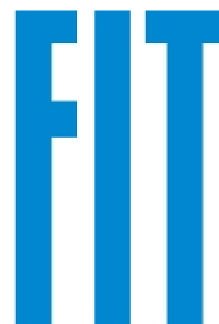
Aplikace *Anonymizér*

Technická zpráva - FIT - VG20102015006 – 2015 – 05

Ing. Filip Orság, Ph.D.



Fakulta informačních technologií, Vysoké učení technické v Brně

07. října 2015

Abstrakt

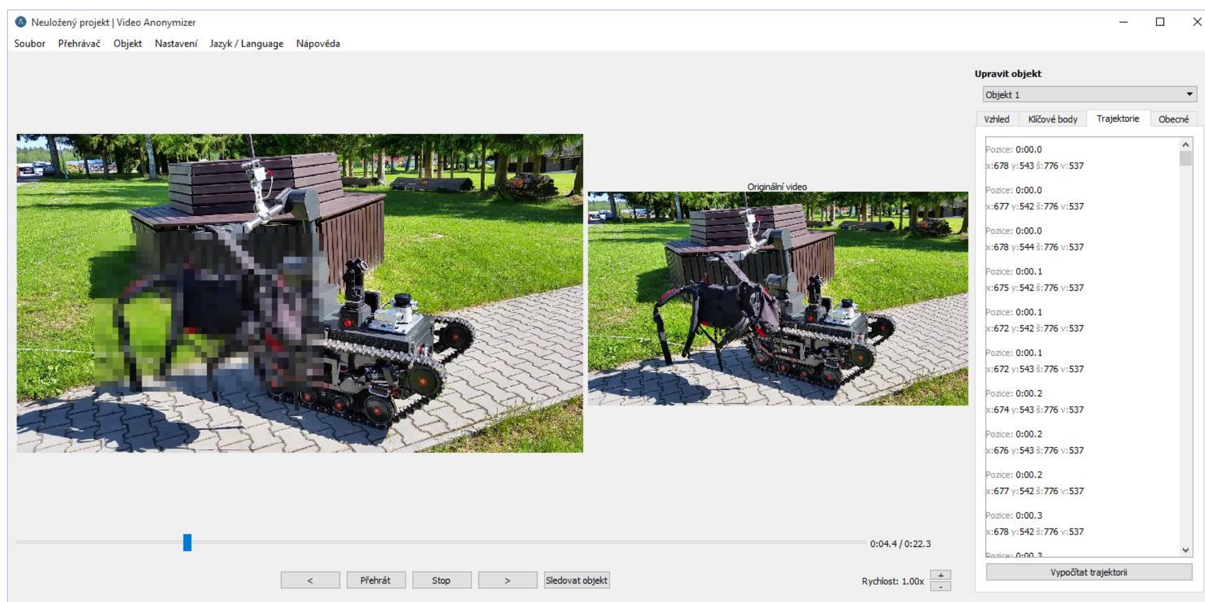
Tato aplikace umožňuje sledovat vybrané objekty ve videu a upravovat jejich vzhled. Pro práci s videem byla použita knihovna FFmpeg. Objekty je možné sledovat pomocí zabudovaných algoritmů sledování objektů, lze upravovat jejich trajektorií a způsob anonymizace (rozmazání, zabarvení). GUI je vícejazyčné, vytvořené pomocí knihovny Qt.

Obsah

1 Uživatelská příručka ... 1
2 Princip sledování maskovaného objektu ... 1
2.1 Sledování částicovými filtry ... 1
2.2 Nelineární rekurzivní Bayesovské sledování ... 2
2.3 Bayesovský bootstrap filtr ... 3
3 Implementace ... 4
3.1.1 Knihovna Qt ... 4
3.1.2 Knihovna OpenCV ... 8
3.1.3 CUDA ... 8
3.1.4 Cereal ... 12
3.1.5 FFmpeg ... 12
4 Uživatelská příručka ... 13
5 Reference ... 18

1 Uživatelská příručka

Anonymizér je samostatná, nezávislá aplikace, která umožňuje anonymizaci videa – tedy zamaskování vybraných objektů ve videu z důvodu zachování anonymity obrazovou deformací nebo úplným vybraného objektu.



2 Princip sledování maskovaného objektu

Pro sledování maskovaného objektu se využívají algoritmy sledování objektů ve videu. Těchto algoritmů je velké množství a aplikace Anonymizér umožňuje snadno využít jakýkoliv z nich. Pro demonstrování principu a základní funkcionalitu byl zvolen základní částicový filtr a jako alternativa zdokonalená verze, která je také využívána ve funkčním vzorku sledovacího systému z tohoto projektu.

2.1 Sledování částicovými filtry

V několika předcházejících letech se stal velmi populárním přístup sledování objektu založený na různých modifikacích a variantách rekurzivního Bayesovského filtrování, nazývaného také částicové filtry (*particle filters*), někdy také označované jako sekvenční metody Monte Carlo (SMC). Částicové filtry poskytují robustní základ pro sledování objektů, který se neomezuje pouze na lineární dynamické systémy a Gaussovo rozložení šumu. Určitou formu částicového filtru použili poprvé na sledování objektů ve videu Isard a Blake a nazvali ho *Condensation* [5], což je zkratka z *Conditional Density Propagation* a v komunitě zaměřené na počítačové vidění je tento algoritmus typickým představitelem částicových filtrů. Tento algoritmus byl použit na sledování objektů v obrazech upravených na konturovou reprezentaci scény [6][7].

Konturové modely jsou relativně robustní vůči změnám osvětlení, ale mohou být výpočetně náročné a náchylné k chybám způsobeným členitostí pozadí scény [8]. Využití informace o barvě je další z mnoha velmi často využívaných technik. Mnohdy je využívána informace ve formě histogramu [1][9]. Částicové filtry využívající barevné informace v podobě histogramů jsou při sledování objektu velmi odolné vůči rigiditě pohybujících se objektů, částečnému zakrytí objektu a šumu. Nicméně tento způsob popisu je omezen tím, že ignoruje prostorovou polohu bodů, což znemožňuje rozeznání objektů

s podobným barevným rozložením. Tento problém je pak znásoben v případě nutnosti práce s monochromatickým videem [10].

Jiným jednoduchým a velmi často používaným přístupem k popisu pohybujícího se objektu je využití vzoru objektu (*template*) – tedy malé části snímku, který přímo reprezentuje vzhled daného objektu. Výhodou tohoto přístupu je, že tento popis v sobě nese informaci o barvě i prostorovém vzhledu. Tento přístup je vhodný pro barevné i monochromatické modely a jeho výpočet bývá velmi rychlý. Na druhou stranu může dojít (a dochází) k tomu, že vybraný vzor se stane nereprezentativní pro sledovaný objekt především z důvodu změny jeho vzhledu v důsledku pohybu (změna měřítka, rotace) a šumu. Aby se předešlo tomuto problému, využívá se například techniky pomalé aktualizace vzoru v průběhu času [11][12][13]. V průběhu aktualizace však hrozí nebezpečí, že vzor postupně začne reprezentovat zcela jiný objekt v důsledku nesprávného určení polohy. Velká pozornost musí být také věnována i (částečnému) zakrytí objektu, které může také vést k selhání algoritmu.

2.2 Nelineární rekurzivní Bayesovské sledování

Nelineární stochastický systém v doméně diskrétního času lze popsat následujícími způsobem. Rovnice

$$x_{n+1} = f(x_n, d_n), \tag{1}$$

se nazývá stavovou (nebo přechodovou) rovnicí a x_{n+1} popisuje stav systému v čase $n+1$, x_n popisuje stav systému v čase n a d_n lze považovat za náhodné sekvence v podobě bílého šumu s neznámou statistikou v diskrétní časové doméně. Následující rovnice se nazývá rovnice měření (nebo pozorování), kde pozorování v čase n je označeno jako y_n, x_n je stav systému ve stejném čase a v_n je, podobně jako u předchozí rovnice, šum:

$$y_n = h(x_n, v_n). \tag{2}$$

Za povšimnutí stojí fakt, že aktuální stav systému závisí pouze na předchozím stavu, což je situace, kterou lze popsat Markovskými modely, a označujeme ji jako Markovský proces. Za určitých okolností lze tyto rovnice redukovat na lineární Markovský model. Cílem sledování objektu (filtrování) je tedy určení stavu x_n, který je skrytý, na základě předchozích pozorování $y_{0:n} = (y_0, y_1, \dots, y_n)$.

