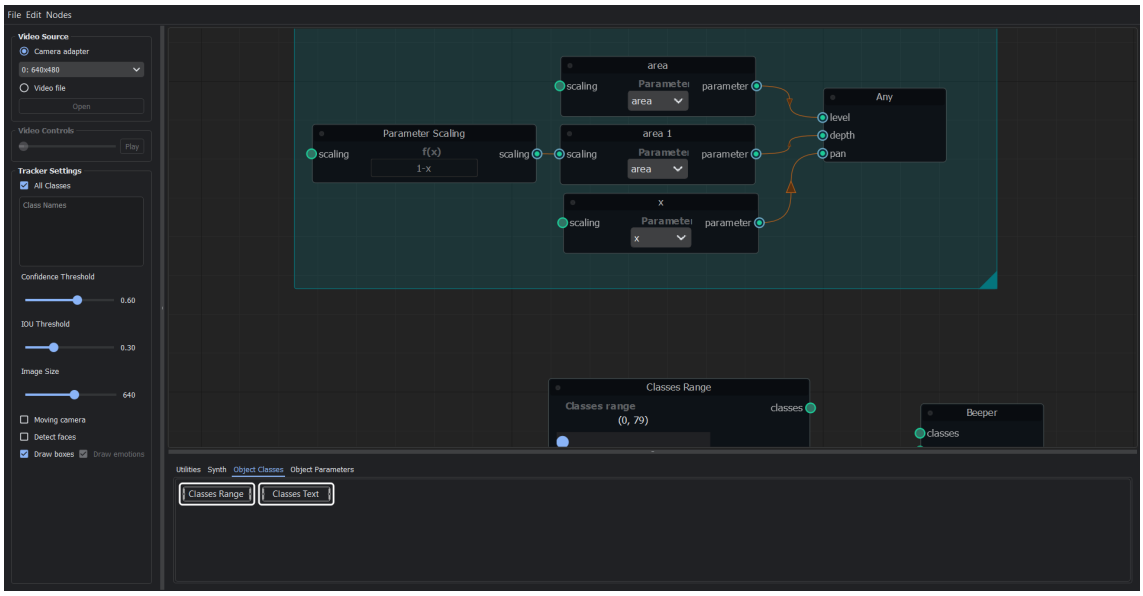
Výsledkom je zoradený zoznam frekvencií v rozsahu od 0.5 Hz do 20 kHz spadajúcich do danej hudobnej stupnice. V metóde `snap()` je pomocou binárneho vyhľadávania (implementovaného vo vstavanom module `bisect`) vybraná z tohto zoznamu

frekvencia, ktorá je najbližšia argumentu metódy, a frekvencia je tak kvantovaná do tónového terénu.

Pri sonifikácii je možné využiť viac rôznych inštancií triedy `Quantizer` a tvoriť tak aj polytonálne štruktúry. Preladením referenčných tónov niektorých inštancii je možné dosiahnuť aj určitý stupeň mikrotonality[1].

2.7 Grafické používateľské rozhranie

Dôležitou súčasťou vytvoreného programu je grafické používateľské rozhranie umožňujúce v reálnom čase upravovať štruktúru sonifikačného systému. Obsahuje najmä uzlový editor pre prepájanie sledovaných objektov a zvukových modulov a tiež ovládacie prvky pre voľbu zdroja videa a nastavenie sledovača objektov.



Obr. 2.3: Okno grafického rozhrania aplikácie

2.7.1 Uzlový editor

Tento typ grafického rozhrania bol zvolený pre jeho pružnosť, modularitu, efektivitu a prehladnosť. Alternatívou jednoduchšou na implementáciu by mohlo byť napr. niekoľko modulačných matíc – to by však pri množstve rôznych parametrov pôsobilo chaoticky a bol by problém so zahrnutím možnosti škálovania parametrov. Prípadne

[1]Celkový tónový terén tak bude obsahovať aj hudobné intervaly menšie alebo väčšie ako temperovaný poltón. Mikrointervalika sa spravidla bude vyskytovať bez použitia kvantizácie, vtedy sa však nedá hovoriť o definovanom tónovom teréne.

by bolo možné prepojenia objektov, zvukových modulov a ich parametrov namiesto GUI realizovať pomocou textových príkazov, no tento spôsob práce by bol oproti uzlovému editoru pomalší a menej intuitívny.

Grafické rozhranie je spôsobom práce podobné programovacím prostrediam Max a Pure Data (v tomto prípade sa však nejedná o plnohodnotné programovacie prostredie, iba o front end jednoduchšieho systému). Editory tohoto typu nájdeme aj v rôznych aplikáciách pre prácu s grafikou alebo videom, ako napr. Blender či DaVinci Resolve.

