VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

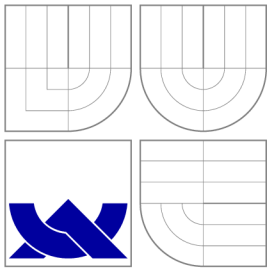
DETEKCE POHYBLIVÉHO OBJEKTU VE VIDEU NA CUDA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

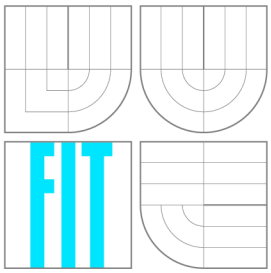
AUTOR PRÁCE
AUTHOR

Bc. MICHAL ČERMÁK

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE POHYBLIVÉHO OBJEKTU VE VIDEU NA CUDA

MOVING OBJECT DETECTION IN VIDEO USING CUDA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MICHAL ČERMÁK
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing.. ADAM HEROUT, Ph.D.
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce se věnuje sledování objektů ve videu z monokulární kamery za pomocí modelu sledovaného tělesa. Stav 3D objektu je určený pomocí minimalizace účelové funkce užitím částicového filtru. Účelová funkce je založena na podobnosti renderované scény a skutečného videa.

Abstract

This thesis deals with model-based approach to 3D tracking from monocular video. The 3D mode pose dynamically estimated through minimization of objective function by particle filter. Objective function is based on rendered scene to real video similarity.

Klíčová slova

Sledování objektu ve videu, 3D sledování, Částicový filtr, GPGPU, Obecné výpočty na grafické kartě, GPU, Grafická karta, nVidia CUDA, SUSAN, Zájmové body, Lokální deskriptory

Keywords

Video tracking, 3D tracking, particle Filter, GPGPU, General-Purpose Computation on Graphics Hardware, GPU, Graphic Processing Unit, nVidia CUDA, SUSAN, Interest points, Local descriptor.

Citace

Michal Čermák: Detekce pohyblivého objektu ve videu na CUDA, diplomová práce, Brno, FIT VUT v Brně, 2011

Detekce pohyblivého objektu ve videu na CUDA

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana doc. Ing. Adama Herouta Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Čermák
25. května 2011

Obsah

1 Úvod **3**

2 Sledování objektů ve videu **4**
2.1 Metody zdola nahoru 4
2.1.1 Mean–shift sledovač 4
2.2 Metody shora–dolu. Statistické metody ve sledování 5
2.2.1 Skryté Markovovy modely 5
2.3 Sekvenční Monte Carlo metody - částicový filtr 7
2.3.1 Monte Carlo 7
2.4 Shrnutí 10

3 Detekce lokálních příznaků **11**
3.1 Volba detektoru 12
3.2 Detektor SUSAN 12

4 Architektura a využití grafických karet pro obecné výpočty **15**
4.1 Historie z hlediska GPGPU 15
4.2 Motivace pro využití GPU 15
4.3 nVidia CUDA 16
4.3.1 Hardwarový model 17
4.3.2 Programátorský model 17
4.3.3 Využití GPU nVidia Fermi 18
4.4 OpenCL 18
4.5 Shrnutí 19

5 Návrh algoritmu **20**
5.1 Návrh implementace SUSAN na grafické kartě 20
5.2 Vyčíslení podobnosti 21
5.3 Částicový filtr 22

6 Testování aplikace a výsledky experimentů **24**
6.1 Validace algoritmu 24
6.2 Testy na reálné scéně 25

7 Závěr **33**

A Obsah DVD **38**

Seznam obrázků

2.1 Skrytý Markovův model 6

3.1 Princip metody SUSAN 12
3.2 Princip metody SUSAN–Velikost USAN oblasti v jednotlivých bodech . . . 13
3.3 Nastavení citlivosti metody 14

4.1 Vývoj aspektů výkonnosti 16
4.2 nVidia CUDA. Hardwarový a softwarový model 17

5.1 Algoritmus-vývojový diagram 23

6.1 Validační scéna: Hrací kostka 25
6.2 Citlivost váhové funkce na posunutí 26
6.3 Citlivost váhové funkce na posunutí. Upravená váhová funkce 27
6.4 Citlivost váhové funkce na rotaci 27
6.5 Render modelu ve výchozí poloze 28
6.6 První snímek videa 28
6.7 Lokální příznaky. Modrými body jsou označeny zájmové body renderované scény, červeně videa 29
6.8 Příklad sledování objektu v reálném videu 30
6.9 Relativní poloha kamery vůči automobilu a jeho vývoj v čase 31
6.10 Relativní úhel kamery vůči automobilu a jeho vývoj v čase 31
6.11 3D poloha v čase 32

Kapitola 1

Úvod

Sledování objektů ve videu je jedno z nejvíce rozvíjejících se oblastí počítačového vidění poslední dekády. Aplikaci nachází v mnoha oblastech lidské činnosti. Příkladem může být využití v nových způsobech interakce člověka s počítačem [20, 1], autonomní navigaci robotů nebo i automobilů [4]. Vzhledem k velké redundanci dat, mezi dvěma po sobě jdoucími snímky, nachází uplatněné též při kódování videa [19]. Známé je též použití v dohledových systémech [22].

Stejně jako v mnoha jiných oblastech zpracování signálů, i zde vítězí statistické přístupy založené na Bayesově statistice. Tento přístup má oproti klasickým metodám velkou výhodu v tom, že dokáže pracovat s neúplnou, zašuměnou informací. Navíc oproti klasické, takzvané frekventistické statistice, dokáže využívat takzvanou a-priorní informaci. Systémy založené na těchto algoritmech jsou navíc poměrně tolerantní k událostem, které u jiných sledovačů znamenají ztrátu sledovaného objektu.

Během studování článků na téma sledování objektů ve videu mě zaujalo, jak malá část jich využívá všechny dostupné a-priorní informace. Ač se to nezdá, tak poměrně běžnou informací tohoto typu je 3D polygonální model sledovaného objektu. Ten může vznikat v době designu objektu (například automobilu), nebo není velkým problémem jej manuálně vytvořit v některém z dostupných CAD nástrojů. Další možností je využití komerčně dostupných 3D scannerů. Mým cílem v této práci je předvést metodu pro sledování ideálně tuhého tělesa, ke kterému právě takovýto 3D model máme k dispozici. Podmínka ne-elasticity tělesa je zřejmá. Pokud by těleso měnilo svůj tvar, musel by se dynamicky měnit i model tělesa, což je za hranicí prezentované práce.

Díky tomu, že máme k dispozici model tělesa, je možné zjistit jeho polohu a rotaci v daném čase. To nám oproti klasickým 2D metodám dává neocenitelnou informaci navíc. Představte si například situaci, kdy váš nový automobil bude vybaven několika málo kamerami, které budou sledovat provoz jiných automobilů v okolním provozu. Ze znalosti aktuální polohy svého a okolo jedoucího automobilu bychom byli schopní reagovat na nebezpečné situace mnohem rychleji a dokonce jim i přecházet.

V následující kapitole 2 je uvedeno základní rozdělení algoritmů a jsou zde též popsány základní algoritmy z každé skupiny. Následuje kapitola 3, popisující detekci významných bodů v obraze. Tato metoda je využita pro výpočet podobnosti 3D modelu a skutečné scény. Ve 4. kapitole je popsána architektura současných grafických karet a jejich využití pro akceleraci algoritmů z předešlé kapitoly. Návrh, implementace a testování algoritmu popisují poslední dvě kapitoly 5 a 6.

Původní zadání práce byla detekce pohybujících se objektů ve videu. Postupným vývojem však došlo k tomu, že byl objekt ve videu ne detekován, ale sledován.

Kapitola 2

Sledování objektů ve videu

Typický sledovač objektů ve videu obsahuje algoritmy dvou tříd [2, 21], jež definují principiální přístup ke sledování. Přístup zdola–nahoru *bottom–up* a shora–dolu *top–down*. V přístupu zdola–nahoru je definována reprezentace a určen stav sledovaného objektu. Příkladem může být algoritmus detekce blobů [10], detekce pomocí mean–shift algoritmu [2] a další. Přístup shora–dolu se snaží určit dynamiku modelu, aktivně generovat hypotézy, jejichž správnost se následně snažíme ověřit oproti aktuálnímu snímku videa. Do této třídy algoritmů patří většina stochastických algoritmů, které jsou založeny na Kalmanově nebo částicovém filtrování.

2.1 Metody zdola nahoru

Tato kapitola ukazuje jednu ze základních metodu sledování objektů za pomocí technik od zdola nahoru. V podstatě vždy jedná o výpočetně velice nenáročné algoritmy, často ryze sekvenčního typu, jejichž výsledky často nejsou tak kvalitní jako u přístupu od shora dolů. V akademické sféře jim z těchto důvodů není věnována taková pozornost jako přístupu shora–dolu. Tyto argumenty vedly k tomu, že ani já jsem je hlouběji nestudoval a dále neimplementoval.

2.1.1 Mean–shift sledovač

Jak již bylo naznačeno, tato třída algoritmů vyniká malou výpočetní náročností, a proto se hodí do vestavěných zařízení, nebo tam kde je čas omezující. Například ve chvíli, kdy je sledování pouze jednou z mnoha komponent a výpočetní čas je nutné věnovat pro další činnosti např. rozpoznání nebo interpretaci zjištěné trasy. Prezentovaný algoritmus vychází z článku [2]
Algoritmus:
Vstup: Model sledovaného objektu q a jeho pozici v předchozím snímku y

1. Inicializuj pozici cíle v aktuálním snímku na y
2. Vypočti model tělesa v poloze $p(y)$ a pomocí vhodné metriky urči jeho vzdálenost od q.

$$m_1 = \rho(p(y), q) \tag{2.1}$$

3. Aplikuj mean shift. Vypočítáme novou polohu z. Funkce g definuje tvar sledovaného objektu, y_i je poloha n_h zkoumaných pixelů.

$$z = \frac{\sum_{i=1}^{n_h} g\left(\left\|\frac{y-y_i}{h}\right\|^2\right) y_i}{\sum_{i=1}^{n_h} g\left(\left\|\frac{y-y_i}{h}\right\|^2\right)} \tag{2.2}$$

4. Vypočítej model tělesa umístěného v poloze z a vyhodnoť jeho vzdálenost od q

$$m_2 = \rho(p(z), q) \tag{2.3}$$

5. Jestliže $m_2 < m_1$ pak $z = 0.5(y + z)$

6. Jestliže je přesnost dostatečná $\|z - y\| < \epsilon$ pak konec, jinak jdi na krok 1.

Jako vzdáleností metrika se často používá vzdálenost histogramů (*Bhattacharyya distance*). Může však být použita jakákoli metrika např. Podobnost definovaná pomocí texturních příznaků nebo význačných bodů.

2.2 Metody shora–dolu. Statistické metody ve sledování

Velké množství vědeckých problémů vyžaduje odhad stavu systému, který se mění v čase, na základě měření, která jsou zatížena šumem. V této kapitole se zaměřím na modelování těchto systémů s diskrétním časem za pomoci Bayesovy statistiky. Tato kapitola vychází z textů [5, 7]

2.2.1 Skryté Markovovy modely

Markovův proces, pojmenovaný po ruském matematiku Andrey Markovovi (1856–1922), je matematický model vývoje stochastického systému, který neobsahuje paměť. Výstup modelu tedy závisí pouze na aktuálním stavu Markovova procesu, tedy nezávisí na jeho minulosti případně budoucnosti.

Skrytý Markovův model (*Hidden Markov Model–HMM*) je speciálním případem Markovova modelu. Aktuální stav, ve kterém se model nachází, však není přímo pozorovatelný resp. měřitelný. Principiální schéma takového modelu je na obrázku 2.1

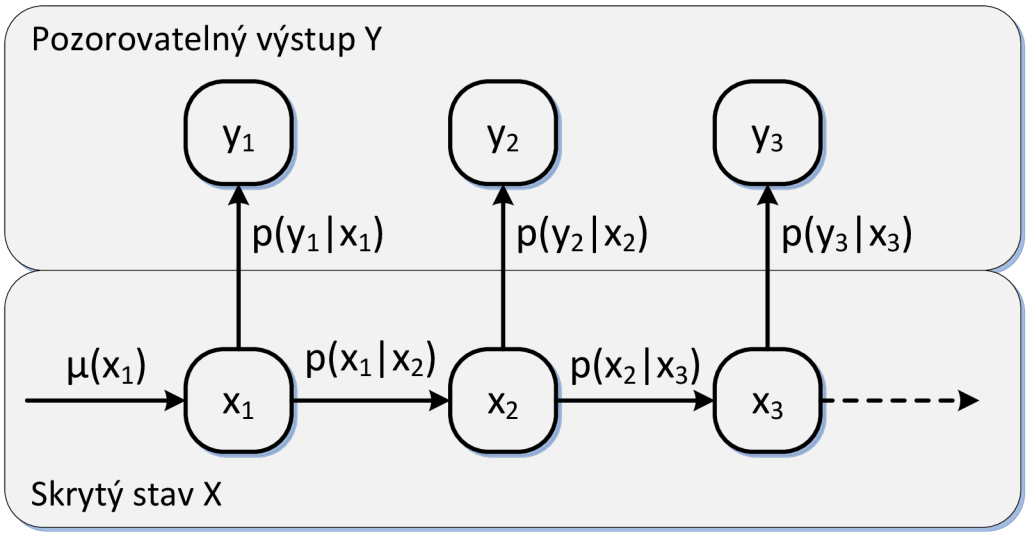
Matematicky lze toto schéma popsat snadno pomocí funkcí hustoty rozdělení pravděpodobnosti (*Probability Density Function–PDF*). Proces přechodu mezi jednotlivými stavy Markova procesu $\{X_n\}_{n\geq 1}$ definuje takzvaná apriorní (*a prior*) PDF.

$$X_1 \sim \mu(\cdot) \text{ a } X_n|(X_{n-1} = x_{n-1}) \sim f(x_n|x_{n-1}) \tag{2.4}$$

kde $\sim$ znamená distribuován podle, $\mu(x)$ je funkce hustoty rozdělení pravděpodobnosti a $f(x_n|x_{n-1})$ značí funkci hustoty rozdělení pravděpodobnosti při přechodu z x_{n-1} do x_n.