Pokud se na celý problém podíváme z Bayesovského úhlu pohledu, pak je problém sledování objektu vytvoření posteriorní hustoty pravděpodobnosti $p(x_n|y_{0:n})$ aktuálního stavu x_n, za předpokladu, že je dána posloupnost pozorování od počátečního stavu 0 do času n a za dané počáteční hustoty $p(x_0)$, přechodové hustoty $p(x_n|x_{n-1})$ a pravděpodobnosti $p(y_n|x_n)$. V principu lze tuto posteriorní hustotu vypočítat rekurzivně ve dvou krocích výpočtem Chapman-Kolmogorovovy rovnice predikčního kroku, která slouží k určení předchozí hustoty $p(x_n|y_{0:n-1})$:

$$p(x_n|y_{0:n-1}) = \int p(x_n|x_{n-1})\, p(x_{n-1}|y_{0:n-1}) dx_{n-1} \tag{3}$$

a rovnice aktualizační, která vychází z úměrnosti:

$$p(x_n|y_{0:n}) \propto p(y_n|x_n) p(x_n|y_{0:n-1}). \tag{4}$$

Rekurzivní vztah obou rovnic nelze obecně určit analyticky kvůli náročnosti, téměř nemožnosti, vyhodnocení obvykle komplikovaného integrálu (kromě speciálních případů, kdy jde o lineární a Gaussovský stavový prostorový model).

Pomocí systémového modelu $p(x_n|x_{n-1})$ je odhadnuta apriorní funkce hustoty pravděpodobnosti (*pdf*) na základě předcházejícího stavu systému $p(x_n|y_{0:n-1})$ v čase $n-1$. S dalším měřením (v čase n) se na základě Bayesova pravidla vypočítá aposteriorní *pdf* (fáze aktualizace) podle rovnice

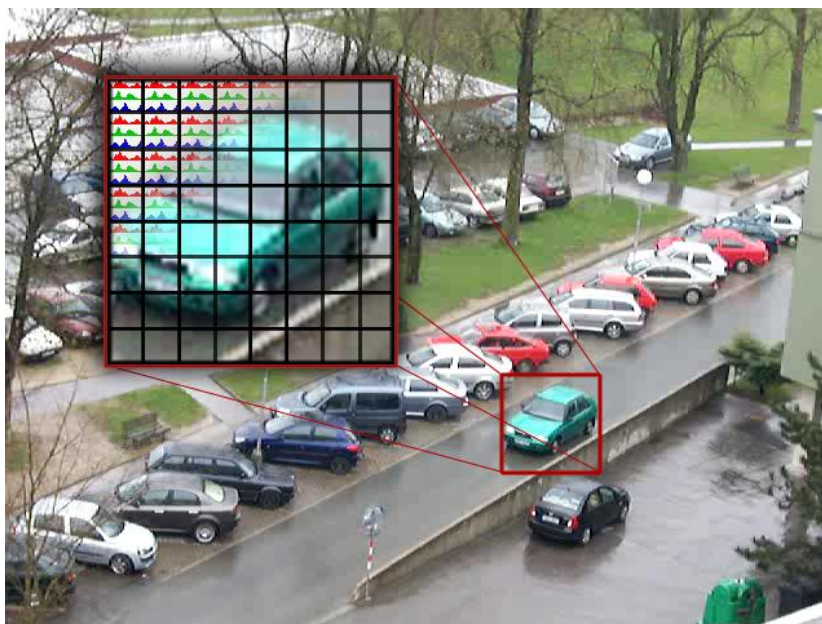
$$p(x_n|y_{0:n}) = \frac{p(y_n|x_n)p(x_n|y_{0:n-1})}{p(y_n|y_{0:n-1})}, \tag{5}$$

kde pravděpodobnost $p(y_n|x_n)$ zastupuje model měření a představuje sadu naměřených hodnot v čase n, filtr modifikuje v této fázi nepřesnou apriorní *pdf* měřením y_n, a $p(y_n|y_{0:n-1})$ je normalizační konstanta.

Rekurentní vztahy mezi fázemi vytváří optimální Bayesovské řešení. Abychom se vyhnuli integraci v rekurzivní Bayesovské rovnici, je posteriorní hustota aproximována množinou vzorků, které mají přiřazeny pravděpodobnostní váhy. Nicméně, posteriorní hustota může být nestandardní, a proto může být značně obtížné vygenerovat vzorky z posteriorní hustoty přímo. Abychom tomuto problému předešli, používá se další, tzv. navrhovaná hustota, z níž lze snadněji generovat vzorky, které jsou použity namísto hodnoty pravděpodobnosti posteriorní hustoty. Tento postup je označován jako *Importance Sampling* [1]. Aposteriorní *pdf* tedy vyjádříme například náhodně generovanými váženými vzorky – *částicemi*. S rostoucím počtem částic se přibližujeme skutečnému aposteriornímu *pdf*. Množinu částic vyjádříme jako:

$$\left\{\forall i = 1, \dots, N: X_{n-1}^{(i)}, w_{n-1}^{(i)}\right\} \tag{6}$$

kde $w_{n-1}^{(i)}$ představuje váhu konkrétní částice. Tuto množinu částic poté používáme při výpočtu v jednotlivých krocích algoritmu. Na následujícím obrázku (obrázek 2.4) je znázorněna ukázka z implementace tohoto algoritmu s mřížkou histogramů pro jednu částici.



Obrázek 2.1: Sledování objektu za použití částicových filtrů, ukázka pole histogramů barevných složek použitých jako příznaky.

2.3 Bayesovský bootstrap filtr

Bayesovský bootstrap částicový filtr (převzorkovávací filtr) je variantou filtru převzorkování na základě významnosti (*Sequential Importance Resampling* – *SIR*)[3], kde je hustota přechodu využita jako

navrhovaná hustota, a kde je krok převzorkování prováděn při každé iteraci. Obecný algoritmus Bayesovského převzorkovávacího filtru je složen z následujících kroků:

1. Predikce: každá částice $\check{\chi}_{n-1}^{i}, i \in \{1, \dots, N\}$ je využita v modelu systému k výpočtu diskrétní aproximace hustoty pravděpodobnosti

$$\chi_n^i = f\left(\check{\chi}_{n-1}^i, d_{n-1}\right). \tag{7}$$

2. Aktualizace: je vyhodnocena pravděpodobnost všech pozorování, každé částici je přiřazena nenormalizovaná váha významnosti:

$$\varpi_n^i = p\left(y_n \middle| \chi_n^i\right). \tag{8}$$

3. Normalizace vah:

$$\omega_n^i = \frac{\varpi_n^i}{\sum_1^N \varpi_n^i}. \tag{9}$$

4. Převzorkování: ze stávající množiny částic $\{\chi_n^1, \dots, \chi_n^N\}$ je vybráno N částic $\check{\chi}_n^i$ na základě jejich významnosti, tedy na základě jejich vah ω_n^i.

Inicializace filtru vyžaduje N částic, které jsou vybrány ze známé hustoty pravděpodobnosti, tyto částice pokračují přímo krokem 2, jsou tedy aktualizovány přímo. Jako jednotlivé měření může být odhadovaný stav vypočítán jako střední hodnota nejvýznamnějších částic nebo může být reprezentován přímo nejvýznamnější částicí (tj. jako *Maximum a-posteriori* – MAP).