V editore je možné tvoriť „patche“ definujúce ktoré triedy objektov sú prepojené s ktorými zvukovými modulmi a ako sú vzájomne prepojené ich parametre. Vďaka použitej knižnici NodeGraphQt je možné patche ukladať do `.json` súborov. Pretiahnutím súborov myšou do editoru je možné kombinovať aj viacero patchov.

Pre vytvorenie front endu systému museli byť vytvorené vlastné typy uzlov, ktoré budú v tejto časti stručne predstavené.

Classes Text a Classes Range

Tieto uzly dovoľujú špecifikovať zoznam tried objektov, ktorý je možné priradiť zvukovým modulom. V uzle *Classes Text* sú triedy definované ich slovným názvom, pričom je implementované automatické doplňovanie. Názvy tried sú oddelené čiarkou. Uzol *Classes Range* umožňuje zoznam tried definovať rýchlejšie pomocou rozsahu číselných identifikátorov tried definovaným uzavretým intervalom.

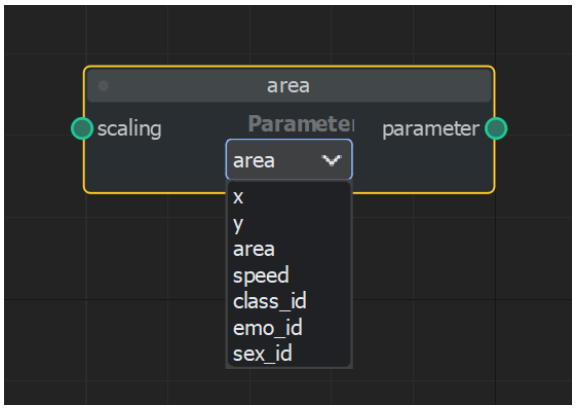
(a) Classes Text

(b) Classes Range

Obr. 2.4: Uzly pre definovanie zoznamu tried

Object Parameter

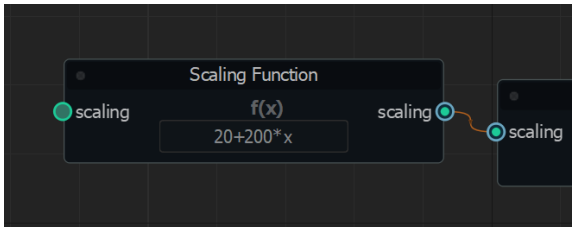
Tento uzol definuje daný parameter obrazového objektu. Na jeho vstup je možné pripojiť uzol definujúci škálovanie parametru. Naškálovaný parameter na výstupe je možné prepojiť s parametrom syntetizátoru.



Obr. 2.5: Uzol *Object Parameter*

Scaling Function

V textovom poli tohoto uzla je možné definovať škálovaciu funkciu premennej x pomocou Python výrazu. Výraz musí obsahovať znak `x` označujúci závislú premennú, môže obsahovať všetky bežné Python operátory (`+, -, *, /, //, **` ...) alebo aj matematické funkcie (`sin(), cos(), log(), exp()`...). Pri zadaní neplatného výrazu je priradená funkcia $f(x) = x$. Škálovacie uzly je možné reťaziť.



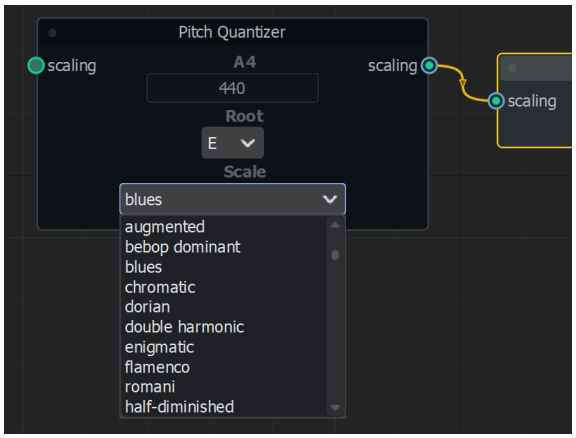
Obr. 2.6: Uzol *Scaling Function*

Pitch Quantizer

Špeciálny druh škálovacieho uzla umožňujúci kvantizáciu výšky tónu do hudobných stupníc. Je možné definovať základný tón, druh stupnice aj referenčnú frekvenciu A4 – obr. 2.7.

Uzly zvukových modulov

Každý zvukový modul má vlastný uzol. Uzly sú vytvorené automatizovane podľa syntetizátorov definovaných v module `synths.py`. Každý parameter syntetizátoru, ktorý je možné prepojiť s parametrom objektu, má podobu vstupného portu uzla –



Obr. 2.7: Uzol *Pitch Quantizer*

viď obrázok 2.8. Daný zvukový modul môže byť v grafe iba raz (inak by sa vytvárali duplicitné prepojenia).