Vzhledem k tomu, že se jedná o skrytý markovův model, není proces zájmu $\{X_n\}_{n\geq 1}$ přístupný našemu pozorování. Máme ale přístup k jeho měřitelnému výstupu $\{Y_n\}_{n\geq 1}$. Vztah mezi stavem a pozorovaným výstupem má opět pravděpodobnostní charakter a nazývá se likelihood.

$$Y_n|(Y_{n-1} = y_{n-1}) \sim g(y_n|x_n) \tag{2.5}$$



Obrázek 2.1: Skrytý Markovův model

V dalším textu bude použita notace programu Matlab pro zápis vektorů $x_{i:j} = (x_i, x_{i+1}, \ldots, x_j)$. Z rovnice 2.4 a Markovova předpokladu pro čas n plyne rovnice

$$p(x_{1:n}) = \mu(x_1) \prod_{2 \leq k \leq n} f(x_k|x_{k-1}) \tag{2.6}$$

Stejně tak z rovnice likelihoodu 2.5 plyne pro čas n a vzájemně nezávislé hodnoty y_k vztah

$$p(y_{1:n}|x_{1:n}) = \prod_{1 \leq k \leq n} g(y_k|x_k) \tag{2.7}$$

Tyto dvě rovnice tvoří Bayesovský inferenční model, ze kterých za pomocí Bayesova teorému získáme

$$p(x_{1:n}|y_{1:n}) = \frac{p(x_{1:n})p(y_{1:n}|x_{1:n})}{p(y_{1:n})} \tag{2.8}$$

kde se $p(y_{1:n})$ často nazývá evidence a má charakter normalizační konstanty (není závislá na $x_{1:n}$).

$$p(y_{1:n}) = \int p(x_{1:n})p(y_{1:n}|x_{1:n})dx_{1:n} = \int p(y_{1:n}, x_{1:n})dx_{1:n} \tag{2.9}$$

Výsledná funkce $p(x_{1:n}|y_{1:n})$ se nazývá (marginální) posteriorní PDF a definuje Bayesovskou míru pravděpodobnosti $x_{1:n}$, ve chvíli, kdy pozorujeme $y_{1:n}$. Jinými slovy získáme novou informaci o stavu procesu zájmu $\{X_n\}_{n \geq 1}$, což je přesně to, o co nám při sledování jde především. Zjistit stav skrytého procesu v čase.

Nevýhodou tohoto přístupu je to, že s běžícím časem n stoupá složitost výpočtu. Chceme tedy v každém čase n aktualizovat naši znalost o $x_{1:n}$. Řešením je rekurzivní výpočet posteriorní PDF $p(x_{1:n}|y_{1:n})$ za pomoci $p(x_{1:n-1}|y_{1:n-1})$.

$$p(x_{1:n}|y_{1:n}) = p(x_{1:n-1}|y_{1:n-1}) \frac{f(x_{1:n}|x_{1:n-1})g(y_n|x_n)}{p(y_n|y_{1:n-1})} \tag{2.10}$$

kde $p(y_n|y_{1:n-1})$ je odhad nového výstupu y_n na základě předchozích $y_{1:n-1}$

$$p(y_n|y_{1:n-1}) = \int p(x_{n-1}|y_{1:n-1})f(x_n|x_{n-1})g(y_n|x_n)dx_{n-1:n} \tag{2.11}$$

Shrnutí

HMM představují velice obecnou skupinu metod pro řešení vývoje systému. Konceptuálně bez problému řešitelné za pomocí rovnic v této kapitole. Klíčový problém však vzniká při výpočtu těchto rovnic a zejména obsažených integrálů. Existují sice dvě výjimky, které ovšem přináší značné omezení.

- Konečné HMM (obdoba konečných automatů). Výpočet integrálů přechází díky konečnosti na výpočet konečný výpočet sumy. Tyto modely se často používají například při zpracování řeči [15, 23]
- Lineárně–Gaussovské modely, které předpokládají všechny PDF Gaussovské a funkce f a g lineární, což může být v některých případech značně omezující. Existuje analytický a efektivní algoritmus pro tento výpočet–Kalmanův filtr.

V obou případech se jedná o přesné a efektivně vyčíslitelné metody. Pokud se pohybujeme mimo rámec daný omezeními těchto metod, zejména pokud mluvíme o nelineárních a ne–Gaussovských problémech, pak neexistuje žádná metoda přesného, analytického výpočtu. Nezbývá nám tedy nic jiného než použít aproximační algoritmy, kterých existuje velké množství. Příkladem může být rozšířený Kalmanův filtr, který PDF aproximuje Gaussovým rozložením, různé aproximace pomocí sumy Gaussových funkcí apod. Ty ovšem často nepodávají správné výsledky a obvykle se jedná o implementačně velice složité algoritmy.

Řešením je použití tzv. sekvenčních Monte Carlo metod, které se v kontextu HMM také často nazývají částicovými filtry 2.3. Jejich velká výhoda oproti předchozí skupině algoritmů je ta, že nezavádí žádné zjednodušující předpoklady jako metody výše a navíc je jejich implementace velice snadná. Díky tomu se jedná se o často používané metody výpočtu nelineárních a ne–Gaussovských HMM.

2.3 Sekvenční Monte Carlo metody - částicový filtr

Sekvenční Monte Carlo (*Sequential Monte Carlo–SMC*) je množina metod umožňující přibližný výpočet jakékoli, často mnoha–dimenzionální sekvence pravděpodobnostních distribucí (PDF). V kontextu skrytých Markovových metod je SMC využíváno pro vyčíslení sekvence posteriorních PDF. SMC metody se v této aplikaci nazývají též částicovými filtry (*Particle Filter–PF*) případně částicové vyhlazení (*Particle Smoothing–PS*).

2.3.1 Monte Carlo

Jak bylo řečeno v předchozí kapitole 2.2.1, není možné rovnice pro výpočet posteriorní PDF $p(x_{1:n}|y_{1:n})$ vypočítat analyticky. Řešením je metoda Monte Carlo. Problém je definován jako vyjádření distribuční funkce.

$$\hat{p}(x_{1:n}|y_{1:n}) = \frac{1}{N} \sum_{1 \leq i \leq N} \delta_{X^i_{1:n}(x_{1:n})} \tag{2.12}$$

kde $X^i_{1:n} \sim p(x_{1:n}|y_{1:n})$ a δ je Dirakův impuls.

Zásadní problém ale nastává v tom, že nejsme schopni získat N vzorků z požadované distribuční funkce $p(x_{1:n}|y_{1:n})$. Máme však řešení - *Importance sampling–IS*. Tato metoda zavádí proporční distribuční funkci $q(x_{1:n}|y_{1:n})$, která musí mít podle [7] tyto vlastnosti

- Kde je nenulová $p(x_{1:n}|y_{1:n})$, měla by být nenulová i $q(x_{1:n}|y_{1:n})$. V kontextu MC metody by mělo platit, že tam kde se „vyplatí“ vzorkovat $p(x_{1:n}|y_{1:n})$ musí to samé platit o $q(x_{1:n}|y_{1:n})$

$$p(x_{1:n}|y_{1:n}) > 0 \Rightarrow q(x_{1:n}|y_{1:n}) > 0 \tag{2.13}$$

- $q(x_{1:n}|y_{1:n})$ je snadno vzorkovatelná.
- Obě distribuční funkce by měli být podobné.

Zavádíme tzv. váhu (*Importatnce Weight*), která definuje vztah mezi $p(x_{1:n}|y_{1:n})$ a $q(x_{1:n}|y_{1:n})$

$$w(x_{1:n}, y_{1:n}) = \frac{p(x_{1:n}, y_{1:n})}{q(x_{1:n}|y_{1:n})} \tag{2.14}$$

Pak můžeme $p(x_{1:n}|y_{1:n})$ vyjádřit pouze pomocí proporční PDF $q(x_{1:n}|y_{1:n})$ a váhy $w(x_{1:n}, y_{1:n})$

$$p(x_{1:n}|y_{1:n}) = \frac{w(x_{1:n}, y_{1:n})q(x_{1:n}|y_{1:n})}{p(y_{1:n})} \tag{2.15}$$

kde

$$p(y_{1:n}) = \int w(x_{1:n}, y_{1:n})q(x_{1:n}|y_{1:n})dx_{1:n} \tag{2.16}$$

Nyní můžeme aproximovat $q(x_{1:n}|y_{1:n})$ pomocí klasické Monte Carlo metody. Tato možnost plyne z jedné podmínek, které jsme na toto proporční PDF kladli. Musí z ní být možné vzorkovat.

$$\hat{q}(x_{1:n}|y_{1:n}) = \frac{1}{N} \sum_{1 \leq i \leq N} \delta_{X^i_{1:n}(x_{1:n})} \tag{2.17}$$

kde $X^i_{1:n} \sim q(x_{1:n}|y_{1:n})$. Tento výsledek můžeme dosadit do rovnic 2.15 a 2.16. Získáme hledanou aproximaci naší posteriorní $p(x_{1:n}|y_{1:n})$ PDF.

$$\hat{p}(x_{1:n}|y_{1:n}) = \sum_{1 \leq k \leq N} W^i_n \delta_{X^i_{1:n}(x_{1:n})} \tag{2.18}$$

kde W^i_n je normalizovaná váha

$$W^i_n = \frac{w(X^i_{1:n}, y_{1:n})}{\sum_{1 \leq k \leq N} w(X^i_{1:n}, y_{1:n})} \tag{2.19}$$

Zavedením proporcionální PDF $q(x_{1:n}|y_{1:n})$ je tedy problém nemožnosti vzorkovat $p(x_{1:n}|y_{1:n})$ vyřešen. Vyvstává však další problém. Vzhledem k tomu, že nyní vycházíme z rovnice 2.8, opět nastává problém zvyšující se výpočetní složitosti se stoupajícím časem n. Řešením tohoto problému je zavedení rekurzivního výpočtu proporční PDF.

$$q(x_{1:n}|y_{1:n}) = q(x_{1:n-1}|y_{1:n-1})q(x_{1:n}|y_{1:n}, x_{n-1}) \tag{2.20}$$

Tedy v čase n může být sada vzorků X^i_n vzorkována z $q(x_{1:n}|y_{1:n}, X_{n-1})$. Dosazením předchozí a upravené rovnice 2.10 do rovnice váhy 2.14 získáme

$$w(x_{1:n}, y_{1:n}) = \frac{p(x_{1:n-1}|y_{1:n-1})}{q(x_{1:n-1}|y_{1:n-1})} \frac{f(x_n|x_{n-1})g(y_n|x_n)}{q(x_{1:n}|y_{1:n}, X_{n-1})} = w(x_{1:n-1}, y_{1:n-1}) \frac{f(x_n|x_{n-1})g(y_n|x_n)}{q(x_{1:n}|y_{1:n}, X_{n-1})} \tag{2.21}$$

Z těchto rovnic se již dá sestrojit algoritmus *Sequential Importance Sampling–SIS*. Jeho nevýhodou je však fakt, že s rostoucím časem n stoupá rozptyl jednotlivých vzorků. Nakonec dojde po určité době k degradaci množiny vzorků do té míry, že všechny mimo jednoho budou mít, v porovnání s počáteční normalizovanou vahou, nepatrnou váhu blížící se nule. Normalizovaná váha jedné částice se bude blížit jedné. Mnoha-dimenzionální posteriorní distribuční funkce pak bude aproximována pouze jedním vzorkem/částicí, což je zjevně špatně.