3 Implementace

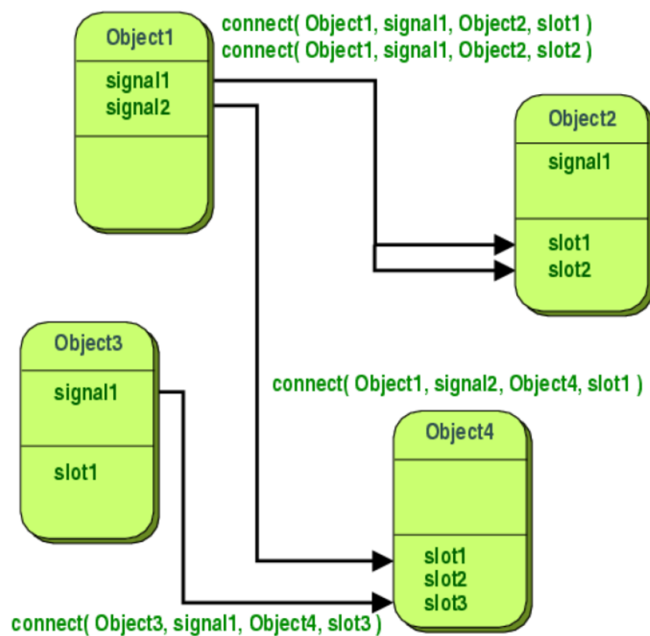
Celá aplikace je naprogramována v jazyce C++. Pro svoji činnost využívá knihovny OpenCV, FFmpeg a cereal. Jako prostředek pro tvorbu GUI byl zvolen framework Qt. Jeden z algoritmů sledování využívá programování grafické karty prostřednictvím CUDA. V následujících kapitolách jsou stručně představeny jednotlivé knihovny.

3.1.1 Knihovna Qt

Knihovna, respektive framework, Qt je multiplatformní knihovna (lze tedy docílit spuštění aplikace vyvinuté v Qt na různých platformách) a má podporu pro jazyk C++, ale existují i verze pro jazyky Python, Perl, C#, Java a mnoho dalších. Tato knihovna je vhodná na tvorbu programů s přívětivým grafickým uživatelským rozhraním. Obsahuje také nástroje na zpracování souborů XML, multimédií, podporu práce s počítačovou sítí a další. V současné době je využívána jako základ pro velké množství známých programů (například internetový prohlížeč Opera, komunikátor Skype, VirtualBox a jiné). Je na ní postavené i unixové prostředí KDE.

Základní odlišností knihovny Qt od jiných knihoven je koncepce a využívání signálů a slotů, které se využívají pro komunikaci mezi objekty. Qt je objektově orientovaná knihovna a všechny elementy jsou tedy objekty. Signálem a slotem se označuje způsob komunikace, který nahrazuje funkce zpětného volání (*callback function*). Funkce zpětného volání měla dvě základní nevýhody – při volání nebyla nutně prováděna typová kontrola a funkce, která byla volána, musela znát ukazatel na volající funkci. Tyto problémy se odstranily právě zavedením signálů a slotů.

Signál je speciální funkce (metoda) bez návratové hodnoty (tedy `void`), která má pouze prototyp, nemá tedy vlastní funkční tělo. Signály jsou emitovány objekty při změně jejich stavu (například stisk tlačítka). Objekty samotné neví o tom, že signál, který vysílají, je přijímán jinými objekty. Slot je pak speciální funkce, která je volaná jako odpověď na přijatý signál, ale lze ji použít i jako jakoukoliv jinou běžnou metodu dané třídy. Qt nabízí mnoho předdefinovaných signálů a slotů a existuje samozřejmě i možnost implementace vlastních a jejich přidružení k různým typům událostí. Třídy, které signály a sloty chtějí využít, musí být potomky základní třídy `QObject`, v níž je definována metoda `connect()`, která zajišťuje vzájemné propojení signálů a slotů a má čtyři parametry – objekt, od něhož očekáváme signál, emitovaný signál, objekt reagující na daný signál a nakonec slot tohoto objektu. Grafické znázornění této situace je na obrázku 3.1.



Obrázek 3.1: Schéma propojení objektů prostřednictvím signálů a slotů.

Knihovna Qt poskytuje množinu modulů, které nabízejí potřebnou funkcionalitu pro vývoj grafických aplikací a mnohem více. Výčet jednotlivých modulů je poměrně dlouhý, zmíníme tedy alespoň několik, pro nás nejdůležitějších modulů, které jsou k dispozici:

- `QtCore` – základní funkce, které nemají vazbu na GUI,
- `QtGui` – většina tříd, které mají vztah ke GUI,
- `QtNetwork` – obsahuje třídy pro komunikaci prostřednictvím UDP a TCP včetně potřebných klientů a serverů,
- `QtXml` – poskytuje prostředky pro práci s XML soubory a s rozhraními SAX a DOM.

Konkrétní třídy, které nás budou při vývoji aplikace zajímat, jsou především graficky orientované třídy. Základem pro vývoj aplikace je třída `QMainWindow`, která se využívá na vytvoření hlavního okna aplikace s možností vložení menu (třída `QMenuBar`), lišty nástrojů (třída `QToolBar`) a stavovou řádku (třída `QStatusBar`). Kromě toho lze do zbývajícího prostoru umístit libovolné grafické prvky, kterým se v Qt říká *widgety* (například třídy textu `QLabel`, tlačítka `QPushButton`, atd.). Widget, jakožto elementární třída, je definován jako třída `QWidget` a slouží jako vrstva mezi prvkem a oknem a je od ní odvozeno mnoho dalších tříd (například právě `QMainWindow`, dialogové okno `QDialog`, další prvky GUI, atd.).

Obrázek 3.2: Výřez schématu propojení objektů prostřednictvím signálů a slotů.

Mezi často používané prvky GUI patří například třída pro editaci řádky `QLineEdit` nebo textu `QTextEdit`. Třída `QDialog` poskytuje prostředky pro tvorbu dialogových oken, u nichž není možné měnit uživatelsky velikost na rozdíl od oken vytvořených jako přímý potomek tříd `QWidget` nebo `QMainWindow`. Potomek třídy `QDialog` je například třída `QMessageBox`, která umožní zobrazit krátkou textovou zprávu, nebo `QFileDialog`, která umožní vybrat a otevřít soubor.

Třídy, které umožní modifikovat a přesně stanovovat rozložení dalších objektů v rámci okna jsou třídy `QHLayout` a `QVLayout`. Díky nim je možné zaručit určitou podobnost výsledné aplikace na různých

platformách. Tyto třídy je možné přirozeně vnořovat do sebe, čímž můžeme docílit složitějších kompozic ovládacích prvků a jejich vzájemnou interakci.

Mezi základní grafické ovládací prvky patří například statický text (třída QLabel), tlačítka (třídy QPushButton nebo QToolButton), nástrojová lišta (třída QToolBar), hlavní menu (třída QMenuBar), číselník s možností změny celočíselné hodnoty (třída QSpinBox) často používaná v kombinaci s posuvníkem (třída QSlider nebo QScrollBar).

Třídy v Qt jsou vytvářeny na principech objektového přístupu, tedy děděním vlastností tříd rodičovských. Malá část stromu tříd a jejich odvození je na následujícím obrázku. Nejsou zde znázorněny všechny třídy, které knihovna poskytuje, ale pouze malý výběr tříd odvozených od třídy QDialog, QFrame, QAbstractButton, QAbstractSlider a QAbstractSpinBox.