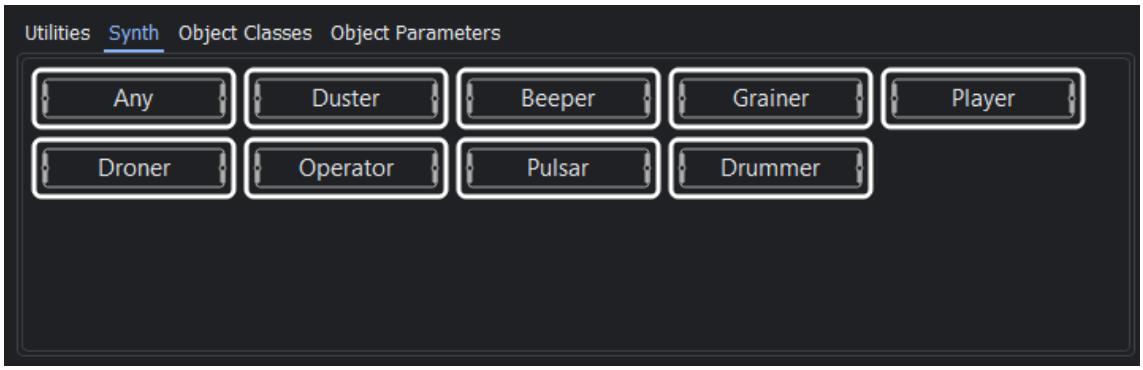
Obr. 2.8: Uzly zvukových modulov

Paleta uzlov

Uzly je možné do grafu vkladať výberom z palety v spodnej časti okna, kde sú typy uzlov rozdelené do kategórii – obr. 2.9. Alternatívny spôsob je prehľadávanie uzlov stlačením klávesy Tab. Paleta obsahuje aj vstavaný prvok *Backdrop* umožňujúci zoskupenie uzlov.

Vyhodnocovanie grafu

Prepojenia uzlov sú vyhodnocované v reálnom čase. Logika je implementovaná vo funkciách, ktoré sú prepojené s Qt *signálmi* (ekvivalent *callback* funkcií).



Obr. 2.9: Paleta uzlov

Napr. pri prepojení dvoch uzlov je vyslaný signál `ports_connected` a zavolaná pripojená funkcia s argumentmi špecifikujúcimi, ktoré porty boli prepojené. Vo funkcii je podľa portov vyhodnotené aké uzly boli prepojené (pomocou Python funkcie `isinstance()`) a prípadne je vytvorený príslušný objekt `SynthMapping` alebo `ParameterMapping`. Vo funkcii `ports_disconnected()` je implementovaná inverzná logika. Pri vytvorení neplatných prepojení (tzn. žiadna podmienka v `ports_connected()` nie je splnená) je prepojenie zrušené.

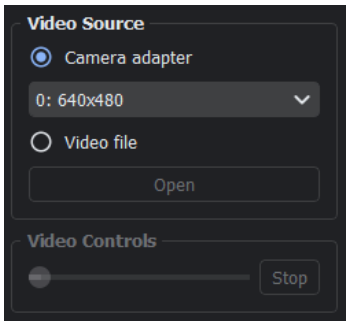
Pri prepojení parametrov (resp. pri úprave už pripojeného uzlu `ObjectParameterNode` či `ParameterScalingNode`) je potrebné rekurzívne nájsť všetky zreťazené škálovacie funkcie. To je implementované pomocou atribútu (metódy s dekorátorom `@property`) uzlu `ParameterScalingNode.chained_function`. Podobne atribút `ParameterScalingNode.connected_param_nodes` rekurzívne hľadá všetky uzly `ObjectParameterNode`, s ktorými je škálovací uzol prepojený.

2.7.2 Ovládacie prvky sledovača

V ľavej časti okna sa nachádza panel, pomocou ktorého je možné ovládať sledovač objektov. V jeho hornej časti je možné zvoliť zdroj videa – obr. 2.10. Nachádza sa tu menu, v ktorom je možné zvoliť jednu z pripojených kamier. Prípadne je možné ako zdroj zvoliť video súbor (podporované sú formáty `.mp4`, `.avi` a `.mkv`). Pri zvolení video súboru sú dostupné ovládacie prvky pre posúvanie prehrávanej pozície alebo pozastavenie videa.

V spodnej časti sú ovládacie prvky parametrov sledovača – obr. 2.11. Ak nás zaujímajú len konkrétne triedy objektov, je možné ich špecifikovať podľa ich názvu v príslušnom textovom poli a detektor bude detekovať len tieto triedy – výstup vtedy bude prehľadnejší a MOT algoritmus bude pracovať rýchlejšie.