Řešením je převzorkování (*Resampling*). Základní algoritmus převzorkování je dán tím, že vzorky/částice replikujeme s pravděpodobností, která je dána jejich normalizovanou vahou. Tím se do nové množiny vzorků dostanou s velkou pravděpodobností částice, která dobře aproximují distribuční funkci. Tento algoritmus se v literatuře nazývá *Sequential Importance Sampling Resampling–SISR*, nebo také jednoduše částicový filtr.
Algoritmus:

- V čase $n = 1$:
  1. Vzorkuj N částic $X_1^i \sim q(x_1, y_1)$
  2. Vypočti novou váhu
$$w(X_1^i, y_1) = \frac{\mu(X_1^i) g(y_1|X_1^i)}{q(X_1^i|y_1)} \tag{2.22}$$
  3. Normalizuj všechny váhy podle vzorce 2.19
  4. Převzorkuj $\{X_1^i, W_1^i\}$ pro získání nových vzorků $\{X_1^i\}$
- V čase $n \leq 2$:
  1. Vzorkuj N částic $X_n^i \sim q(x_n, y_n, X_{n-1}^i)$
  2. Vypočti novou váhu
$$w(X_{n-1:n}^i, y_n) = \frac{f(X_n^i|X_{n-1}^i) g(y_n|X_n^i)}{q(X_n^i|y_n, X_{n-1}^i)} \tag{2.23}$$
  3. Normalizuj všechny váhy podle vzorce 2.19
  4. Převzorkuj $\{X_{1:n}^i, W_n^i\}$ pro získání nových vzorků $\{X_{1:n}^i\}$

V obou případech je pak aproximace posteriorní distribuční funkce dána vzorcem 2.18. Konečně se dostáváme k tomu, že pomocí této aproximace můžeme určit parametry skrytého procesu X_n. Často používaná je maximální hodnota posteriorní PDF (*Maximum a Posteriori–MAP*) daná jako

$$arg\,max\ \hat{p}(x_n|y_{1:n}) = arg\,max\ \{W_n^i X_n^i(x_n)\} \tag{2.24}$$

Druhou metodou, často používanou při sledování objektů ve videu je střední hodnota posteriorní pravděpodobnosti.

$$mean\ \hat{p}(x_n|y_{1:n}) = \sum_{1 \leq i \leq N} W_n^i X_n^i(x_n) \tag{2.25}$$

2.4 Shrnutí

V této kapitole jsou matematických aparát pro výpočet posteriorní PDF, ze které se přeneseně zjistit i stav sledovaného objektu. Zmíněný Kalmanův filtr je sice v této oblasti stále více používaný, má však několik zásadních nevýhod.

- Vzhledem k jeho lineární povaze jsme velice často omezeni v modelování dynamiky sledovaného tělesa.
- Funguje dobře pouze v případech, kdy se dá dobře odlišit sledovaný objekt od podobných objektů na pozadí. Pokud ne, je nutné, aby distribuční funkce byly multimodální, což Kalmanův filtr neumožňuje.
- Ve chvíli, kdy se sledovač založený na této metodě jednou selže, nedokáže se již zotavit. Příkladem může být situace, kdy sledovač zaměří nesprávný cíl na pozadí.

Všechny tyto nevýhody odstraňuje sledovač založený na částicovém filtrování. Navíc je výborně paralelizovatelný, což je vlastnost z pohledu téma této práce zásadní. Monte Carlo metody mají navíc tu vlastnost, že se zvyšujícím se počtem částic/vzorků klesá chyba lineárně. Výkon grafické karty tedy může pomoci ke zpřesnění výpočtu posteriorní PDF a tím i k přesnosti sledování. Výše zmíněné důvody vedly k volbě částicového filtru jako algoritmu pro implementaci.

Kapitola 3

Detekce lokálních příznaků

Pro výpočet likelihoodu je použitý detektor lokálních příznaků. V následující kapitole jsou shrnuty základní požadavky na detektor a je představen konkrétní detektor SUSAN použitý v návrhu a implementaci. Často budou zaměňovány termíny lokální příznak (local feature) a význačný bod (interest point). V kontextu této práce a použité metody se jedná o synonyma, neplatí to však obecně[18].

Lokální příznaky (local features) je podmnožina pixelů obrazu, které se jistým způsobem liší od svého bezprostředního okolí. Jsou spojeny se změnou jedné nebo více vlastností obrazu (nejčastěji intenzity, barevné informace nebo textury)[18]. Zavádí se zejména kvůli potřebě redukovat výpočetní náročnost aplikací, které provádí hledání souvislosti ve dvou nebo více obrazech. Výpočet hrubou silou, kdy bychom porovnávali vzájemně všechny pixely dvou obrazů, bohužel není, ani v dnešní době, výpočetně dostupný. Řešením je zvolit určitou podmnožinu vhodných bodů, které je možné spolehlivě detekovat v obou odrazech, a provádět výpočty pouze nad touto podmnožinou[9]. Požadavky na použitý detektor:

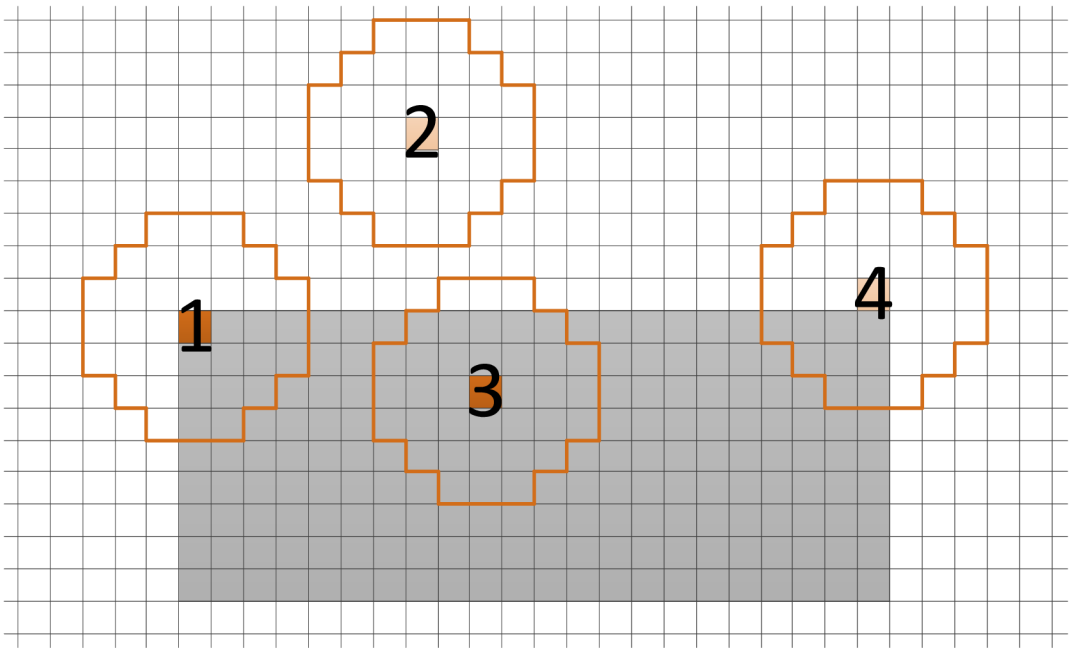
- Detektor by měl najít všechny význačné body, a naopak by neměl detekovat nesprávné.
- Opakovatelnost. Mějme dvě scény nahlížené z různých úhlů, pod různými světlenými podmínkami apod. Co největší část lokálních příznaků by měla být detekována ve shodné poloze ve scéně. Podíl nalezeného počtu význačných bodů ku celkovému počtu význačných bodů v obraze nazýváme míra opakovatelnosti (repeatability rate).
- Kvantita. Počet detekovaných příznaků by měl být dostatečně velký, tak aby i pro malé objekty detekoval dostatečný počet příznaků, ale zároveň dostatečně dobře redukovat obrazovou informaci. Hustota příznaků by měla odrážet obrazovou informaci. V představené detektoru SUSAN a dalších, je možné měnit hustotu příznaků měnit dynamicky, podle potřeby aplikace, změnou několika málo parametrů detektoru.
- Přesnost. Detekované příznaky by měli být přesně lokalizovány bez ohledu na proměnné vlastnosti scény.
- Efektivita. Jsou preferovány algoritmy, které jsou efektivní a dovolují použití v real-time aplikacích.
- Robustnost. Detektor by neměl být citlivý na kompresní artefakty, obrazový šum, diskretizační efekty, pohybové rozmazání apod.

3.1 Volba detektoru

Zvolený detektor by měl splňovat hlavní podmínku a tou je možnost efektivní implementace na GPU. Vzhledem k zvolené metodě výpočtu pomocí částicového filtru bude třeba počítat velké množství hypotéz (částic) a proto je právě efektivita více než zásadní. Další důležité vlastnosti jsou opakovatelnost, dobrá prostorová přesnost a robustnost. Naopak invariance k rotaci, škálování a affiní transformaci není díky použité metodě důležitá. V článku [18, 16] je shrnující tabulka, které porovnává různé detektory podle jejich vlastností. Podle výše popsaných požadavků byl vybrán SUSAN detektor[17], který se ukázal jako výborná volba pro implementaci na GPU.

3.2 Detektor SUSAN

Princip detektoru vysvětluje následující obrázek 3.1, který zobrazuje tmavý obdélník na světlejším pozadí. Zavádí princip masky (nejčastěji kruhové), kde se středový pixel nazývá nukleus (z lat. nucleus - jádro nebo oříšek). Pixely masky, které mají stejný, nebo podobný jas tvoří tzv. USAN obast (Univalue Segment Assimilating Nucleus). Velikosti USAN ob-



Obrázek 3.1: Princip metody SUSAN

lasti, společně s jejím centrem (centroid) a druhými momenty tvoří dostatečně silný deskriptor. Oproti jiným, známým metodám, se liší tím, že nepotřebuje počítat obrazové derivace. Díky tomu není třeba používat metody pro potlačení šumu v obraze, na které jsou tyto metody citlivé. SUSAN naproti tomu šum potlačuje díky integračnímu principu metody. Jak je vidět na dalším obrázku 3.2, velikost USAN oblasti může dosahovat velikosti rovné velikosti integrační masky (nucleus 2 na obrázku). Jen poznamenám, že velikost masky v tomto případě je 37 (pro $n \to \infty$ ve váhové funkci 3.1). Na hranách bude rovná přibližně

jedné polovině velikosti masky, a v blízkosti rohu bude v intervalu $(0, 0.5)$ v závislosti na velikosti úhlu rohové oblasti. Další ukázka obrázku s přidaným šumem je představena v původním textu[17]. Tento šum, pokud je dostatečně malý, nemá zásadní vliv na velikost USAN oblasti. Tato vlastnost plyne z integrační vlastnosti metody. Z předchozího odstavce

37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37
37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37
37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37
37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37
37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37
37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37
37	37	37	37	36	35	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	35	36	37	37	37	37	37
37	37	37	36	34	32	30	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	30	32	34	36	37	37	37
37	37	36	34	32	28	25	23	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	23	25	28	32	34	36	37	37
37	37	35	32	28	13	17	20	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	20	17	13	28	32	35	37	37
37	37	34	30	25	17	22	26	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	26	22	17	28	30	35	37	37
37	37	34	29	23	20	26	31	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	31	26	20	23	29	34	37	37
37	37	34	29	22	22	29	34	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	34	29	22	22	29	34	37	37
37	37	34	29	22	22	29	34	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	34	29	22	22	29	34	37	37
37	37	34	29	22	22	29	34	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	34	29	22	22	29	34	37	37
37	37	34	29	23	20	26	31	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	31	26	20	23	29	34	37	37
37	37	34	30	25	17	22	26	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	26	22	17	25	30	34	37	37
37	37	35	32	28	13	17	20	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	20	17	13	28	32	35	37	37
37	37	36	34	32	28	25	23	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	23	25	28	32	34	36	37	37

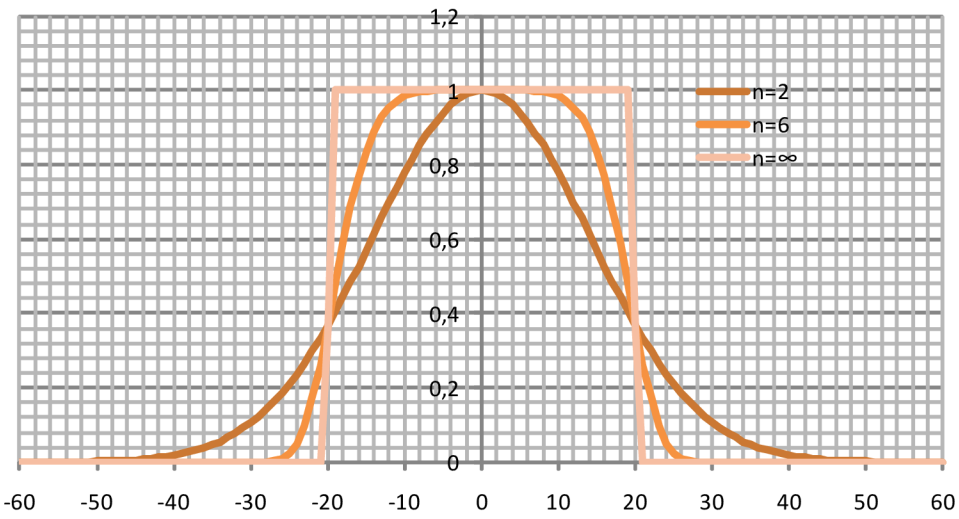
Obrázek 3.2: Princip metody SUSAN–Velikost USAN oblasti v jednotlivých bodech

plyne, že velikost USAN oblasti definje lokální vlastnost obrazu. Čím je USAN menší tím je pro nás oblast zajímavější. Z předchozí věty plyne i jméno metody SUSAN, která hledá nejmenší (Smallest) USAN (Univalue Segment Assimilating Nucleus). První otázka je, jak zjistit zda a do jaké míry je pixel masky $\vec{r}$ shodný se středovým pixelem $\vec{r_0}$. Hodnotící funkce je zvolena takto:

$$c(\vec{r}, \vec{r_0}) = e^{(\frac{I(\vec{r}) - I(\vec{r_0})}{t})^m} \tag{3.1}$$

Pro tři různé hodnoty exponentu m a $t = 20$ má funkce tento graf 3.3. Exponent m značí citlivost detektoru na malé změny jasu. Pokud je m malé, jsou pixelům s malou diferencí od středového dávány menší hodnoty funkce a naopak při větší vzdálenosti větší hodnoty. Práh t je pro 256 úrovní vstupního obrazu nastaven obvykle na hodnotu $t = 20$. Větší hodnota znamená větší USAN oblast a tím pádem i menší odezvu lokálního deskriptoru. Pro $m \to \infty$ funkce degraduje na skokovou funkci. Velikost USAN oblasti je dána sumou přes všechny pixely masky.

$$n(\vec{r_0}) = \sum_{\vec{r}} c(\vec{r}, \vec{r_0}) \tag{3.2}$$



Obrázek 3.3: Nastavení citlivosti metody

Vzhledem k tomu, že nás zajímají body, které mají velikost n USAN oblasti v rozmezí, které popisuje hrany a rohy, je možné provést rozhodnutí, zda se jedná zájmový bod či ne

$$R(\vec{(}r_0) = \begin{cases} g - n\vec{(}r_0) & \text{if } n\vec{(}r_0) < g, \\ 0 & \text{othwerwise} \end{cases}$$

Jednoduše potlačíme body, které mají odezvu větší než je tzv. geometrický práh g. Jeho nastavením například na $g = 0.5$zajistíme to, že odezva bude nenulová pro hrany a rohy.