V každém graficky orientovaném programu je třeba vytvořit instanci třídy QApplication, které se případně předávají argumenty z příkazové řádky. Na konci programu je vhodné vracet výsledek činnosti aplikace zprostředkovaný metodou exec(). V hlavní funkci main() je vhodnější vytvářet objekty na zásobníku, tedy jako lokální proměnné. Následující úsek zdrojového kódu ukazuje jednoduchý programu typu *„Hello World“*:

```
#include <QApplication>
#include <QLabel>

int main(int argc, char** argv)
{
  QApplication app(argc, argv);

  QLabel win("Hello World!");
  win.show();

  return app.exec();
}
```

V uvedené ukázce je vidět, že hlavním oknem programu může být jakákoliv třída odvozená od třídy QWidget nebo QWidget samotný. Obvykle pouze vytvoříme instance třídy hlavního okna a zavoláme její metodu show(), která okno zobrazí a zajistí chod aplikace do chvíle, kdy je okno uzavřeno. Následující zdrojový kód je ukázkou propojení pomocí signálu a slotu:

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char** argv)
{
  QApplication app(argc, argv);

  QPushButton btn("&Quit");
  QObject::connect(&btn, SIGNAL(clicked()), app, SLOT(quit()));
  btn.show();

  return app.exec();
}
```

Zde je tlačítko (proměnná btn) propojeno s hlavním oknem aplikace (proměnná app, která je ukazatelem na unikátní instanci třídy QApplication). K propojení tlačítka a aplikace slouží metoda connect() zděděná z třídy QObject. Tato metoda je přetížená, takže se může volat s různými argumenty, ale nejčastěji se používá varianta uvedená v ukázce výše, tedy čtyři parametry v tomto pořadí: ukazatel na instanci objektu Qt (v našem případě QPushButton) od kterého očekáváme signál, následuje signál, který spustí daný slot (de facto jde o funkci), dalším je ukazatel na instanci objektu, který dostane informaci o vygenerovaném signálu (proměnná app) a posledním je slot, který je aktivován v případě, že je emitován signál (tedy je zavolána příslušná funkce – quit() v našem případě).

Další informace jsou dostupné na oficiálních stránkách v podobě dokumentace frameworku Qt.

3.1.2 Knihovna OpenCV

OpenCV (*Open Source Computer Vision library*) je knihovna funkcí primárně zaměřených na výpočty počítačového vidění v reálném čase. Knihovna byla vytvořena oficiálně v roce 1999 iniciativou firmy Intel Research, aby posunula hranice aplikací zaměřených na náročné výpočty, přičemž část projektu byla zaměřena například na výpočet zobrazení 3D scény metodou sledování paprsku (*ray tracing*) v reálném čase a zobrazování 3D scén. Cíle projektu OpenCV na jeho začátku byly popsány jako

- umožnění pokroku ve výzkumu počítačového vidění v otevřeném projektu (aby bylo zamezeno „znovuobjevování kola“),
- poskytnutí společné programátorské infrastruktury pro vývojáře, na níž by bylo možné stavět aplikace počítačového vidění s důrazem na čitelnost a přenositelnost,
- umožněné

Knihovna je nezávislá na platformě a vývoj se zaměřil na zpracování obrazu v reálném čase s tím, že došlo k optimalizaci mnoha funkcí pro technologii CUDA, a pokud systém disponuje i knihovnou *Intel's Integrated Performance Primitives* (Intel IPP), je tato využita k dalšímu zvýšení výkonu. Je tedy zřejmé, že využití této knihovny je úzce spjato s firmami Intel a nVidia a s jejich produkty, což je také jedním z důvodů, proč jsme se při výběru hardware přikláněli spíše k těmto výrobcům (zrychlení výsledného kódu je znatelné). Aktuálně je ve verzi 2.3 s tím, že verze 2 byla převratná v mnoha ohledech, především pak přechodem na C++, což učinilo z OpenCV objektově orientovanou knihovnu. Taktéž byly funkce rozděleny na logické celky a není tedy nutné s každou aplikací využívající OpenCV dodávat i celou knihovnu, ale pouze relevantní části.

Z hlediska programátorského přináší OpenCV několik základních konceptů usnadňujících práci – jmenný prostor `cv`, automatická správa paměti, automatické alokování paměti výstupních parametrů, saturační aritmetika, pevně dané typy pixelů, omezené používání vzorů (*template*), základní vstupně výstupní pole reprezentující téměř libovolný typ vektorů a matic dostupných v OpenCV, ošetření chyb systémem výjimek a v neposlední řadě využití vláken a možnost opakovaného spouštění funkcí z libovolného vlákna.

3.1.3 CUDA

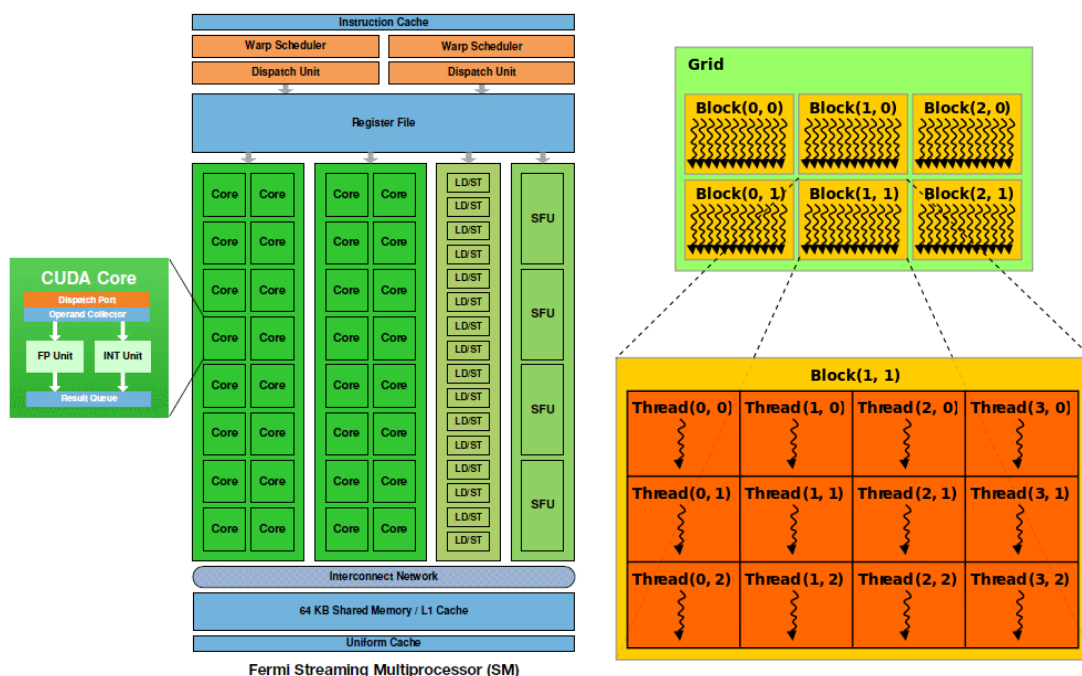
CUDA (*Compute Unified Device Architecture*) je hardwarově-softwarová architektura umožňující, podobně jako OpenCL, na GPU spouštět programy napsané v jazycích C/C++, FORTRAN nebo programy postavené na technologiích OpenCL, DirectCompute a jiných. Tato architektura je však dostupná pouze na grafických kartách společnosti nVidia, která ji vyvinula.