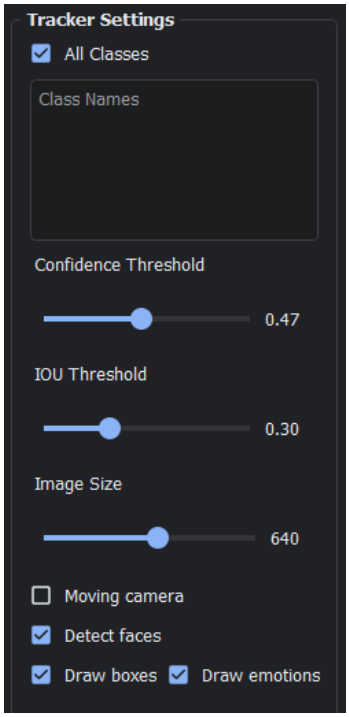
Posuvníkmi je možné v reálnom čase ladiť príslušné parametre sledovača. Zvyšovaním parametru *Confidence Threshold* sa zníži počet falošných detekcií (a tým



Obr. 2.10: Ovládacie prvky videa

pádom aj celkový počet objektov). Znižovaním hodnoty parametru *IOU Threshold* sa zníži počet duplicitných detekcií, ale môže to aj znemožniť detekciu dvoch objektov tej istej triedy blízko seba (s prekrývajúcimi sa ohraničujúcimi rámčekmi). Parameter *Image Size* definuje na akú šírku bude zmenená veľkosť obrazu pred jeho vstupom do detektoru. Zmenšenie hodnoty parametru urýchli detekciu, ale zmenší jej presnosť.

Ovládacím prvkom *Moving camera* je možné zapnúť kompenzáciu pohybu kamery, *Detect faces* zapína / vypína analýzu tváre, *Draw boxes* a *Draw emotions* povoľujú vykresľovanie rámčekov a emotikonov.



Obr. 2.11: Ovládacie prvky sledovača

2.8 Využitie

Vďaka modularite a variabilite vytvoreného systému je možné ho využiť v množstve rozdielnych situácií. Možnosť vypínať jednotlivé časti systému, ako analýza tváre či možnosť filtrácie určitých tried objektov, dovoľuje používateľovi zamerať sa na konkrétny fenomén, ktorý bude sonifikovaný. Výsledný zvuk bude závisieť od použitých zvukových modulov a spôsobu prepojenia parametrov.

Systém môže nájsť uplatnenie napríklad v nasledujúcich oblastiach:

1. **Performatívne situácie**
   Uzlový editor dovoľuje spôsob práce podobný živému programovaniu hudby (live coding). Je tak možné systém použiť v rámci hudobného či sound-artového vystúpenia, s využitím predpripravených patchov alebo vytváraním prepojení na mieste. Vstupom môže byť pripravené video alebo živý obraz z kamery.
2. **Umelecké inštalácie**
   Vytvorením komplexnejších patchov je možné definovať nepredvídateľné správanie – zvuk môže závisieť napr. od toho, aké typy objektov sa práve nachádzajú v obraze. Systém teda môže fungovať bez obsluhy v kontexte inštalácie. V takejto situácii by však bolo vhodnejšie, keby sa patche obmieňali v čase na základe rôznych parametrov. Automatizované načítavanie patchov program zatiaľ neumožňuje, avšak mohlo by to byť realizované pomocou externého skriptu.
3. **Kompozícia**
   Systém môže byť prostriedkom pre tvorbu zaujímavých zvukových či hudobných štruktúr a byť tak nástrojom pre skladateľov elektroakustickej hudby. Spadá do kategórie kompozičných programov typu prevodník [2].
4. **Sound design audiovizuálneho diela**
   Sonifikačný systém je možné využiť pre experimentálnu tvorbu zvukovej stopy audiovizuálneho diela. Obrazová zložka diela je vstupom pre systém, pričom na základe nej je zvuková zložka tvorená algoritmicky, čo zabezpečí ich vzájomný súlad – pri štandardnej práci je toto pracnou záležitosťou. Zvukový výstup systému môže byť použitý ako jedna z vrstiev výsledného sound designu.
5. **Analytická alebo navigačná sonifikácia**
   Napriek tomu, že hlavným cieľom bola umelecká sonifikácia, systém umožňuje aj iné formy sonifikácie. Napr. zvukový modul *Beeper* dokáže síce jednoducho, ale zrozumiteľne komunikovať dáta pomocou zvuku. Prípadne modul *Player* je schopný prehrávať zvukové vzorky na základe názvu triedy objektu. Teda je možné systém použiť pre analytické účely. Pre navigačnú sonifikáciu v aplikáciách v reálnom svete systém v súčasnej podobe pre jeho technickú náročnosť nie je najvhodnejším prostriedkom, avšak pri istých úpravách by jeho základ

mohol slúžiť aj tomuto účelu.

Je vhodné podotknúť, že pre dosiahnutie zmysluplných výsledkov musí používateľ do istej miery vedieť, ako systém funguje. Intuitívnosť nebola hlavným parametrom pri tvorbe používateľského rozhrania. Cieľom bolo poskytnúť priestor pre experimentáciu ľuďom, ktorí sa nejakým spôsobom venujú problematike či už umelej inteligencie, počítačového videnia, sound designu alebo algoritmickej kompozície.