Mimochodem, tato metoda, alespoň její první krok, je velice podobný metodě LBP (local binary patterns), které se používají ke klasifikaci texturních příznaků[14], kterou je též možno velmi dobře použít k detekci a sledování pohybujícího se objektu[24].

Kapitola 4

Architektura a využití grafických karet pro obecné výpočty

Tato kapitola osvětlí základní pojmy z oblasti grafických karet (*Graphics Processing Units–GPUs*), popíše motivace, které dávají důvod k jejich použití mimo rámec původního využití a popíše jazyky, v této oblasti používané.

4.1 Historie z hlediska GPGPU

Vývoj grafických akcelerátoru sahá až do sedmdesátých let minulého století, avšak až na přelomu století přinesl revoluci v podobě programovatelných jednotek (*shaderů*). Do této doby byli programátoři značně omezení fixním řetězcem. Nedlouho po uvedení programovatelného řetězce se v roce 2003 se objevily první články a práce, které využívaly možností soudobých grafických karet. Jedna z prvních byla i disertační práce Marka Harrise [6], který v této práci jako první použil termín GPGPU (*General-Purpose Computation on Graphics Hardware*).

Od té doby se grafické akcelerátory dále vyvíjely. Příchod nového grafického API DirectX verze 10 v roce 2006 si do jisté míry vynutil unifikaci jednotek pro výpočet vrcholů a fragmentů, které do té doby byly oddělené. První akcelerátor, podporující toto rozhraní byla nVidia 8800GTX. S ní přišel i první specializovaný GPGPU jazyk - CUDA, který se od té doby stále vyvíjí.

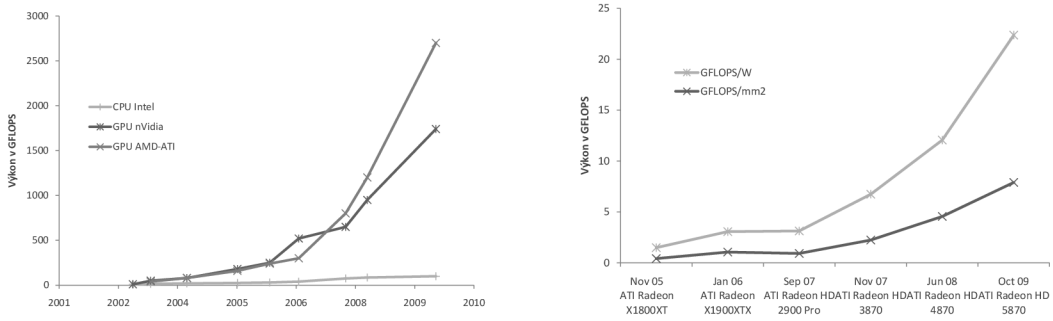
Jako u všech technologií, i zde trvalo několik let, než se prosadily. V první fázi byl GPGPU omezen pouze na akademickou sféru. Až v předchozím roce (2009) se začali objevovat aplikace, které mohli využívat i obyčejní uživatelé. Příkladem může být zpracování videa, obrazu nebo i dešifrování hesel za pomoci hrubé síly GPU. Rozšiřování GPGPU a nechuť společnosti nVidia poskytnout technologii CUDA ostatním výrobcům vedla na přelomu let 2008-2009 k definici obecného standardu pro GPGPU s názvem OpenCL. Poslední velkou událostí, která se na tomto poli udála, bylo uvedení rozhraní DirectX 11 a s tím spojený GPGPU jazyk DirectX Compute, který je zaměřen spíše na vývojáře her, ovšem jeho obecnému použití nic nebrání.

4.2 Motivace pro využití GPU

Jako nejspíše v každé publikaci, která v některé fázi výpočtu používá grafické karty, je i zde definována hlavní a jediná motivace pro k jejich využití. Výkon. Skutečně není jiný důvod

pro jejich využití než tento. Pokud aplikace vysoký výkon nepotřebuje, není žádný důvod je používat. Klasické procesory poskytují v jedno–vláknových aplikacích mnohem vyšší výkon, podporují multitasking, chráněný režim, vývoj aplikací pro ně je snazší. A jistě by se daly najít i další výhody.

Nicméně stále existují aplikace, kde je argument výkonu zásadní a výkon klasických procesoru nedostatečný. Může posunout dobu výpočtu ze dnů na hodiny, z týdnů na dny, může zajistit vykonání výpočtu real-time, tam kde to před tím nebylo možné.



(a) Porovnání výkonnosti GPU a CPU

(b) Tepelný výkon W na GFlop a výkon na mm^2

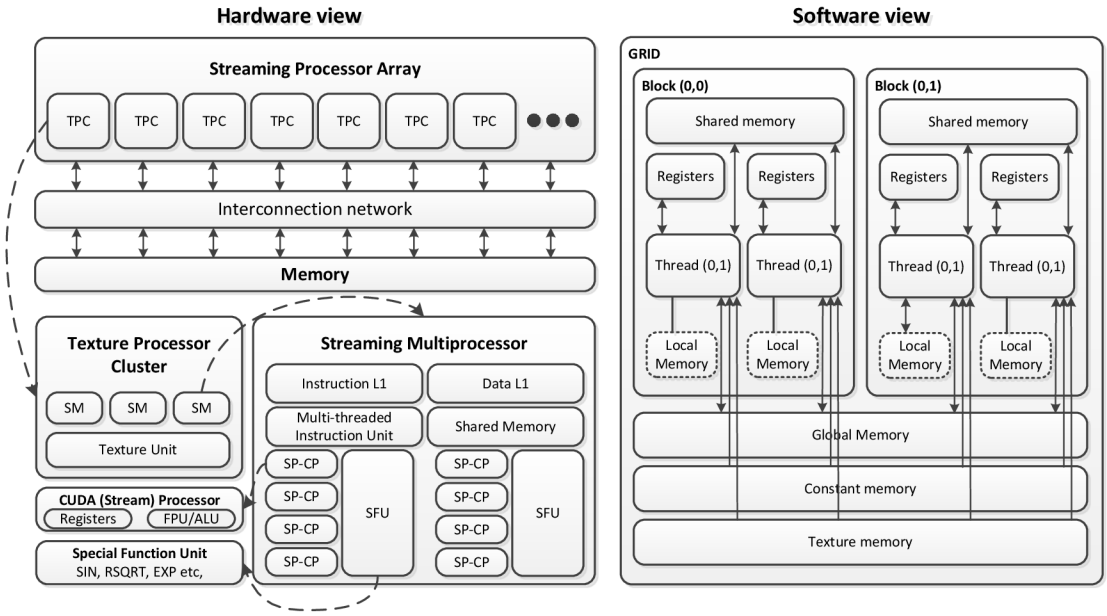
Obrázek 4.1: Vývoj aspektů výkonnosti

Jak je vidět z grafu 4.1(a) dnešní grafické čipy dosahují výkonu až 2.7 TFlops u nejnovější generace společnosti AMD, 1.7 Tflop u nadcházející generace nVidia. Pro porovnání aktuální čtyř a šesti–jádrové procesory se výkonem pohybují kolem 100 GFlops. Rozdíl ještě umocní to, že ani tohoto výkonu běžně nedosahují. Aplikace by musela intenzivně využívat technologie jako SSE (*Streaming SIMD Extensions*) a musela by využívat všechny jádra procesoru. Například využitím technologie OpenMP. Výsledkem tedy často je, že optimalizovat tímto způsobem pro procesor se nevyplatí, a je výhodnější použít pro akceleraci grafickou kartu. Mimochodem, s hlediskem výkonu a obecně architektury grafického čipu souvisí i graf 4.1(b), který ukazuje výkonnost v GFlops na jeden Watt vyzářeného tepla. Přesné hodnoty u procesorů bohužel nejsou uváděny, ale dají se odhadnout pro hodnoty výkonnosti 100 GFlops a 100W na přibližně 1 Gflops/W. Při dnešní snaze o co největší úspory energií je přesunutí výpočtů na grafickou kartu jistě přínosné i z tohoto, ekologického, hlediska.

4.3 nVidia CUDA

Nvidia Cuda je hardwarová a softwarová platforma, která umožňuje mapování datově paralelních výpočtů na grafické karty, bez potřeby mapovat výpočty na grafická API. Podporuje všechny významné operační systémy (Windows, Linux, MacOs) a to jak ve 32–bitové, tak ve 64–bitové verzi. Její syntaxe vychází z jazyka C, jsou ovšem podporovány i některé konstrukce jazyka C++. Konkrétně se jedná o šablony, které jazyk podporuje již od první verze. V poslední verzi 3.0, která je aktuálně k dispozici pouze registrovaným vývojářům, umožňuje navíc využívat polymorfního volání funkcí, výchozích parametrů funkcí, přetěžování operátorů a prostorů jmen [3]. Nad rámec těchto jazyků definuje minimalistickou sadu

konstrukcí, které slouží k vyjádření paralelismu.



Obrázek 4.2: nVidia CUDA. Hardwarový a softwarový model

4.3.1 Hardwarový model

Obrázek 4.2 [25] ilustruje hardwarový a softwarový model architektury CUDA. Z hardwarového hlediska se jedné o hierarchii, kde na nejvyšším stupni leží grafické karta, která obsahuje grafický čip a vysoce propustnou paměť. Grafický čip obsahuje pole tzv. *texture processor cluster–TPC*. Tato úroveň hierarchie stojí za škálovatelností v rozsahu nejen jednotlivých výkonnostních řad jedné generace, ale dokonce napříč několika generacemi. S lepším (menším) výrobním procesem se totiž na čip o stejné velikosti vejde více TPC. Další úroveň obsahuje multi–procesor. Jeho základní částí je osm 32– bitových stream procesorů, které umožňují základní aritmetické a logické operace nad celočíselnými a desetinnými čísly. Pro výpočet složitějších funkcí dále každý multi–procesor obsahuje pár SFC (*Special Function Unit*) jednotek. Každý multi–procesor dále obsahuje instrukční a datovou cache a také často využívanou sdílenou paměť, se kterou může pracovat kterýkoli z procesorů. Multi–procesor ovládá instrukční jednotka, která postupně spouští vykonání připravených vláken.

4.3.2 Programátorský model

Hardwarový model do značné míry definuje i ten softwarový 4.2. Na nejvyšším stupni hierarchie zde leží takzvaný kernel, což je program běžící na grafickém čipu. Do této doby bylo možné, aby v jedné chvíli byl spuštěn pouze jeden kernel, nebylo tedy možné vyčlenit část z multi-procesorů na jeden kernel a druhou část na jiný kernel. Jeden kernel je složen z pole tzv. bloků vláken, které se nazývají grid.

Plánovač postupně z gridu vybírá dosud nespočítané bloky a předává je k výpočtu na multi-procesory. Platí tedy, že jeden blok je počítán jedním blokem. Z této úvahy plyne i to,

že pro plné vytížení by mělo být více bloků než multi-procesorů, a to ideálně několikanásobně s přihlédnutím na zvyšování počtu multi-procesorů v budoucích generacích.

Každý blok se skládá typicky z 64 až 512 vláken. Disproporce mezi jejich počtem a počtem CUDA-procesorů je klíčová k dosažení vysokého výkonu. Plánovač vláken postupně předává k exekuci balíky vláken - *warpy*, které mají k dispozici veškerá potřebná data pro vykonání instrukce. Ostatní vlákna během této doby mohou čekat na data. Díky tomu je možné skrýt zpoždění hlavní paměti, a to i v případě kdy není použita žádná cache a dosáhnout tak vysokého výkonu. Podmínkou tohoto principu je samozřejmě paměť s vysokou propustností a také to aby paměť byla schopná obsloužit více požadavků v jednom čase. S jedním blokem je také spjata sdílená paměť, se kterou mohou pracovat všechna vlákna v daném bloku.