CUDA byla představena poprvé v roce 2006, následovalo uvolnění SDK (*Software Development Kit*) pro programátory profesionálních karet Tesla založených na architektuře jádra G80. Dále byl vývoj této technologie rychlý s různými změnami a rozšířením na běžné grafické karty, což umožnila masivní rozšíření využití této technologie s tím, že aktuální je verze 4 (rok 2011) a připravuje se verze 5.

Architektura čipu GPU firmy nVidia, je založena na velkém množství relativně jednoduchých skalárních procesorů, což je rozdíl od klasického CPU, který bývá velmi komplexní. Čipy této firmy se liší například i od architektury konkurenční firmy AMD, jejíž multiprocesory jsou tvořeny komplexnějšími VLIW – *Very Long Instruction Word* – jednotkami, které jsou organizovány do celků nazývaných streaming multiprocesory. Vzhledem k tomu, že se jedná o SIMT (*Single Instruction Multiple Threads*) architekturu, je řízení jednotek a plánování instrukcí vcelku jednoduché a spolu s malou vyrovnávací

pamětí zabírá malou část plochy čipu, což má však za následek právě omezení v predikci skoků a časté výpadky cache. Poslední významnou částí, která je rozměrově velice podobná CPU, je řadič paměti, který zabírá přibližně stejnou plochu.

Výše zmíněný streaming multiprocesor se obecně skládá z několika (v současnosti až z 32) procesorů, pole registrů, sdílené paměti, několika jednotek pro ukládání a čtení a speciální funkční jednotky pro výpočet složitějších funkcí jako je sinus, kosinus nebo logaritmus (viz následující obrázek). Výpočetní možnosti zařízení jsou dány množinou instrukcí, které jsou podporovány, konkrétní vlastnosti však pro nás v tomto případě nejsou rozhodující.



Obrázek 3.3: Schéma multiprocesoru v grafickém čipu firmy nVidia Fermi (obrázek vlevo) a uspořádání vláken a bloků (obrázek vpravo) [14].

Z hlediska programátorského je způsob programování CUDA podobný OpenCL s tím rozdílem, že je poněkud jednodušší v inicializační fázi, neboť nemusíme řešit tu skutečnost, že nevíme, s jakým hardware a s jakými prostředky přijdeme do kontaktu, což snižuje složitost programátorského modelu. Tento fakt a skutečnost, že CUDA byla představena dříve než OpenCL, je také důvodem pro větší rozšířenost a oblíbenost využití CUDA pro různé výpočetně náročné úkoly.

Vnitřní uspořádání vláken a bloků je opět podobné OpenCL. Aplikace využívající CUDA je složena z částí, které jsou prováděny procesorem, a z částí prováděných na GPU. Jádra aplikace běžící na GPU jsou spouštěny hostem zavoláním daného jádra (*kernel*), tedy funkce prováděné každým spuštěným vláknem (*thread*). Vlákna jsou organizována do 1, 2, nebo 3rozměrných bloků, kde vlákna ve stejném bloku mohou sdílet data a lze synchronizovat jejich běh. Počet vláken na jeden blok je závislý na výpočetních možnostech konkrétního zařízení. Každé vlákno je v rámci bloku identifikováno unikátním indexem (proměnnou `threadIdx`), viz také předcházející obrázek.

Bloky jsou pak dále organizovány do 1, 2 nebo 3rozměrné mřížky. Blok lze v rámci mřížky identifikovat unikátním indexem přístupným ve spuštěném kernelu přes zabudovanou proměnou `blockIdx`. Každý blok vláken musí být schopen pracovat nezávisle na ostatních, aby byla umožněna škálovatelnost systému (GPU s více jádry spustí více bloků paralelně, v porovnání s GPU, které obsahuje méně jader, kde budou bloky spouštěny sériově). Speciálním pojmem je pak tzv. *warp*, což je množina vláken

zpracovávaných v jednom okamžiku. Jeho velikost je závislá na počtu výpočetních jednotek s tím, že konkrétní počet a organizace spuštěných vláken v jednom bloku, stejně jako i počet a organizace bloků v mřížce se určuje při spouštění jádra.

Průběh inicializace a spuštění výpočtu je ve srovnání s OpenCL snadnější, neboť zde odpadá nutnost zjišťování vlastností zařízení. Přítomnost zařízení a jeho parametry je možné zjistit, pokud chceme optimalizovat běh aplikace, ale lze to nechat i na ovladačích, které jsou schopny do určité míry za nás rozhodnout.

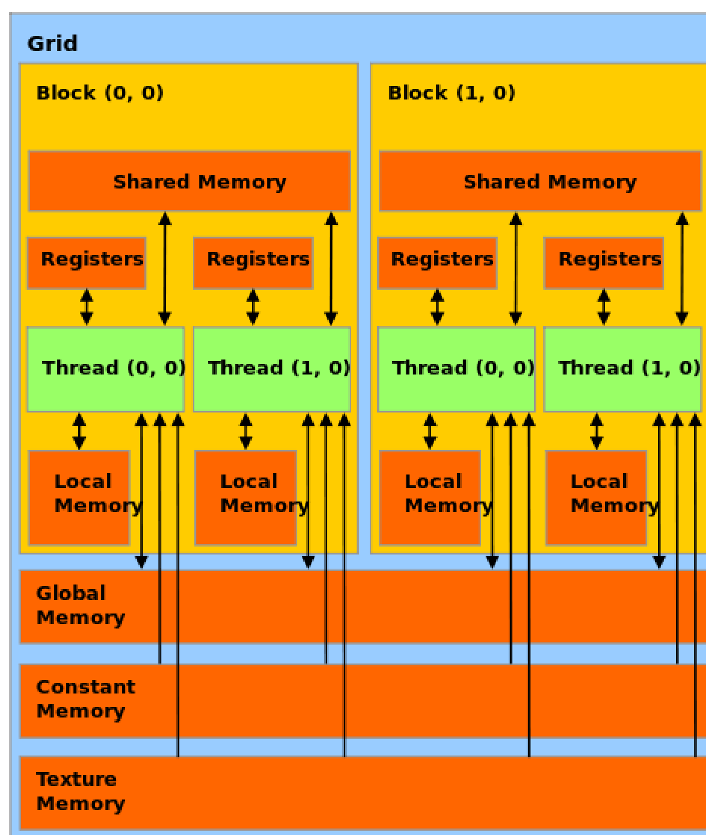
Následující ukázka kódu je převzatá z oficiální dokumentace a je mírně upravená [14]. Kód se dělí na dvě části – kód pro GPU a pro CPU. Kód pro GPU je přeložen překladačem pro CUDA a poté spouštěn na jádrech procesoru grafické karty. Kód pro CPU je inicializační a jeho hlavním úkolem v tomto konkrétním příkladu je vyhrazení paměti na GPU, přesun hodnot z paměti procesoru do paměti grafické karty, spuštění výpočtu na GPU, čekání na výsledek (synchronizace) a následný přesun výsledků výpočtů z paměti grafické karty do paměti CPU.