Záver

Táto diplomová práca sa venovala sonifikácii videa modernými technikami počítačového videnia založenými na umelej inteligencii.

V teoretickej časti boli predstavené možnosti umeleckej sonifikácie vzhľadom k algoritmickej kompozícii, generatívnej hudbe a sound artu. Bol vysvetlený princíp vybraných algoritmov počítačového videnia umožňujúcich detekciu a sledovanie objektov. Tiež boli zvážené platformy, pomocou ktorých je možné sonifikáciu realizovať, pričom sa ako najvhodnejšie ukázalo prostredie SuperCollider. Teoretická časť tiež obsahuje zhrnutie základov zvukovej syntézy v kontexte tohoto prostredia a uvádza aj niektoré menej tradičné druhy syntézy ako sú granulárna či pulsarová syntéza. Tieto poznatky boli zužitkované v praktickej časti práce.

Výsledkom praktickej časti je implementácia variabilného modulárneho sonifikačného systému pracujúceho v reálnom čase, ktorý môže byť aplikovaný v rozličných kontextoch. Aplikácia je napísaná v programovacom jazyku Python, využívajúc množstvo externých modulov. Hlavnými využitými prostriedkami počítačového videnia sú detekčný model YOLOv7, MOT knižnica Norfair a knižnica pre analýzu tváre Deepface. Pre tvorbu zvuku je použité SuperCollider API pre Python s názvom Supriya.

Sonifikačný systém ponúka množstvo zvukových modulov rozličného charakteru, využívajúcich rôzne typy zvukovej syntézy a aj obsiahlu, automatizovane vytvorenú zvukovú databanku. Spôsob prepojenia týchto modulov s dátami získanými z obrazu systém dovoľuje používateľovi definovať pomocou pomerne prepracovaného grafického rozhrania, využívajúceho uzlový editor. Celý systém je založený na objektovom princípe – v obraze sú sledované vizuálne objekty, ktorým sú podľa definovaných prepojení algoritmicky priraďované zvukové objekty. Uzlový editor okrem iného umožňuje škálovanie parametrov objektov a aj ich kvantizáciu do hudobných stupníc. Je tak možné rôznorodými spôsobmi prepájať obraz a zvuk.

Modularita systému dovoľuje ďalej ho rozširovať. Môže byť vytvorené množstvo ďalších zvukových modulov alebo typov uzlov editoru. Tiež je možné integrovať ďalšie modely a algoritmy počítačového videnia, prípadne aj generatívne modely umelej inteligencie. V rámci možností diplomovej práce bola však vytvorená uspokojivá forma aplikácie, ktorá zatiaľ nemá dostupnú obdobu.

Literatúra

[1] SUCHÁNEK, Jiří. *Sonifikace v hudební kompozici a sound artu*. Brno, 2021. Dizertačná práca. Janáčkova akademie múzických umění v Brně.

[2] DLOUHÝ, Dan. *Počítačem podporovaná algoritmická kompozice*. Brno: Nakladatelství JAMU, Novobranská 3, 662 15 Brno, 2018. ISBN 978-80-7460-141-5.

[3] FOO, Brian. Music Eclipticalis: A Song Using the Position of the Stars As Seen From Earth. In: *Data-Driven DJ* [online]. 2015: Brian Foo, c2015 [cit. 2022-12-01]. Dostupné z: `https://datadrivendj.com/tracks/stars/`

[4] PARVIAINEN, Tero. How Generative Music Works: A Perspective. *Teropa.info* [online]. [cit. 2022-10-23]. Dostupné z: `https://teropa.info/loop/`

[5] BAINTER, Alex. Introduction to Generative Music. In: *Medium* [online]. San Francisco, California: A Medium Corporation, 2019 [cit. 2022-10-23]. Dostupné z: `https://medium.com/@alexbainter/introduction-to-generative-music-91e00e4dba11`

[6] WOOLLER, Rene, Andrew R. BROWN, Eduardo MIRANDA, Rodney BERRY a Joachim DIEDERICH. In: *Generative arts practice: a Creativity & Cognition Symposium*. Sydney: Creativity & Cognition Studio Press, 2005, s. 109-124. ISBN 0-9751533-3-1.