Na nejnižším stupni hierarchie stojí vlákno. Instrukce vlákna jsou vykonávány pro jednoduché instrukce na CUDA-procesorech, v případně složitějších operací na SFU. V závislosti na počtu vláken má každé vlákno k dispozici sadu registrů, které jsou přístupné v jednom taktu.

Z výše uvedených informací doporučeného počtu bloků a vláken v bloku je zřejmé, že hodnota počtu vláken, které by měl celkem kernel obsahovat, je minimálně několik jednotek až desítek tisíc. Na tento počet nezávislých pod-úloh by měla být úloha dekomponovatelná, aby byl výkon GPU maximálně využit.

4.3.3 Využití GPU nVidia Fermi

Na první čtvrtletí roku 2010 bylo oznámeno vydání nového akcelerátoru společnosti nVidia s kódovým označením Fermi [13]. Ten byl, jako první vyvíjen s důrazem především na GPGPU segment. Ne jako předešlé generace, které byly primárně grafickými akcelerátory. Stejně jako u předchozích generací, opět vzrostl počet SIMD výpočetních jednotek (nyní nazývané CUDA Cores) a to konkrétně z 240 u předchozí generace GTX-200 na 512 u Fermi. Teoretický výkon je při stejné frekvenci více než dvojnásobný.

Další výhodou je zvýšení výkonu v double výpočtech (tedy v 64–bitových číslech s pohyblivou řádovou čárkou). Tohoto zvýšení bylo dosaženo změnou architektury, kdy na double výpočty nejsou vyčleněny speciální jednotky, ale podílí se na nich vždy dvě 32–bitové CUDA Cores. Podíl výkonu ve 32–bitech (float) oproti 64–bitech (double) je tedy dvojnásobný. Využití 64–bitové přesnosti je diskutováno v [11], jakožto článku zabývajícím se využití GPU v sekvenčních Monte Carlo (SMC) výpočetech. Závěr autorů je takový, že 32–bitová čísla s pohyblivou čárkou jsou pro jejich aplikaci dostatečná a ani já nepředpokládám u mé aplikace opak. Pokud se však mýlím a z testování výsledné aplikace vyplyne nutnost použití 64–bitové přesnosti, pak bude výkon při použití Fermi několikanásobně vyšší než u předchozí generace.

Další důležitá skupina inovací je v přepracování plánovače vláken. Díky tomu je možné daleko rychlejší přepínání kontextu například mezi GPGPU a grafickou aplikací. Dále pak tato změna umožnila zavést jakousi obdobu simultánního multi-treadingu (SMT), který je známý z klasických procesorů a umožňuje lepší využití všech výpočetních jader. Pro více podrobností o této a dalších technologických změnách viz [13]

4.4 OpenCL

Jako alternativa k technologii CUDA byla poměrně nedávno (2008-12-08) vydaná specifikace otevřeného standardu OpenCL. Definuje API založené na jazyku C, konkrétně normě

C99. Umožňuje, stejně jako nVidia CUDA, explicitně specifikovat paralelní části kódu (kernel). Jejich běh však není vázán pouze na grafické karty, ale umožňuje exekuci i na klasických více–jádrových procesorech, nebo obecně, díky otevřenosti standardu, na kterékoli budoucí paralelní architektuře.

4.5 Shrnutí

I přes výhody platformy OpenCL byl jako implementační jazyk volena CUDA a to ze dvou hlavních důvodů. První je ten, že v době volby tématu této práce byla technologie CUDA více rozšířena, vývojový kit byl, díky delší době vývoje, stabilnější a existovalo pro ni více knihoven. Druhým důvodem je, že samotná technologie CUDA je psaná přímo na míru grafických karet společnosti nVidia a přejímání nových možností hardware je proto rychlejší (viz 4.3.3).

Kapitola 5

Návrh algoritmu

Princip zvoleného algoritmu je velice jednoduchý a popisuje ho vývojový diagram 5.1. Program na svém vstupu též očekává prvotní polohu sledovaného objektu a jeho model. Polohu je možné získat ze specializovaného detektoru (detektor pohybu, obličeje apod.). 3D model je možné získat pomocí metod 3d scanování, manuálním modelováním, nebo je k dispozici z jiných zdrojů. A aplikaci je pro načítání modelů použita knihovna Open Asset Import Library - Assimp[1].

Nejprve načteme snímek videa. Pokud není dispozici a dosáhli jsme konce videa je sledování u konce. Snímek bude přenesen do paměti grafické karty, které je určena pro textury (viz kapitola 5.1) a bude na něj aplikován algoritmus SUSAN. Mezitím je pro každou hypotézu polohy modelu v 3D prostoru vykreslený snímek jeho aktuální polohy. Pro vykreslení je využita knihovna OpenGL. Z důvodu jeho dalšího využití a aplikaci není vykreslován na obrazovku do okna, ale je využito rozšíření OpenGL framebuffer object (FBO). Ten umožňuje vykreslení snímku přímo do paměti grafické karty (Off-screen Rendering). Framebuffer object poskytuje přípojná místa pro jednotlivé výstupy framebufferu (Color, Depth a Stencil buffer). Color buffer připojíme k výstupní textuře. Textury v grafické paměti je navíc možné využívat a zpracovávat pomocí CUDA OpenGL Interoperability (dokumentace), čehož je využito. Rekapituluji, že je načten snímek videa, a podle jednotlivých hypotéz renderovány snímky modelu. Oba tyto výstupy jsou uloženy a zpřístupněny pomocí OpenGL Interoperability v grafické paměti jako textury. Následuje zpracování algoritmem výpočtu zájmových bodů (SUSAN), který je pro oba výstupy naprosto shodný.

5.1 Návrh implementace SUSAN na grafické kartě

Prvním krokem metody porovnání středového s okolními pixely (maskou). V tomto kroku jsem se rozhodl interpretovat vstupní obraz přenesený do paměti grafické karty jako texturu. CUDA má, stejně jako grafická rozhranní, možnost využívat texturovací jednotky (TPC. Viz obrázek 4.2). Ty umožňují prostorově orientované načítání 2D vstupní dat, jejich lineární filtraci a možnost adresovat texely mimo rozměr textury (mody CLAMP a WRAP). Oproti klasickému přístupy načítání 2D pole má toto zřejmé výhody. Pro hraniční model CLAMP je možné zcela vynechat podmínku, zda načítáme ze správné oblasti obrázku. Při konvolučních a obecně všech operacích, které využívají okolní pixel, kolem středového je buď nutné nechat jakési pásmo netečnosti vzdálené o r=poloměr masky od hrany obrazu. Druhou možností je právě režim CLAMP, který rozšiřuje možnost načítání

[1]Domovská stránka projektu: http://assimp.sourceforge.net/

obrazu za jeho původní hrany tak, že duplikuje hranové pixely. Pro metodu SUSAN to znamená pro většinu masek zasahujících a hranu textury pouze zvětšení USAN oblasti a tedy potlačení odezvy příznaku odpovídající této masce, což zjevně není chybou a navíc přináší nezanedbatelné zvýšení efektivity, zvláště pro malé vstupní textury a velké poloměry masek. V těchto případech je totiž pásmo netečnosti v poměru ke zbývající části obrazu velké.

Dalším důvodem je možnosti lineární interpolace textury. V původním textu o metodě SUSAN[17] je vždy používána maska o velikosti 37 pixelů (viz obrázek 3.1). Díky lineární interpolaci je možné navrhnout jiné masky, například takové, které budou mít větší počet vzorků poblíž nucleu, a se vzrůstající vzdáleností jejich počet bude klesat a navíc přitom nebudeme omezeni nutností načítat data ze středů pixelů. To umožní potlačit alias efekty, které díky použití této uniformní masky mohou vznikat. Vzdálenosti relativně od středu masky budou umístěny v poli, které bude umístěno v konstantní paměti GPU. Vzhledem k tomu, že vždy jeden instrukční warp bude načítat data ze stejné adresy konstantní paměti, nebude docházet k serializaci požadavků na čtení. GPU navíc obsahuje constant cache (pro Fermi o velikosti 8KB/multiprocesor), takže výpadek bude pouze při prvním použití tohoto pole.

Pro výpočet odezvy $c(\vec{r}, \vec{r_0})$ by na klasickém procesoru bylo výhodné použít předpočítanou LUT tabulku obsahující funkci zobrazenou na grafu 3.3. Tuto tabulku by bylo možné i na GPU, mělo by to ovšem dopad na výkon. Jak již bylo zmíněno, konstantní cache je pouze jedno-portová, zatímco jednotlivá vlákna přistupují do tabulky náhodně a je tedy nutná serializace. Jako efektivní se ukázalo provést výpočet hodnotící funkce $c(\vec{r}, \vec{r_0})$ přímo na GPU. Dělení konstantou se dá nahradit násobením převrácené hodnoty, pro pevné hodnoty mocniny se dá využít kombinace smyčky a šablony, která ji efektivně rozvine v době překladu. Výpočet funkce e^x se dá nahradit v GPU její rychlejší, ale méně přesnou verzí (i tak zdaleka za limitem využitelnosti).

V této verzi implementace jsem se rozhodl následující blok, vyčíslení podobnosti, vyhodnotit klasicky v procesoru. Důvod je ten, že detekovaných příznaků je v poměru k celkovému počtu pixelů obrazu malé množství. Při klasickém výpočetním vzoru, kdy každý pixel vyhodnocuje jedno vlákno, to vede ke značné neefektivitě výpočtu. Pole příznaků má ale podobu řídké matice (z definice klíčových bodu). Pro GPU a CUDA platformu existuje výkonná knihovna CUDPP[2]. CUDPP obsahuje mimo jiné i implementaci algoritmu pro kompaktaci řídké matice, takže je možné přenést z paměti grafické paměti do operační paměti pouze zlomek velikosti původní matice, což je důležité zvýšení efektivity.

Předchozí odstavce detailně vysvětlili implementaci zvoleného detektoru na grafické kartě a uvedly i ideu jak využít texturovacích jednotek k vylepšení metody SUSAN za hranici představené v původním článku[18].

5.2 Vyčíslení podobnosti

Na vstupu máme dvě kompaktní pole lokálních příznaků, které jsou přeneseny z paměti GPU do operační paměti. Jedna množina příznaků Y_V, které generuje snímek videa, druhá generována z vykreslení jedné hypotézy Y_M. Otázkou je, jak tyto množiny porovnat tak, aby výsledkem byla hodnota váhy částice.

Zavedeme nyní funkci $z(y_V, y_M)$, která definuje, zda k příznaku z renderovaného snímku y_M byl nalezený odpovídající příznak ve video snímku y_V. Toto rozhodnutí provedeme na

[2] http://gpgpu.org/developer/cudpp

základě toho, zda v euklidovském t-okolí nalezneme příznak.

$$z(y_V, y_M) = \begin{cases} 1 & \text{if } euclid(y_V, y_V) < t, \\ 0 & \text{othwerwise} \end{cases}$$

Chyba vzdálenosti dvou příznaků se dá vyjádřit jejich kvadratickou euklidovskou vzdáleností. Pro celý sledovaný objekt je tedy dán sumou vzdáleností přes všechny nalezené shody. Chyba bude započtena pouze pro stav, kdy funkce $z(y_V, y_M)$ bude nenulová

$$Err(Y_V, Y_M) = \sum_{y_V \in Y_V} \sum_{y_V \in Y_M} euclid(y_V, y_V) z(y_V, y_M) \tag{5.1}$$

Nastává však další problém. Může se stát, že funkce nalezne pouze několik málo shod (tedy pro větší část příznaků bude $z(y_V, y_M) = 0$. Zavedl jsem tedy normalizační koeficient $k(Y_V, Y_M)$, jako poměr nalezených shod, k celkovému počtu příznaků v renderovaném snímku.

$$k(Y_V, Y_M) = \frac{\sum_{y_V \in Y_V} \sum_{y_V \in Y_M} z(y_V, y_M)}{\sum_{y_V \in Y_V} \sum_{y_V \in Y_M} 1} \tag{5.2}$$

5.3 Částicový filtr

Hlavní smyčka výpočtu odpovídá algoritmu 2.3.1 částicového filtru prezentovanému výše. Nyní zbývá dodefinovat informace o skrytém stavu X, apriorní, proporční a likelihood funkce hustoty rozdělení pravděpodobnosti. Stav definuje polohu 3D objektu v šesti-rozměrném stavovém prostoru $X_0 \in \mathbb{R}^6$ takto

$$X_1 = \{x, y, z, R_x, R_y, R_z\} \tag{5.3}$$

Proporční PDF definujeme takto

$$q(x_n | y_n, X_{n-1}) = f(x_n | y_{1:n}) \tag{5.4}$$

neboli proporční PDF bude rovna apriorní. Vzorkování proporční distribuce pak nebude nic jiného než aplikace apriorní PDF na X_{n-1}. Váha pak bude přesně odpovídat likelihoodu. Viz rovnice 2.22 a 2.23. Verze s touto proporční PDF se velice často v oblasti počítačového vidění [8, 12].