```
// Kód pro GPU
__global__ void addVectors(float* A, float* B, float* C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N) C[i] = A[i] + B[i];
}

// Kód pro CPU
int main()
{
    int N = 1024;
    size_t size = N*sizeof(float);

// Alokace vstupních vektorů v paměti CPU
    float* h_A = (float*)malloc(size);
    float* h_B = (float*)malloc(size);

// Inicializace vstupních vektorů
    for(int i = 0; i < N; i++) { h_A[i] = N - i; h_B[i] = i; }

// Alokace paměti na zařízení
    float* d_A, d_B, d_C;
    cudaMalloc(&d_A, size);
    cudaMalloc(&d_B, size);
    cudaMalloc(&d_C, size);

// Přesun vektorů z hlavní paměti do paměti zařízení
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

// Spuštění výpočtu = volání jádra (funkce addVectors)
    int threadsPerBlock = 256;
    int blocksPerGrid = (N + threadsPerBlock - 1)/threadsPerBlock;
    addVectors<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);

// Přesun výsledků z paměti zařízení do hlavní paměti
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

// Uvolnění paměti na zařízení
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

// Uvolnění hlavní paměti
    free(h_A);
    free(h_B);
}
```

Pro optimalizaci výpočtů je velmi důležitým aspektem při používání GPU práce s pamětí. Paměťový model je vcelku komplexní a jeho znalost a správné využívání vede k žádoucímu výsledku, naopak jeho nerespektování končí většinou kódem, který je mnohdy pomalejší než jeho sériová varianta pro CPU.

Paměti jsou uspořádány v hierarchii s tím, že na grafické kartě samotné je až 6 druhů paměti, se kterými může programátor pracovat (viz obrázek 3.4):

- Registry: jsou umístěny v jednotlivých procesorech a jejich rozdělení mezi jednotlivé procesory plánuje překladač, každé vlákno má přístup pouze registrům svého procesoru, jde tedy o lokální, velmi rychlou paměť.
- Lokální paměť: je také blízko registrů a je použita v případě, že dojde k vyčerpání dostupných registrů. Tato paměť je také přístupná pouze jednomu vláknu, fyzicky je však umístěna v globální paměti, což z ní paradoxně činí pomalejší typ než je například sdílená paměť.
- Sdílená paměť: je jedinou pamětí (kromě registrů), která je umístěna přímo na čipu procesoru. Mohou k ní přistupovat všechna vlákna v daném bloku a přistupuje se k ní přes brány nazývané banky. Každá banka může zpřístupnit pouze jednu adresu v jednom taktu a v případě, že více procesorů požaduje přístup přes stejnou banku, dochází k výpadkům paměti v podobě čekání na vyřízení požadavku. Speciálním případem je situace, kdy všechna vlákna čtou ze stejné adresy a banka hodnotu zpřístupní v jednom taktu všem procesorům najednou šířením (*broadcast*).
- Globální paměť: je sdílená mezi všemi procesory a data nejsou ukládána do paměti cache.
- Paměť konstant: je paměť sdílená, určená pouze ke čtení, a je pro ni na čipu procesoru vyhrazena paměť cache první úrovně (L1). Podobně jako sdílená paměť umožňuje šíření dat.
- Paměť textur: je také sdílená paměť, je určena pouze ke čtení a na rozdíl od globální paměti disponuje cache. Je optimalizována pro dvourozměrnou prostorovou lokalitu, takže vlákna ve stejném warpu, která čtou texturu z prostorově blízkých souřadnic, dosahují vyššího výkonu.



Obrázek 3.4: Grafické znázornění hierarchie pamětí a vazby na procesor (vlákno) a blok [14].

Z přehledu je zřejmé, že při programování GPU je nutné dbát na to, jak se využívá paměť, kolik dat je potřeba a jak je organizovat, aby bylo dosaženo nejlepší možné optimalizace kódu. Jednotlivé paměti a jejich charakteristiku shrnuje následující tabulka:

Typ paměti	Umístění	Cache	Přístup	Viditelnost	Životnost
Registry	na čipu	ne	čtení i zápis	1 vlákno	po dobu života vlákna
Lokální paměť	externí	ne	čtení i zápis	1 vlákno	po dobu života vlákna
Sdílená paměť	na čipu	ne	čtení i zápis	všechna vlákna v bloku	po dobu existence bloku
Globální paměť	externí	ne	čtení i zápis	všechna vlákna a CPU	do uvolnění
Paměť konstant	externí	ano	čtení	všechna vlákna a CPU	do uvolnění
Paměť textur	externí	ano	čtení	všechna vlákna a CPU	do uvolnění

3.1.4 Cereal

Cereal je knihovna napsaná v jazyce C++, která usnadňuje serializaci. Cereal umí zpracovat libovolný datový typ a reversibilně je transformuje na jinou reprezentaci, například kompaktní binární kódy, XML, nebo JSON. Cereal byl navržen tak, aby byl rychlý, paměťově nenáročný a snadno rozšiřitelný. Nemá externí závislosti a je možné ho snadno přiložit k libovolnému dalšímu kódu nebo použít samostatně, více viz [15].

3.1.5 FFmpeg

FFmpwg je skupina programů a knihoven pro přehrávání a kompresi videa, využitelná a využívaná na Linuxu, MS Windows i Mac OS X. Jeho části jsou obsaženy v mnoha projektech jako je například MPlayer, VLC player, Avidemux a díky integraci ve FFDshow také v balíčcích kodeků jako CCCP a K-lite.

O kódování videa se stará knihovna *libavcodec* a mezi obsažené kodeky patří následující video a audio kodeky:

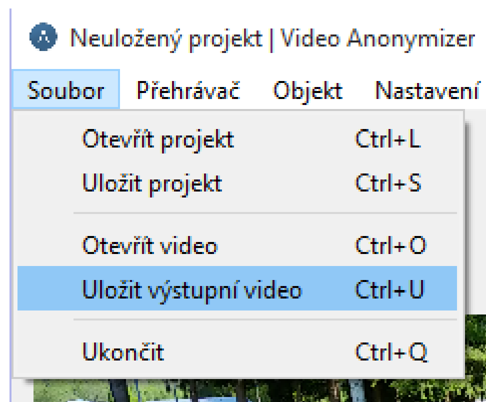
- Video
    - Motion JPEG, lossless (bezztrátový) JPEG
    - H.261, H.263, H.263+, H.264
    - Standardní ISO MPEG-4(DivX, XviD kompatibilní)
    - prvotní MPEG-4 varianta od MS, v3 (DivX3); varianta 2 (použitý ve starých ASF souborech)
    - RealVideo 1.0 a RealVideo 2.0
    - MPEG-1 video a MPEG-2 video
    - HuffYUV bezztrátová komprese
    - ASUS Video v1 a v2
    - bezztrátový video kodek z FFmpeg
    - Sorenson video 1, Sorenson H.263 používaný ve Flash Video
    - DV Sony Digital Video
    - snow Experimentální vlnkově orientovaný kodek z FFmpeg
- Audio
    - mp2 MPEG Layer 2
    - ac3 AC3, alias Dolby Digital
    - adpcm_ima_wav IMA adaptivní PCM
    - sonic experimentální lossy/lossless kodek.

4 Uživatelská příručka

Následující podkapitoly shrnují seznam položek pro ovládání aplikace.

Otevření videa

Pro načtení videa v mediálním formátu zvolte v hlavní nabídce **Soubor -> Otevřít video** a vyberte příslušné video. Lze použít klávesovou zkratku *Ctrl + O*.



Uložení videa

Pro uložení výstupního videa zvolte v hlavní nabídce **Soubor -> Uložit výstupní video**. Výstupní video obsahuje všechny sledované objekty. Tyto objekty však nelze později měnit. Pro uložení výstupního videa lze použít klávesovou zkratku *Ctrl+U*. Pokud si přejete uložit rozpracovaný projekt, kde lze objekty později upravovat, použijte: *Uložení projektu*.