[7] Soil Choir. *Jiří Suchánek* [online]. Brno: Jiří Suchánek, 2022 [cit. 2022-11-30]. Dostupné z: `https://www.jiri-suchanek.net/project/soil-choir/`

[8] JAKOVICH, Joanne a Kirsty BEILHARZ. ParticleTecture: Interactive granular soundspaces for architectural design. *Conference on New Interfaces for Musical Expression* [online]. New York, 2007, **2007**(NIME07) [cit. 2022-12-01]. Dostupné z: doi:10.1145/1279740.1279775

[9] ROZANOFF, Seth. SONDA Festival 2022. *Computer Music Journal*. 2023, **45**(4), 73-74. ISSN 0148-9267. Dostupné z: doi:doi:10.1162/comj_r_00628

[10] Iannis Xenakis. In: *Lines & Marks* [online]. Berlin: Romeo Alaeff, 2015 [cit. 2022-12-01]. Dostupné z: `https://linesandmarks.com/iannis-xenakis-observations/`

[11] CHION, Michel, John DACK a Christine NORTH. *Guide to sound objects* [online]. English translation by John Dack and Christine North. London: Electro-Acoustic Resource Site, 2009 [cit. 2022-11-22]. Dostupné z: `http://ears.huma-num.fr/onlinePublications.html`

[12] KLETTE, Reinhard. *Concise Computer Vision: An Introduction into Theory and Algorithms*. London: Springer, 2014. ISBN 978-1-4471-6319-0.

[13] SZELISKI, Richard. *Computer Vision: Algorithms and Applications* [online]. 2nd Edition. New York: Springer, 2022 [cit. 2022-11-11]. ISBN 3030343715. Dostupné z: `https://szeliski.org/Book/`

[14] KEITA, Zoumana. YOLO Object Detection Explained. In: *Datacamp* [online]. New York, USA: DataCamp, 2022 [cit. 2022-11-17]. Dostupné z: `https://www.datacamp.com/blog/yolo-object-detection-explained`

[15] REDMON, Joseph, Santosh DIVVALA, Ross GIRSHICK a Ali FARHADI. *You Only Look Once: Unified, Real-Time Object Detection* [online]. 2015 [cit. 2022-11-21]. Dostupné z: `https://arxiv.org/abs/1506.02640`

[16] *YOLOv5* [online]. GitHub, 2020 [cit. 2022-11-17]. Dostupné z: `https://github.com/ultralytics/yolov5`

[17] *YOLOv7* [online]. GitHub, 2022 [cit. 2022-11-17]. Dostupné z: `https://github.com/WongKinYiu/yolov7`

[18] *COCO: Common Objects in Context* [online]. COCO Consortium, 2015 [cit. 2022-11-21]. Dostupné z: `https://cocodataset.org/`

[19] BEWLEY, Alex, Zongyuan GE, Lionel OTT, Fabio RAMOS a Ben UPCROFT. Simple online and realtime tracking. In: *2016 IEEE International Conference on Image Processing (ICIP)* [online]. Ithaca: IEEE, 2016, s. 3464-3468 [cit. 2022-11-21]. ISBN 978-1-4673-9961-6. ISSN 2381-8549. Dostupné z: doi:10.1109/ICIP.2016.7533003

[20] WOJKE, Nicolai, Alex BEWLEY a Dietrich PAULUS. Simple online and realtime tracking with a deep association metric. In: *2017 IEEE International Conference on Image Processing (ICIP)* [online]. Beijing: IEEE, 2017, s. 3645-3649 [cit. 2022-11-21]. ISBN 978-1-5090-2175-8. ISSN 2381-8549. Dostupné z: doi:10.1109/ICIP.2017.8296962

[21] ROADS, Curtis. *Microsound*. Cambridge, Mass: MIT Press, 2001, xii, 409 s. 1 CD. ISBN 0-262-18215-7.

[22] WISHART, Trevor. *On Sonic Art*. Amsterdam: Harwood-Academic, 1996. ISBN 371865847X.

[23] BUFFONI, Louis-Xavier. Working with Object-Based Audio. In: *Audiokinetic Blog* [online]. Montréal: Audiokinetic, 2016 [cit. 2022-11-26].

Dostupné z: https://blog.audiokinetic.com/en/working-with-object-based-audio/

[24] Max (software). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-11-27]. Dostupné z: https://en.wikipedia.org/wiki/Max_(software)

[25] *Max Documentation* [online]. San Francisco: Cycling '74, c1997-2022 [cit. 2022-11-27]. Dostupné z: https://cycling74.com/learn/documentation

[26] SuperCollider. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-11-29]. Dostupné z: https://en.wikipedia.org/wiki/SuperCollider

[27] *SuperCollider 3.12.2 Documentation* [online]. SuperCollider [cit. 2022-11-29]. Dostupné z: https://doc.sccode.org/

[28] ROADS, Curtis. *The computer music tutorial*. 1. Cambridge, MA, USA: MIT Press, 1996. ISBN 0252181583.