Apriorní informace definuje dynamický model stavu. Ten je definován pomocí dvou komponent, deterministické A a stochastické w_{t-1}. Deterministická popisuje 3D těleso pohybující se konstantní rychlostí pohybu a rotace.

$$f(X_n, X_{N-1}) \sim A\, X_{N-1} + w_{t-1} \tag{5.5}$$

Poslední je definice distribuční funkce likelihoodu. Ten je definován jako velikost chybové funkce mezi aktuálním snímkem videa a aktuální částicí5.1. Výsledek navíc bude modifikovanná normalizační koeficientem5.1

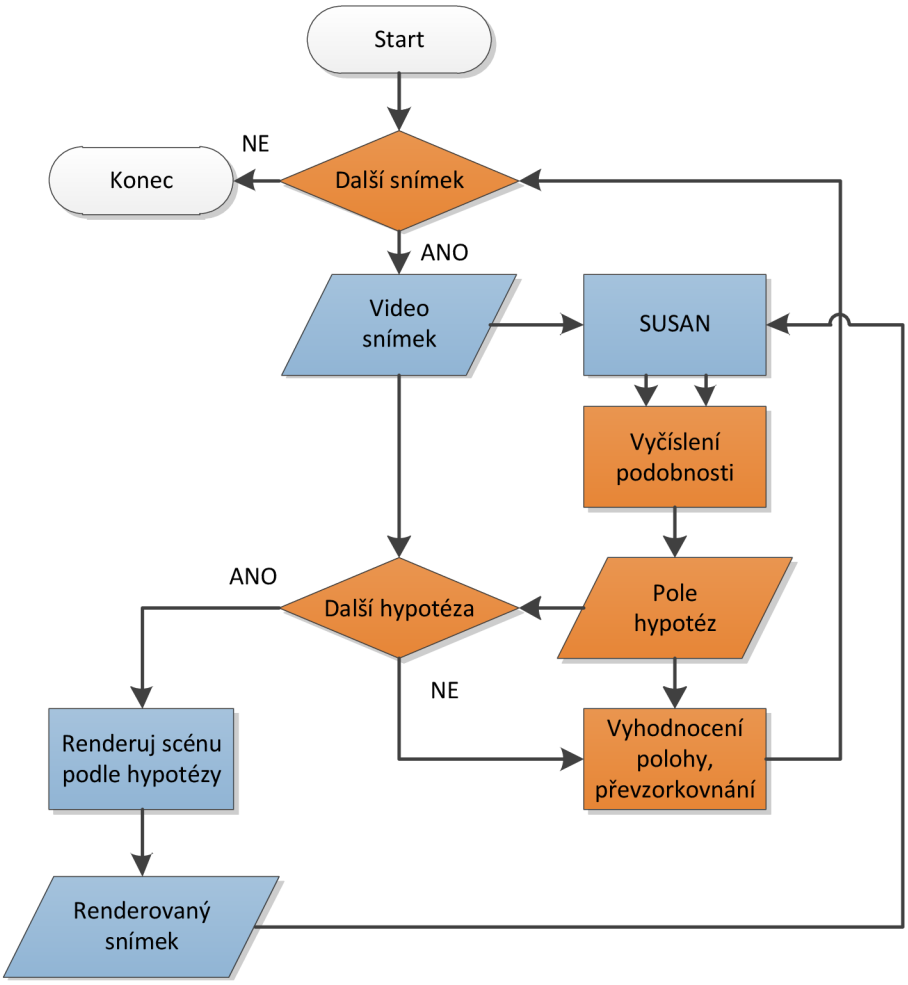
$$d = Err(Y_V, Y_M) k(Y_V, Y_M) \tag{5.6}$$

Vztah mezi vzdáleností snímků a nenormalizovanou vahou částice je určen

$$w_n = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{d^2}{2\sigma^2}} \tag{5.7}$$

jakožto nelineární Gaussova mapovací funkce. Nakonec tedy platí, že vztah pro distribuční funkci likelihoodu je dán takto

$$g(y_n | X_n) \sim w_n(X_n, y_n) \tag{5.8}$$



Obrázek 5.1: Algoritmus-vývojový diagram

Kapitola 6

Testování aplikace a výsledky experimentů

Podle návrhu aplikace z předchozí kapitoly byl implementován 3D sledovač objektů. V této kapitole bude empiricky dokázáno to, že funkce podobnosti renderovaného snímku a snímku videa dostatečně dobře reflektuje stav objektu. Tedy, že pro těleso v poloze odpovídající videu bude výsledná váha vysoká a naopak. Následují testy s polo-reálným videem vytvořeným v programu Blender a nakonec testy na reálném videu.

6.1 Validace algoritmu

Nejprve jsem chtěl empiricky dokázat, zda je implementovaný algoritmus funkční. Vytvořil jsem proto v modelovacím nástroji Blender scénu obsahující hrací kostku. Díky tomu jsem měl přesné určení stavu v každém okamžiku videa.

První test by měl vyzkoušet citlivost na posunutí v prostoru. Mějte tedy testovací objekt ve videu umístěný ve scéně se stavem $X_V = \{x, y, z, Rx, Ry, Rz\}$. Test spočívá v umístění modelu do scény s posunutím $X_M = \{x+\Delta x, y+\Delta y, z, Rx, Ry, Rz\}$ a vyhodnocení podobnosti-váhy částice v tomto stavu. Výsledkem je 3D graf s osami Δx, Δy a funkční hodnotou $w(\Delta x, \Delta y)$. Kostka byla v tomto testu postavena rovnoběžně s osami a jedničkou ve směru kamery. Na první pohled je z grafu 6.2 zřejmé, že nejvyšší odezvu má objekt ve správném stavu. Tedy pro stavy $X_V = \{x, y, z, Rx, Ry, Rz\}$ a $X_M = \{x+0, y+0, z, Rx, Ry, Rz\}$. Vzniká však problém s tím, že odezva stavu, která je způsobena pouze dvěma hranami kostky (a ne všemi 4, které jsou v tomto testu vidět) je příliš velká. Poměr výšky vedlejších, lokálních maxim, a maxima globálního bude odpovídat podle vztahu 6.1: $k(Y_V, Y_M) = 0.5$, což je ve shodě s grafem.

Dalším problémem je oblast grafu mezi lokálními a globálním maximem, kde odezva klesá k nule. To je způsobeno nelinearitou funkce 5.2. Jinými slovy je tomto stavu hrací kostka v poloze, kdy je vzdálenost příznaků, které leží na dvou hranách, velice blízká. Na kolmém páru hran se skokově změní stav funkce $z(y_V, y_M)$ z $z(y_V, y_M) = 0$ na $z(y_V, y_M) = 1$. Vzdálenost pro všechny příznaky, které leží na těchto hranách je tedy minimálně t a způsobí malou váhu částice.

Řešením je změnit funkci pro výpočet $z(y_V, y_M)$, nebo $k(Y_V, Y_M)$. Rozhodl jsem se změnit funkci $k(Y_V, Y_M)$, tak aby více preferovala stavy, kdy bude nalezeno více odpovídajících příznaků.

$$k_{new}(Y_V, Y_M) = k^2(Y_V, Y_M) \tag{6.1}$$

Obrázek 6.1: Validační scéna: Hrací kostka

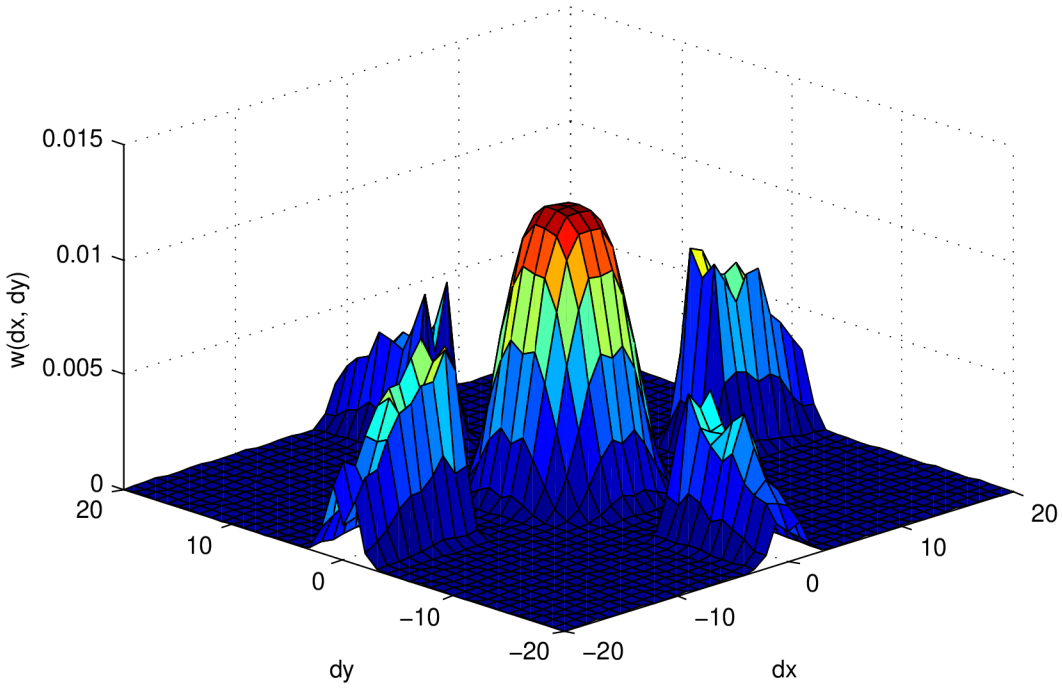
Pro tuto změněnou jsem vytvořil nový graf 6.3, který lépe vyjadřuje podobnost stavů. Obdobný test jsem provedl pro rotaci. Tedy stavy $X_V = \{x, y, z, Rx, Ry, Rz\}$ a $X_M = \{x, y, z, Rx + \Delta Rx, Ry + \Delta Ry, Rz\}$. Použita byla modifikovaná funkce $k_{new}(Y_V, Y_M)$. Výsledkem je graf 6.4 Z předchozích grafů 6.3 a 6.4 plyne, že váha částice dostatečně silně vyjadřuje podobnost stavu modelu a stavu objektu ve videu. Aplikace navíc dovoluje nepřímo měnit střední hodnotu funkcí z grafů tak, že změníme střední hodnou funkce 5.7.

6.2 Testy na reálné scéně

Po vyzkoušení vlastností algoritmu na ideální scéně „rotující kostka“ došlo i za zkoušky algoritmu v reálné scéně 6.10. Touto scénou je scéna automobilu, kterou jsem našel na video-portálu youtube.com[1]. Vzhledem k tomu, že v této chvíli jsem neměl informace o relativní poloze kamery a sledovaného automobilu, vlastnostech kamery (její obrazový úhel), nastavil jsem je manuálně. V této chvíli není absolutní přesnost důležitá, protože z principu algoritmus hledá nejlepší korespondenci lokálních příznaků. Za několik málo snímku sledovač nalezne správný stav. Čtenář si jistě všimne toho, že render se od skutečné scény značně liší.

- Pozadí videa není homogenní, oproti tomu pozadí renderu ano.