Otevření projektu

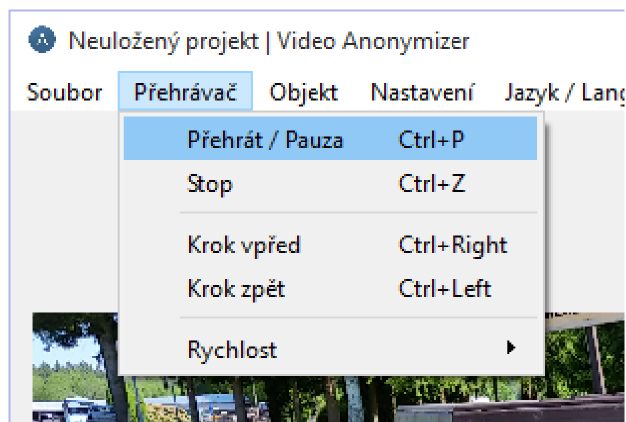
Rozpracované projekty vytvořené programem Anonymizer lze načíst pomocí hlavní nabídky **Soubor -> Načíst projekt**. Projekty lze načíst ve formátech JSON a XML. Lze použít klávesovou zkratku *Ctrl+L*. Pokud si přejete načíst nové video, použijte: *Otevřít video*.

Uložení projektu

Rozpracovaný projekt si můžete uložit pro pozdější úpravy v hlavní nabídce **Soubor -> Uložit projekt**. Po opětovném načtení projektu lze objekty a jejich nastavení dále upravovat. Projekt zůstane ve stavu, v jakém jste jej uložili. Nejedná se však o video soubor. Projekt lze uložit ve formátech JSON a XML. Lze použít klávesovou zkratku *Ctrl+S*. Pro vytvoření výstupního videa použijte: *Uložení videa*.

Přehrávání / Pozastavení videa

Pro přehrávání či pozastavení (pauzy) videa vyberte v hlavní nabídce **Přehrávač -> Přehrát / Pauza**. Lze použít i klávesovou zkratku *Ctrl+P* a tlačítko **Přehrát (Pauza)**, nacházející se v menu přehrávače pod videem. Pokud si přejete zastavit video (vrátit na začátek), použijte: *Zastavení videa*.



Zastavení videa

Pro zastavení videa vyberte v hlavní nabídce **Přehrávač -> Stop**. Lze použít i klávesovou zkratku *Ctrl+Z* a tlačítko **Stop**, nacházející se v menu přehrávače pod videem. Pokud si přejete video pozastavit (pauza), použijte: *Přehrávání / Pozastavení*.

Krokování videa

Video lze posunovat o jeden snímek vpřed i vzad vůči aktuálně zobrazenému snímku.

Krok vpřed
Pro krok vpřed vyberte v hlavní nabídce **Přehrávač -> Krok vpřed**. Lze použít i klávesovou zkratku *Ctrl+Right* a symbol **>**, nacházející se v menu přehrávače pod videem.

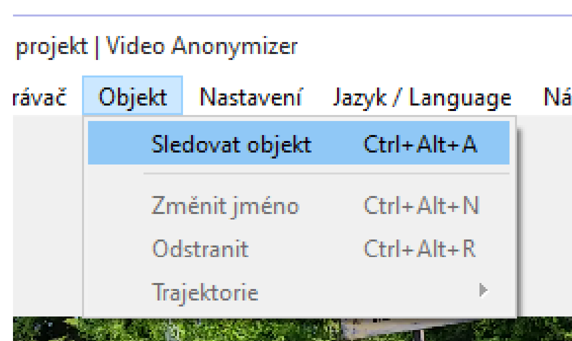
Krok zpět
Pro krok zpět vyberte v hlavní nabídce **Přehrávač -> Krok zpět**. Lze použít i klávesovou zkratku *Ctrl+Left* a symbol **<**, nacházející se v menu přehrávače pod videem.

Změna rychlosti
Přehrávač nabízí možnost změnit rychlost přehrávání videa. Aktuálně nastavená rychlost je zobrazena vpravo od nabídky přehrávače pod videem.

Rychleji
Rychlost přehrávání videa můžete zvýšit v hlavní nabídce **Přehrávač -> Rychlost -> Přehrávat rychleji**. Lze použít klávesovou zkratku *Ctrl+Up* a symbol "+" vedle zobrazení aktuálně nastavené rychlosti.

Pomaleji
Rychlost přehrávání videa můžete snížit v hlavní nabídce **Přehrávač -> Rychlost -> Přehrávat pomaleji**. Lze použít klávesovou zkratku *Ctrl+Down* a symbol "-" vedle zobrazení aktuálně nastavené rychlosti.

Skutečná rychlost
Rychlost přehrávání videa lze vrátit zpět na skutečnou rychlost videa v hlavní nabídce **Přehrávač -> Rychlost -> Skutečná rychlost**.



Sledování nového objektu
Pro přidání nového objektu ke sledování vyberte v hlavní nabídce **Objekt -> Sledovat objekt**. Lze použít i klávesovou zkratku *Ctrl+Alt+A* a tlačítko **Sledovat objekt**, nacházející se v menu přehrávače pod videem. Následně budete vyzváni k zadání názvu objektu a zvolení počáteční pozice objektu. Při výběru počáteční pozice lze libovolně používat přehrávač pro nastavení požadovaného snímku. Pomocí selekce vyberte oblast v daném snímku, která má být sledována: *Výběr objektu*.

Výběr objektu
Oblast vyberete stisknutím tlačítka myši ve snímku videa v místě, kde objekt začíná a tažením do místa, kde objekt v daném snímku končí. Po správném označení objektu tlačítko myši uvolněte. Pokud si přejete oblast změnit, vyberte novou oblast stejným způsobem, jak bylo popsáno výše. Předchozí výběr bude tak nahrazen výběrem novým.

Přejmenování objektu

Pro přejmenování aktuálně aktivního objektu (vybraného v sekci *Upravit objekt*) vyberte v hlavní nabídce **Objekt -> Změnit jméno**. Objekt přejmenujete i v sekci **Upravit objekt** pod záložkou **Obecné** pomocí tlačítka **Změnit jméno**. Lze použít i klávesovou zkratku *Ctrl+Alt+N*.

Odstranění objektu

Pro odstranění aktuálně aktivního objektu (vybraného v sekci "Upravit objekt") vyberte v hlavní nabídce **Objekt -> Odstranit**. Objekt odstraníte i v sekci **Upravit objekt** pod záložkou **Obecné** pomocí tlačítka **Odstranit**. Lze použít i klávesovou zkratku *Ctrl+Alt+R*.

Nastavení vzhledu objektu

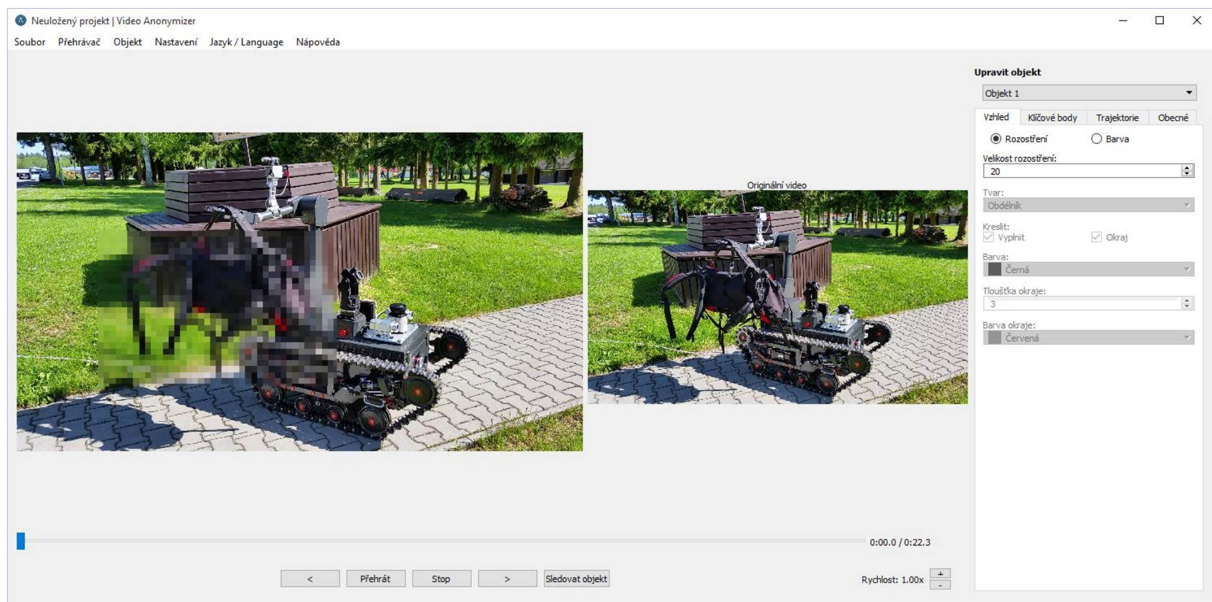
Vzhled aktuálně aktivního objektu (vybraného v sekci "Upravit objekt") lze nastavit na záložce **Vzhled**.

Rozostření nebo barva

Pokud si přejete rozmazat (rozkostičkovat) sledovaný objekt, zaškrtněte možnost **Rozostření**. Chcete-li celý objekt překrýt zvolenou barvou a/nebo nakreslit okraj kolem objektu, zvolte možnost **Barva**.

Velikost rozostření

U rozostření lze nastavit jeho velikost v položce **Velikost rozostření**. Velikost rozostření určuje, jak velké mají být čtverce, které rozostření tvoří. Menší hodnota znamená větší rozostření.



Výběr tvaru

Položka **Tvar** slouží k výběru tvaru objektu, který má být vykreslený na místo sledovaného objektu. Tato možnost funguje jen v případě vyplňování objektu barvou nebo kreslení okraje (ne při rozostření).

Vyplnění objektu

Po zaškrtnutí možnosti **Vyplnit** v položce **Kreslit** se celý sledovaný objekt vyplňuje barvou, která se nastavuje v položce **Barva**. Kromě předdefinovaných barev lze také *Vybrat vlastní barvu*.

Nakreslit okraj

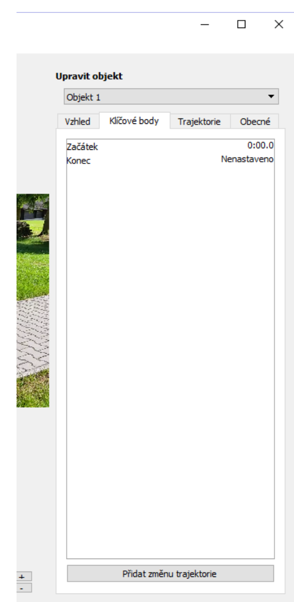
Po zaškrtnutí možnosti **Okraj** v položce **Kreslit** je kreslen okraj kolem sledovaného objektu. Tloušťka okraje se nastavuje v položce **Tloušťka okraje**. Okraj je kreslen barvou, která se nastavuje v položce **Barva okraje**. Kromě předdefinovaných barev lze také *Vybrat vlastní barvu*.

Výběr vlastní barvy

Vlastní barvu lze při nastavování barvy okraje a barvy objektu přidat po kliknutí na seznam barev a výběru první položky **+ Přidat novou**.

Klíčové body

V sekci **Upravit objekt** se nachází záložka **Klíčové body**. Zde jsou uvedeny body, kde dochází k uživatelem definované změně trajektorie. Existují tři typy bodů: **Začátek**, **Konec** a **Změna trajektorie**. Po kliknutí na daný bod jej lze upravit a podívat se na snímek, kde je definovaný. V případě "Změny trajektorie" lze bod i odstranit.