[29] SCHIMMEL, Jiří. *Studiová a hudební elektronika* [online]. Druhé vydání. Brno: Vysoké učení technické v Brně, 2015 [cit. 2023-05-12]. ISBN 978-80-214-4452-2. Dostupné z: https://www.vutbr.cz/

[30] HO, Nathan. Pulsar Synthesis. In: *Nathan Ho* [online]. Nathan Ho, c2023, 31. 7. 2022 [cit. 2023-05-13]. Dostupné z: https://nathan.ho.name/posts/pulsar-synthesis/

[31] *Motpy* [online]. GitHub, 2020 [cit. 2022-12-09]. Dostupné z: https://github.com/wmuron/motpy/

[32] *Norfair* [online]. Tryolabs, c2022 [cit. 2022-12-10]. Dostupné z: https://tryolabs.github.io/norfair/

[33] *Deepface* [online]. Github, 2020 [cit. 2022-12-09]. Dostupné z: https://github.com/serengil/deepface

[34] *Supriya* [online]. GitHub, 2014 [cit. 2022-12-09]. Dostupné z: https://github.com/josiah-wolf-oberholtzer/supriya

[35] *PySide2* [online]. Python Software Foundation, 2018 [cit. 2023-05-05]. Dostupné z: https://pypi.org/project/PySide2/

[36] *NodeGraphQt* [online]. Johnny Chan, c2023 [cit. 2023-05-05]. Dostupné z: http://chantonic.com/NodeGraphQt/api/html/index.html

[37] *OpenMoji* [online]. OpenMoji, 2018 [cit. 2023-05-05]. Dostupné z: https://openmoji.org/

[38] UNNA, Jacob. Concurrency in Python: Cooperative vs Preemptive Scheduling. In: *Medium* [online]. San Francisco, California: A Medium Corporation, 2020 [cit. 2023-05-06]. Dostupné z: https://medium.com/fullstackai/concurrency-in-python-cooperative-vs-preemptive-scheduling-5feaed7f6e53

[39] *Freesound API* [online]. Barcelona: Music Technology Group at Universitat Pompeu Fabra, 2005 [cit. 2022-12-09]. Dostupné z: https://freesound.org/docs/api/

[40] Real-Time Object Detection on COCO. *Papers With Code* [online]. Menlo Park: Meta AI, 2021 [cit. 2022-12-10]. Dostupné z: https://paperswithcode.com/sota/real-time-object-detection-on-coco

Zoznam symbolov a skratiek

API Application Programming Interface (rozhranie pre programovanie aplikácií)

BPF Band Pass Filter (pásmovopriepustný filter)

BPM Beats Per Minute (údery za minútu)

BSF Band Stop Filter (filter pásmová zádrž)

COCO Common Objects in Context

CNN Convolutional Neural Network (konvolučná neurónová sieť)

CUDA Compute Unified Device Architecture

DAW Digital Audio Workstation

DCO Digitally Controlled Oscillator (digitálne riadený oscilátor)

FIFO First In, First Out

FPS Frames Per Second (snímky za minútu)

GUI Graphical User Interface (grafické používateľské rozhranie)

HPF High Pass Filter (hornopriepustný filter)

HRTF Head-related Transfer Function (prenosová funkcia hlavy)

IOU Intersection over Union (prienik ku zjednoteniu)

LFO Low Frequency Oscillator (nízkofrekvenčný oscilátor)

LPF Low Pass Filter (dolnopriepustný filter)

MIDI Musical Instrument Digital Interface

MOT Multiple Object Tracking (sledovanie viacerých objektov)

MSP Max Signal Processing

NMS Non Max Suppression (potlačenie nemaximálnych hodnôt)

OSC Open Sound Control

PWM Pulse Width Modulation (modulácia striedy pulzného priebehu)

SORT Simple Realtime Online Tracking

UGen Unit Generator

VCO Voltage Controlled Oscillator (napätím riadený oscilátor)

YOLO You Only Look Once

Zoznam príloh

A Dokumentácia zvukových modulov

Názov	Popis	Parametre	
Beeper	Jednoduché sínusové pípanie	`freq`	Frekvencia tónu [Hz]
		`tempo`	Tempo pípania [BPM]
Duster	Nepravidelné pásmovo ohraničené impulzy	`freq`	Stredná frekvencia BPF [Hz]
		`tempo`	Priemerné tempo [BPM]
Droner	„Drone“ zložený z 5 pulzných oscilátorov s PWM	`freq`	Základná frekvencia oscilátorov [Hz]
		`movement`	Ovláda frekvenciu LFO aj hĺbku modulácie $\langle 0, 1 \rangle$
		`detune`	Miera vzájomného rozladenia oscilátorov [−]
		`f_cutoff`	f_m LPF [Hz]
		`f_resonance`	Parameter úmerný Q filtru $\langle 0, 1 \rangle$
Operator	Trojoperátorový FM syntetizátor so sériovým zapojením	`freq`	Nosná frekvencia [Hz]
		`tempo`	Tempo LFO s funkciou amplitúdovej obálky [BPM]
		`fm_ratio1`	Pomer mod. frekvencie 1. operátoru voči nosnej [−]
		`fm_ratio2`	Pomer mod. frekvencie 2. operátoru voči nosnej [−]
		`fm_depth1`	Hĺbka modulácie 1. operátoru [Hz]
		`fm_depth2`	Hĺbka modulácie 2. operátoru [Hz]