[1] http://www.youtube.com/watch?v=PKuaxTbFLEk

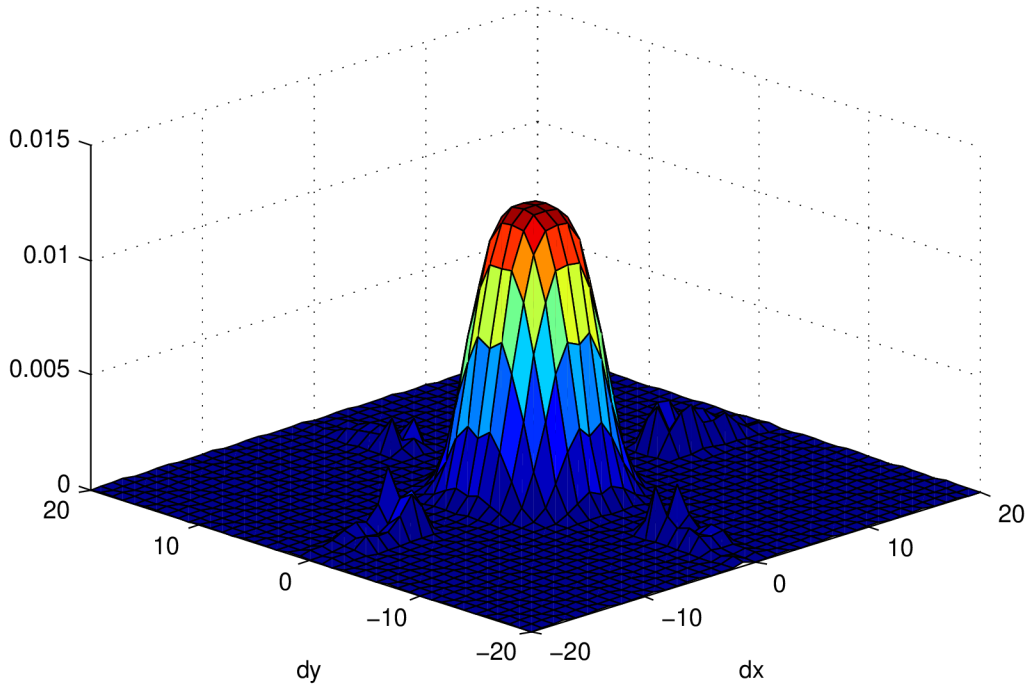


Obrázek 6.2: Citlivost váhové funkce na posunutí

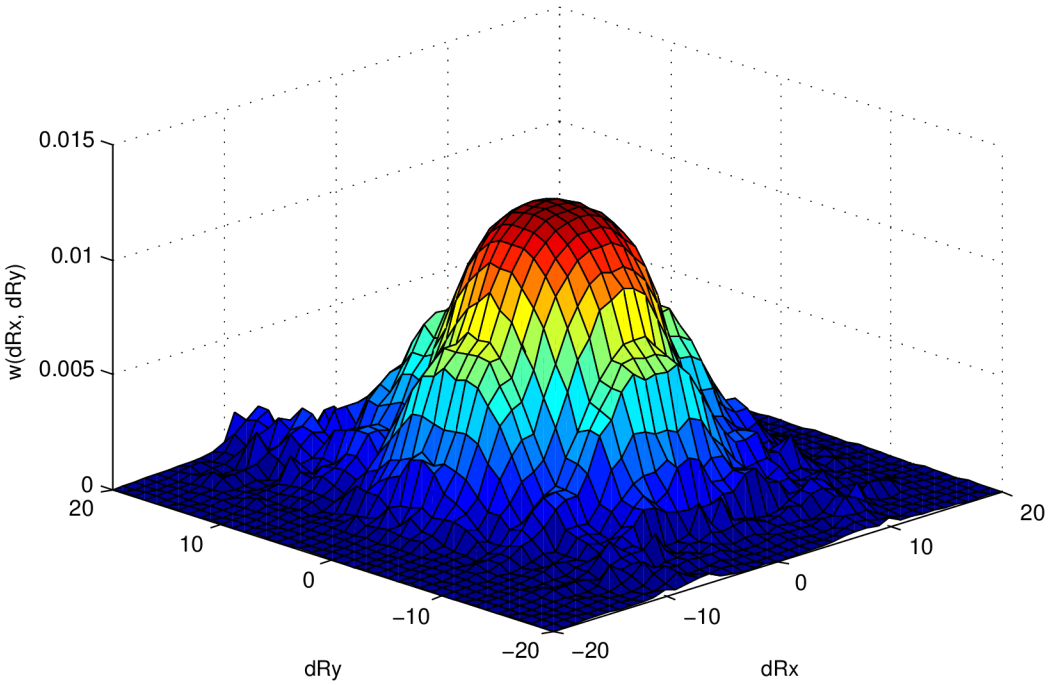
- Samotný model automobilu je jiný než automobil ve videu. Hlavním rozdílem je jiná barva, která ovšem díky použitému algoritmu nehraje vliv. Další rozdíly jsou v geometrii modelu. Zejména jsou to odlišná kola, absence střechy, průhledné přední sklo a několik dalších detailů.

- Osvětlení renderu je pouze základní s ambientní a difusní složkou. Použit je Phongův osvětlovací model. Naproti tomu osvětlení ve snímku videa je velice bohaté. Na přední části je několik odlesků, zadní stranu pokrývá stín kameramana. Samotný automobil též vrhá stín.

Z pohledu algoritmu je však hlavní, že větší část hran a důležitých detailů odpovídá modelu. Po spuštění program produkoval tyto výsledky 6.8. Odpovídající vstupní videa jsou uložena na přiloženém DVD. Výstupem programu není jen samotný render vypočítané polohy a rotace, ale i hodnota stavu. To je možné vynest do grafu v závisloti na čase t, respektive snímku n. Dalším možným výstupem je graf modelující pohyb automobilu v 3d prostoru. Bohužel není možné přímo určit přesnou polohu automobilu, jeho poloha však vizuálně velmi dobře odpovídá automobilu ve videu.



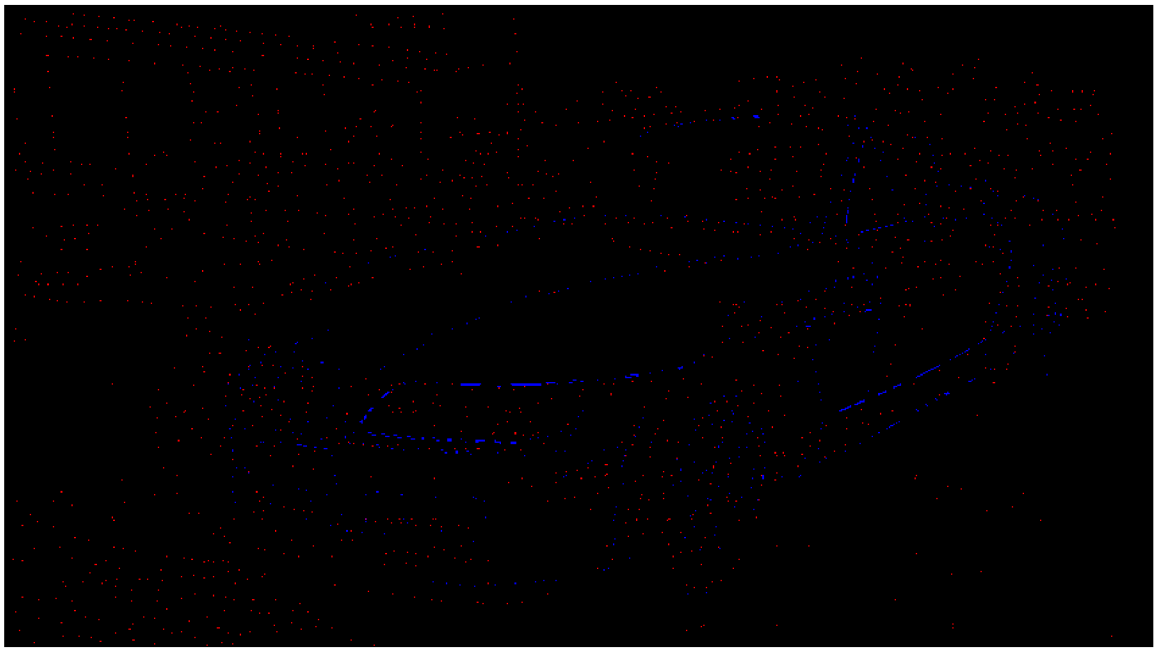
Obrázek 6.3: Citlivost váhové funkce na posunutí. Upravená váhová funkce



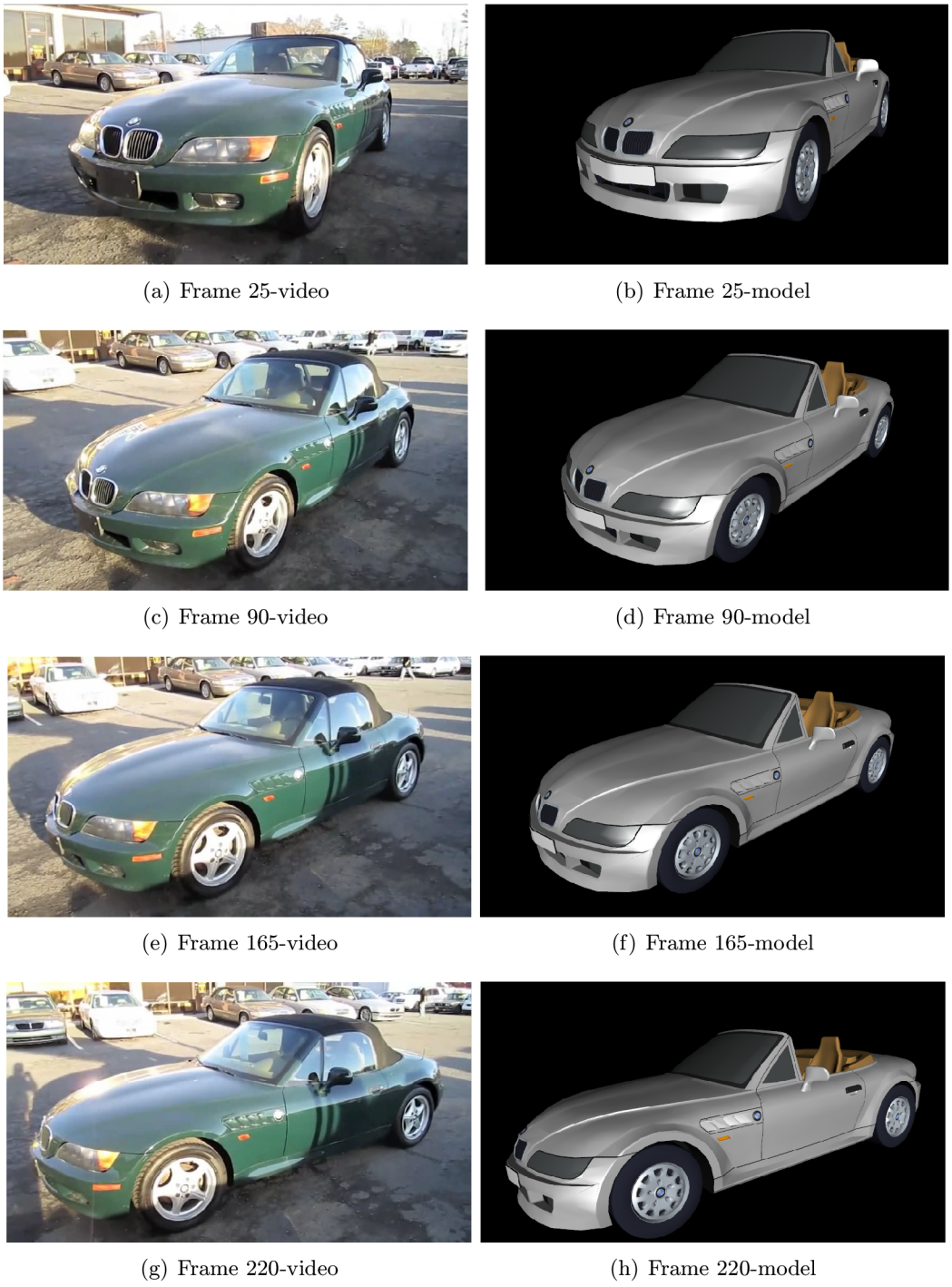
Obrázek 6.4: Citlivost váhové funkce na rotaci

Obrázek 6.5: Render modelu ve výchozí poloze

Obrázek 6.6: První snímek videa



Obrázek 6.7: Lokální příznaky. Modrými body jsou označeny zájmové body renderované scény, červeně videa