Začátek: Počáteční pozice objektu

Konec: Poslední snímek sledovaného objektu

Změna trajektorie: Změna pozice objektu na daném snímku. Slouží k úpravě trajektorie, pokud byla vypočítána nesprávně

Změna trajektorie

Pro změnu trajektorie aktuálně aktivního objektu (vybraného v sekci "Upravit objekt") vyberte v hlavní nabídce **Objekt -> Trajektorie -> Změnit trajektorii**. Lze použít i klávesovou zkratku *Ctrl+Alt+T* a tlačítko **Přidat změnu trajektorie**, nacházející se v sekci **Upravit objekt** na záložce **Klíčové body**. Výběr nové pozice objektu probíhá stejně jako přidávání nového objektu: *Výběr objektu*.

Změna počáteční pozice sledování

Pro změnu počáteční pozice sledování aktuálně aktivního objektu (vybraného v sekci "Upravit objekt") vyberte v hlavní nabídce **Objekt -> Trajektorie -> Změnit začátek**. Lze použít i klávesovou zkratku *Ctrl+Alt+B* a v sekci **Upravit objekt** na záložce **Klíčové body** u položky **Začátek** možnost **Změnit pozici**. Výběr nové počáteční pozice objektu probíhá stejně jako přidávání nového objektu: *Výběr objektu*.

Nastavení konce sledování objektu

Pro nastavení (změnu) konce sledování aktuálně aktivního objektu (vybraného v sekci "Upravit objekt") vyberte v hlavní nabídce **Objekt -> Trajektorie -> Nastavit konec sledování**. Lze použít i klávesovou zkratku *Ctrl+Alt+E* a v sekci **Upravit objekt** na záložce **Klíčové body** u položky **Konec** možnost **Nastavit konec sledování** (nebo **Změnit poslední snímek sledování** v případě, že konec již byl nastaven). Snímek vyberte pomocí ovládání přehrávače a potvrďte.

Nastavení sledování do konce videa

Pro nastavení sledování aktuálně aktivního objektu (vybraného v sekci "Upravit objekt") do konce videa vyberte v hlavní nabídce **Objekt -> Trajektorie -> Nastavit sledování do konce videa**. Alternativně lze v sekci **Upravit objekt** na záložce **Klíčové body** u položky **Konec** vybrat možnost **Nastavit sledování do konce videa**.

Průběžné vypočítávání trajektorie

Trajektorie se automaticky vypočítává průběžně vzhledem ke snímkům, které jsou zobrazovány přehrávačem. Pokud si přejete dopředu vypočítat celou trajektorii, použijte: *Vypočítání trajektorie*.

Zobrazit trajektorii

V sekci **Upravit objekt** se nachází záložka **Trajektorie**. Zde je vypsána celá trajektorie aktuálně aktivního objektu (vybraného v sekci "Upravit objekt"). "**x**" a "**y**" vyjadřují souřadnice objektu na daném snímku. "**š**" vyjadřuje šířku a "**v**" vyjadřuje výšku. Kliknutím na položku trajektorie můžete skočit na daný snímek.

Vypočítání trajektorie

Pro vypočítání celé trajektorie aktuálně aktivního objektu (vybraného v sekci "Upravit objekt") vyberte v hlavní nabídce **Objekt -> Trajektorie -> Vypočítat trajektorii**. Lze použít i klávesovou zkratku *Ctrl+Alt+O* a tlačítko **Vypočítat trajektorii**, nacházející se v sekci **Upravit objekt** na záložce **Trajektorie**. Vypočítání trajektorie dopředu tímto způsobem znamená, že pro daný objekt se vypočítá celá trajektorie a nebude již třeba *Průběžného vypočítávání*. Tím je samotná práce s videem urychlena a je dostupný kompletní seznam pozic objektu v záložce **Trajektorie**.

Zobrazovat originální video

Pro zobrazení originálního videa vedle upraveného videa vyberte v hlavní nabídce **Nastavení -> Zobrazit originální video**. Lze použít i klávesovou zkratku *Ctrl+Shift+H*. Originální video se zobrazuje napravo od upraveného videa.

Zobrazovat časové údaje

Pro zobrazení časových údajů snímků namísto jejich čísel vyberte v hlavní nabídce **Nastavení -> Zobrazit časové údaje**. Touto volbou se změní hodnoty v celém programu. Lze použít i klávesovou zkratku *Ctrl+Shift+T*.

Zobrazovat čísla snímků

Pro zobrazení čísel snímků namísto jejich časových údajů vyberte v hlavní nabídce **Nastavení -> Zobrazit čísla snímků**. Touto volbou se změní hodnoty v celém programu. Lze použít i klávesovou zkratku *Ctrl+Shift+N*.

Změna jazyka

Jazyk aplikace lze nastavit v hlavní nabídce menu **Jazyk/Language**. Změna jazyka se projeví až po novém spuštění aplikace Video Anonymizer.

5 Reference

[1] Bay, H., Tuytelaars, T. a Gool, L. SURF: Speeded Up Robust Features. In: *Computer Vision – ECCV 2006*. Berlín: LNCS Springer, s. 404-417, 2006. doi 10.1007/11744023_32. ISBN 978-3-540-33832-1.

[2] Rubinstein, R.Y., Dirk, P. K., *Simulation and the Monte Carlo Method*, Wiley & Sons, New York, 2008, ISBN: 978-0-470-17794-5.

[3] Gordon, N., J., Salmond, D., J., Smith, A., F., M., *Novel approach to nonlinear/non-Gaussian Bayesian state estimation*, Radar and Signal Processing, IEE Proceedings F In Radar and Signal Processing, 1993

[4] Nummiaro, K., Koller-Meier, E., Gool, V., L., *An Adaptive Color-Based Particle Filter*, 2002

[5] Isard, M., Blake, A., *Condensation – Conditional Density Propagation for Visual Tracking*, International Journal of Computer Vision, 1997

[6] Isard, M., Blake, A., *Contour Tracking by Stochastic Propagation of Conditional Density*, European Conference on Computer Vision, 1996

[7] Heap, T., Hogg, D. *Wormholes in Shape Spaces: Tracking through Discontinuous Changes in Shape, International Conference on Computer Vision*, 1998

[8] Luo, X., Huang, Y. *Visual Tracking With Singular Value Particle Filter*, International Workshop on Machine Learning for Signal Processing, 2010

[9] Pérez, P., Hue, C., Vermaak, J., Gangnet, M., *Color-Based Probabilistic Tracking*, ECCV 2002

[10] Khalid, S., M., Ilyas, U., M., Sarfaraz, S., M., Ajaz, A., M. *Bhattacharyya Coefficient in Correlation of Gray-Scale Objects*, Journal Of Multimedia, 2006

[11] Nguyen, H., T., Worring, M., Boomgaard, R., *Occlusion Robust Adaptive Template Tracking*, 2001

[12] Matthews, I., Ishikawa, T., Baker, S. *The Template Update Problem*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2003

[13] Jurie, F., Dhome, M. *Real Time Robust Template Matching*, BMVC, 2002

[14] nVidia Corporation: *CUDA C Programming Guide* [online]. Publikováno 10-2012 [cit. 2012], nVidia. Dostupné z <http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf>.

[15] Knihovna cereal [online]. Publikováno 2013 [cit. 08-2015]. Dostupné z <https://github.com/USCiLab/cereal>.