		`env_curve1`	Krivosť obálky 1. operátoru [−]
		`env_curve2`	Krivosť obálky 2. operátoru [−]
Pulsar	Pulsarový syntetizátor – pulsaret je niekoľko periód funkcie sin váhovaných oknom	`freq`	Základná frekvencia pulsaru [Hz]
		`formant_freq`	Formantová frekvencia pulsaru [Hz]
		`sine_cycles`	Počet periód v pulsarete [−]
		`window_curve`	Krivosť okna (umocnenie pílového priebehu) [−]
Drummer	Bicí automat s využitím subtraktívnej syntézy	`tempo`	Tempo [BPM]
		`density`	Hustota rytmu (pravdepodobnosť) $\langle 0, 1 \rangle$
		`vol_kick`	Hlasitosť basového bubna $\langle 0, 1 \rangle$
		`vol_snare`	Hlasitosť malého bubna $\langle 0, 1 \rangle$
		`vol_hat`	Hlasitosť hi-hat $\langle 0, 1 \rangle$
		`vol_clap`	Hlasitosť tlesknutia $\langle 0, 1 \rangle$
		`f_cutoff`	f_m LPF [Hz]
		`f_resonance`	Parameter úmerný Q filtru $\langle 0, 1 \rangle$
Player	Jednoduchý sampler	`buffer_id`	ID Bufferu so samplom [−]
		`playback_rate`	Rýchlosť prehrávania [−]

Grainer	Granulárny sampler	`buffer_id`	ID Bufferu so samplom [−]
		`grain_duration`	Dĺžka zrna [ms]
		`scan`	Rýchlosť posúvania pozície v sampli [−]
		`randomize`	Náhodná odchýlka v trvaní zrna, pozícii a smere prehrávania $\langle 0, 1 \rangle$
		`density`	Hustota zŕn $\langle 0, 1 \rangle$

B Identifikátory tried

Objekty COCO

0: person
1: bicycle
2: car
3: motorcycle
4: airplane
5: bus
6: train
7: truck
8: boat
9: traffic light
10: fire hydrant
11: stop sign
12: parking meter
13: bench
14: bird
15: cat
16: dog
17: horse
18: sheep
19: cow
20: elephant
21: bear
22: zebra
23: giraffe
24: backpack
25: umbrella
26: handbag
27: tie
28: suitcase
29: frisbee
30: skis
31: snowboard
32: sports ball
33: kite
34: baseball bat
35: baseball glove
36: skateboard
37: surfboard
38: tennis racket
39: bottle
40: wine glass
41: cup
42: fork
43: knife
44: spoon
45: bowl
46: banana
47: apple
48: sandwich
49: orange
50: broccoli
51: carrot
52: hot dog
53: pizza
54: donut
55: cake
56: chair
57: couch
58: potted plant
59: bed
60: dining table
61: toilet
62: tv
63: laptop
64: mouse
65: remote
66: keyboard
67: cell phone
68: microwave
69: oven
70: toaster
71: sink
72: refrigerator
73: book
74: clock
75: vase
76: scissors
77: teddy bear
78: hair drier
79: toothbrush

Pohlavie

0: Man
1: Woman

Emócia

0: neutral
1: happy
2: surprise
3: fear
4: sad
5: angry
6: disgust

C Obsah elektronickej prílohy

Elektronická príloha obsahuje zdrojový kód vytvorenej aplikácie. Kvôli množstvu požiadavok na externé moduly a veľký dátový objem modulov aj zvukovej databanky je poskytnutá skompilovaná spustiteľná verzia, ktorú je potrebné stiahnuť z odkazu v súbore `install.txt`.

- `/` — koreňový adresár priloženého archívu
  - `src` — zdrojový kód
    - `audio` — zložka zvukovej databanky
      - `database_builder.py` — skript pre zostavenie databanky
      - `mp3_to_wav.py`
      - `normalize_wav.py`
    - `patches` — zložka s patchmi uzlového editoru
      - `...`
    - `obj_sonify.py` — hlavný skript aplikácie
    - `synths.py` — definície syntetizátorov
    - `tracker.py` — sledovač objektov
    - `mapping.py` — abstrakcie pre prepojenie syntetizátorov s objektmi
    - `face_tracking.py` — funkcie pre analýzu tváre
    - `yolo.py` — pomocné funkcie pre prácu s modelom YOLO
    - `nodes.py` — uzly uzlového editoru
    - `gui.py` — grafické používateľské rozhranie
    - `text_completer.py` — QtWidget pre automatické doplňovanie textu
    - `hotkey_functions.py` — funkcie menu uzlového editoru
    - `hotkeys.json` — položky menu uzlového editoru
    - `requirements.txt` — externé moduly pre inštaláciu pomocou `pip`
  - `install.txt` — návod pre stiahnutie skompilovanej verzie