(a) Frame 25-video

(b) Frame 25-model

(c) Frame 90-video

(d) Frame 90-model

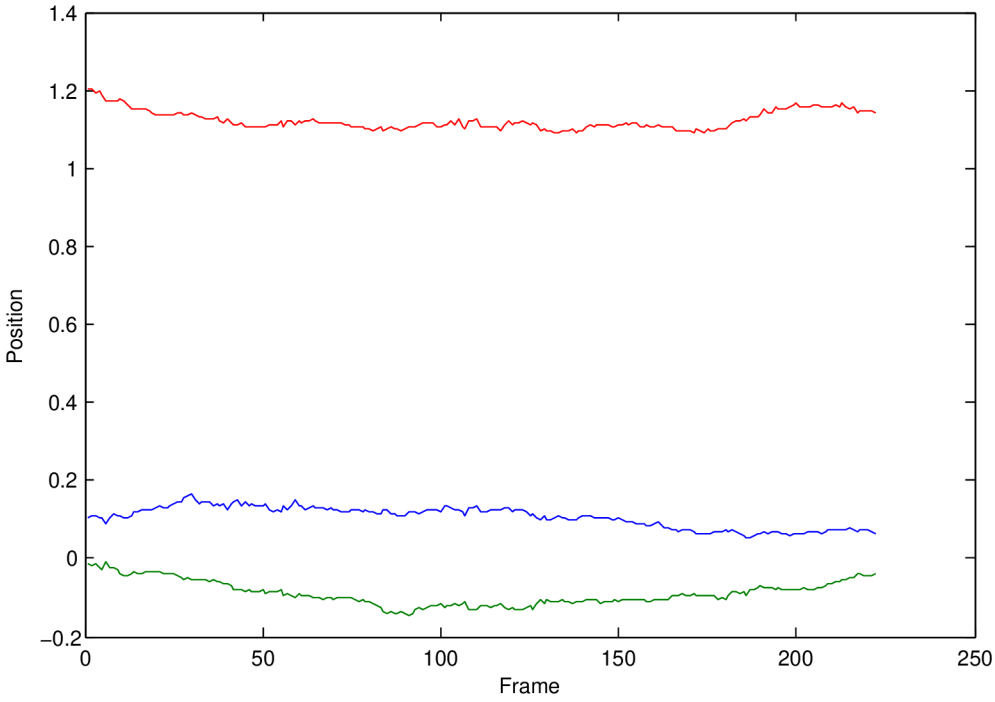
(e) Frame 165-video

(f) Frame 165-model

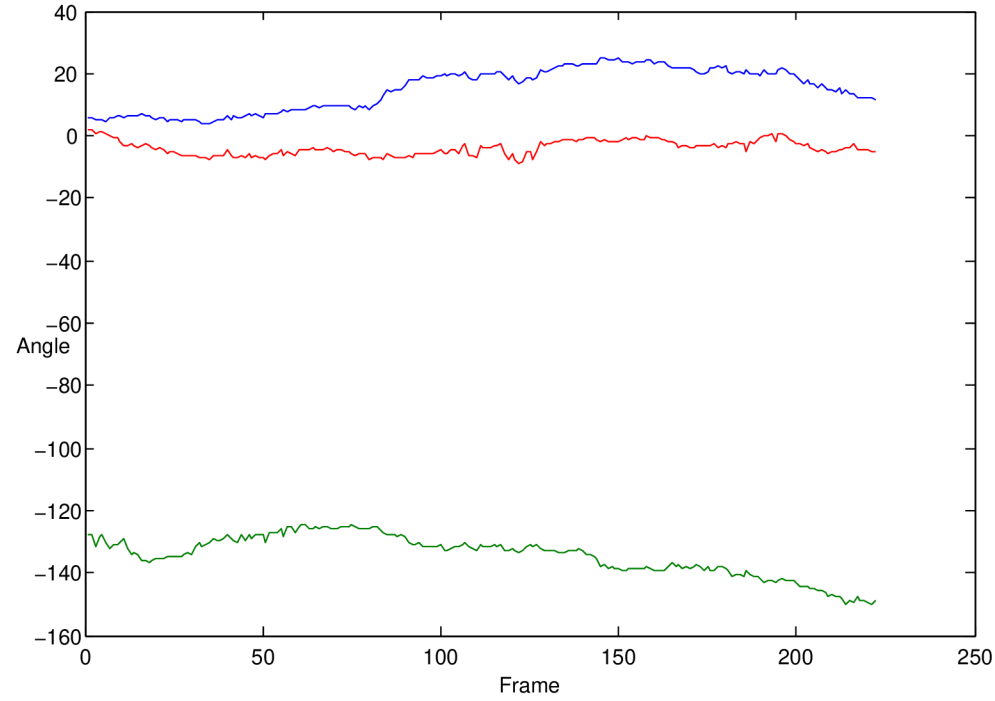
(g) Frame 220-video

(h) Frame 220-model

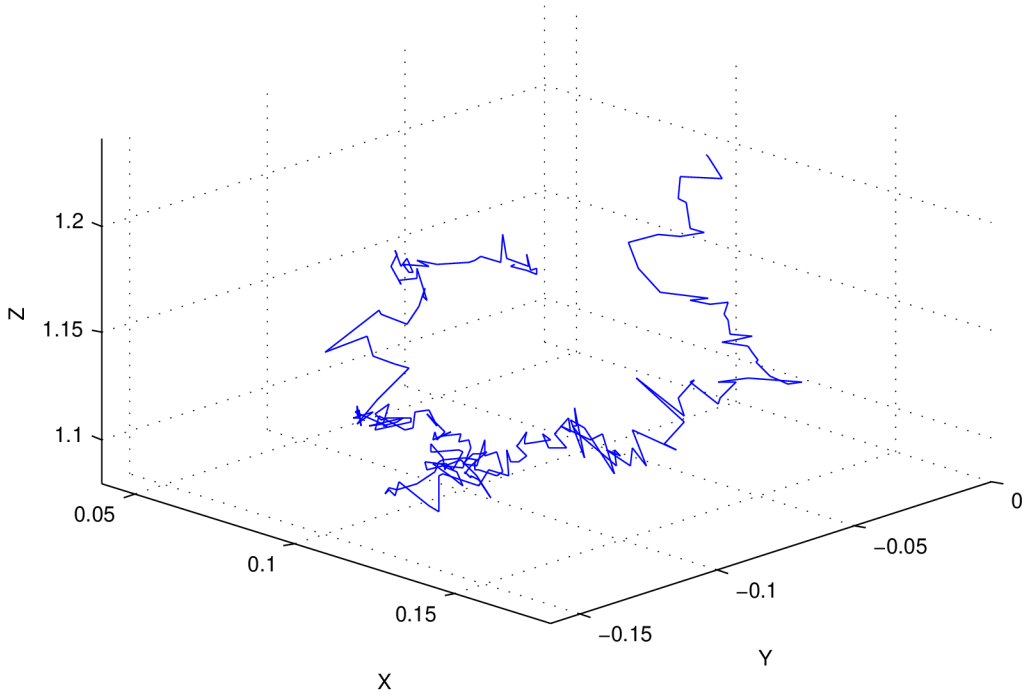
Obrázek 6.8: Příklad sledování objektu v reálném videu



Obrázek 6.9: Relativní poloha kamery vůči automobilu a jeho vývoj v čase



Obrázek 6.10: Relativní úhel kamery vůči automobilu a jeho vývoj v čase



Obrázek 6.11: 3D poloha v čase

Kapitola 7

Závěr

V rámci této práce byly prostudovány metody pro sledování objektů ve videu pomocí statistický metod založených na částicových filtrech. Dále byla prostudována literatura týkající se detekce zájmových bodů s důrazem na metody možné implementovat pomocí grafické karty a vybrán jeden konkrétní algoritmus SUSAN pro implementaci. Vlastní implementace algoritmu byla důkladně optimalizována tak, aby mohl na grafické kartě fungovat bez omezení a efektivně. Představeno bylo vylepšení tohoto algoritmu využívající možnosti texturovacích jednotek GPU a tím i zlepšena jeho odolnost vůči alias efektům, které mohou vznikat díky uniformitě obrazové informace. Představen byl sledovač založený na prohledávání prostoru stavů pomocí částicového filtru. Ten se snažil o minimalizaci účelové funkce. Ta byla definována jako vzdálenost blízkých příznaků, které byly výstupem algoritmu SUSAN. Program velmi dobře fungoval jak pro testovací scénu, tak pro reálnou.

Program má několik nevýhod. První a zásadní je nemožnost běhu ve skutečném čase. Výsledkem profilace bylo zjištění, že největší část výpočtu zabírá vykreslování velké množství scén a nejspíše i režie spojená se spouštěním velkého množství GPU kernelů. Možným řešením by bylo vyčlenit jednu grafickou kartu pro vykreslování scény a druhou pro výpočet SUSAN příznaků. Druhou grafickou kartu jsem však bohužel neměl k dispozici. Další velkou nevýhodou je poměrně velké množství vstupních parametrů programu, které se musí empiricky nastavit tak, aby postihly konkrétní dynamiku modelu. Jinak se často stává to, že renderované modely příliš kmitají kolem správné polohy a nezřídka je výsledkem zcela špatný stav (model se příliš vzdálí a ztratí kontakt s objektem ve videu). Možností by bylo zvýšit počet částic, tak aby lépe pokrývali stavový prostor, to však vede k prodloužení doby výpočtu. Dimezionalita stavového prostoru pracuje v tomto ohledu proti nám. Lineární zvýšení počtu částic vede v 6D prostoru pouze k malému zlepšení pokrytí. Volné parametry modelu je však možno často nastavit velice přesně. Alespoň přibližně vždy víme, jakou dynamiku těleso bude mít.

Možností dalšího vývoje je mnoho. Určitě by pomohl kvalitnější modelu dynamiky sledovaného tělesa (přidání rychlosti a zrychlení) do pohybu tělesa, Nyní je pohyb definován pouze náhodným driftem. To by vedlo k menší šanci na ztrátu objektu ve videu, nepřímo bychom tak mohli snížit počet částic a tím zvýšit rychlost. Další možnost zlepšení je v rychlosti aplikace. Ze zkušenosti se psaním CUDA aplikací často plyne, že i když je kód velice dobře optimalizovaná, stačí i malá změna k několikanásobnému zrychlení. Představená metoda má velký potenciál hlavně s přihlédnutím do budoucna, kdy bude dostatečný výkon na běh podobných aplikací v reálném čase. Oproti běžně používaným metodám je informace o poloze a rotaci tělese důležitým krokem kupředu v oblasti počítačového vidění.

Literatura

[1] Black, M. J.; Jepson, A. D.: A Probabilistic Framework for Matching Temporal Trajectories: Condensation-Based Recognition of Gestures and Expressions. 1998.

[2] Comaniciu, D.; Ramesh, V.; Meer, P.; aj.: Kernel-Based Object Tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 25, 2003: s. 564–577.

[3] Corporation, N.: *NVIDIA CUDA Programming Guide 3.0*. 2009.
URL http://developer.nvidia.com/page/home.html

[4] Deans, M.; Kunz, C.; Sargent, R.; aj.: Combined feature based and shape based visual tracker for robot navigation. In *Aerospace Conference, 2005 IEEE*, March 2005, s. 339–346, doi:10.1109/AERO.2005.1559326.

[5] Doucet, A.; Johansen, A. M.: A Tutorial on Particle Filtering and Smoothing: Fifteen years Later. In *Handbook of Nonlinear Filtering*, editace D. Crisan; B. Rozovsky, Oxford University Press, 2009.
URL http://people.cs.ubc.ca/~arnaud/doucet_johansen_tutorialPF.pdf

[6] Harris, M. J.: Real-Time Cloud Simulation and Rendering. 2003.

[7] Hsieh, M.-h.: Adaptive monte carlo methods for rare event simulation: adaptive monte carlo methods for rare event simulations. In *WSC '02: Proceedings of the 34th conference on Winter simulation*, Winter Simulation Conference, 2002, ISBN 0-7803-7615-3, s. 108–115.

[8] Isard, M.; Blake, A.: CONDENSATION - conditional density propagation for visual tracking. *International Journal of Computer Vision*, ročník 29, 1998: s. 5–28.

[9] Kaněčka, P.: *Vyhledání význačných bodu v rastrovém obraze, diplomová práce*. 2007.
URL http://www.fit.vutbr.cz/study/DP/rpfile.php?id=1354

[10] Koller, D.; Weber, J.; Malik, J.: Robust multiple car tracking with occlusion reasoning. In *In European Conference on Computer Vision*, Springer-Verlag, 1994, s. 189–196.

[11] Lee, A.; Yau, C.; Giles, M. B.; aj.: On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. Jul 2009, 0905.2441.
URL http://arxiv.org/abs/0905.2441

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

[13] NVIDIA: NVIDIA's Next Generation CUDA Compute Architecture: Fermi. 2009, [Online; navštíveno 29-prosince-2009].
URL http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[14] Ojala, T.; Pietikinen, M.; Menp, T.: Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 24, 2002: s. 971–987, ISSN 0162-8828, doi:http://doi.ieeecomputersociety.org/10.1109/TPAMI.2002.1017623.

[15] Rabiner, L. R.: A tutorial on hidden Markov models and selected applications in speech recognition. 1990: s. 267–296.

[16] Schmid, C.; Mohr, R.; Bauckhage, C.: Evaluation of Interest Point Detectors. *Int. J. Comput. Vision*, ročník 37, June 2000: s. 151–172, ISSN 0920-5691, doi:10.1023/A:1008199403446.
URL http://portal.acm.org/citation.cfm?id=350676.350678

[17] Smith, S. M.; Brady, J. M.: SUSAN–A New Approach to Low Level Image Processing. *Int. J. Comput. Vision*, ročník 23, May 1997: s. 45–78, ISSN 0920-5691, doi:10.1023/A:1007963824710.
URL http://portal.acm.org/citation.cfm?id=258049.258056

[18] Tuytelaars, T.; Mikolajczyk, K.: Local invariant feature detectors: a survey. *Found. Trends. Comput. Graph. Vis.*, ročník 3, July 2008: s. 177–280, ISSN 1572-2740, doi:10.1561/0600000017.
URL http://portal.acm.org/citation.cfm?id=1391081.1391082

[19] Wikipedia: Motion compensation — Wikipedia, The Free Encyclopedia. 2009, [Online; navštíveno 29-prosince-2009].
URL http://en.wikipedia.org/wiki/Motion_compensation

[20] Wikipedia: Project Natal — Wikipedia, The Free Encyclopedia. 2009, [Online; navštíveno 29-prosince-2009].
URL http://en.wikipedia.org/wiki/Project_Natal

[21] Wikipedia: Video tracking — Wikipedia, The Free Encyclopedia. 2009, [Online; navštíveno 29-prosince-2009].
URL http://en.wikipedia.org/wiki/Video_tracking

[22] Yan, W.; Forsyth, D.: Learning the Behavior of Users in a Public Space through Video Tracking. In *Application of Computer Vision, 2005. WACV/MOTIONS '05 Volume 1. Seventh IEEE Workshops on*, ročník 1, Jan. 2005, s. 370–377.

[23] Young, S.; Kershaw, D.; Odell, J.; aj.: *The HTK Book Version 3.0*. Cambridge University Press, 2000.

[24] Zdeněk, H.: *Sledování pohybujících se objektů ve video sekvenci, diplomová práce.* 2007.
URL http://www.fit.vutbr.cz/study/DP/rpfile.php?id=10469

[25] Zein, A.; Mccreath, E.; Rendell, A.; aj.: Performance Evaluation of the NVIDIA GeForce 8800 GTX GPU for Machine Learning. In *ICCS '08: Proceedings of the 8th international conference on Computational Science, Part I*, Berlin, Heidelberg: Springer-Verlag, 2008, ISBN 978-3-540-69383-3, s. 466–475, doi:http://dx.doi.org/10.1007/978-3-540-69384-0˙52.

Dodatek A

Obsah DVD

Přiložený DVD nosič obsahuje zdrojové kódy aplikace, přeloženou aplikaci, včetně všech potřebných dynamických knihoven a samozřejmě pdf a tex verzi této zprávy. Přiloženy jsou i všechna doprovodná videa.

Obsah jednotlivých adresářů:

- src - Zdrojové kódy aplikace, VS 2008 projekt
- doc - Pdf a tex verze této zprávy
- video - Testovací videa
- models - Využité 3D modely