Homography estimation for image stitching using RANSAC on GPU

Bc. Juraj Marticek
xmarti97@vutbr.cz

Bc. Peter Urgoš
xurgos00@vutbr.cz

Abstrakt

Tento projekt skúma integráciu OpenCL s výpočtom GPU pre efektívny výpočet algoritmu RANSAC v kontexte spájania obrázkov (image stitching). Naše zameranie sa sústreďuje na proces odhadu homografií, ktorý je používaný aplikáciách, ako je panoramatická konštrukcia a mozaika obrazu. Projekt postupuje v rôznych fázach, počnúc počiatočnou sekvenčnou implementáciou, výberom vhodných algoritmov a postupne smerom k využívaniu GPU pre optimalizáciu rýchlosti výpočtu.

Úvod

Image stitching je dôležitý proces v počítačovom videní, ktorý zahŕňa zarovnanie a kombináciu viacerých obrázkov na vyprodukovanie panorámy vo vysokom rozlíšení, alebo aj kompozitného obrázka. Hlavná úloha spočíva v presnom odhadnutí **homografie**[1] medzi párom obrázkov. Rozhodli sme sa potýkať tento problém implementovaním nami navrhnutej pipeline zaŕňajúcej detekovanie príznakov (feature detection), popis týchto príznakov (feature detection), matching kľúčových bodov (keypoint matching), a odhad homografie, pri ktorom sme sa zamerali na **GPU akceleráciu RANSAC algoritmu**, ktorý v sebe obsahuje zložitý výpočet v podobe odhadnutia homografie pre menší počet kľúčových bodov (viz. popis implementácie).

Metodika

V tejto sekcii si popíšeme postup tvorenia vyššie popisovanej pipeline pre hľadanie homografie medzi obrázkami. Pre každú jej časť bolo potrebné rozlíšiť a správne vybrať algoritmus, ktorý je najviac vhodný pre náš účel.

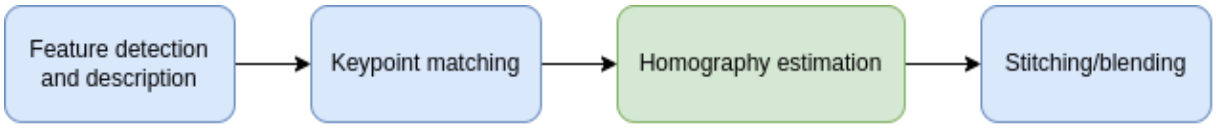

Figure 1: „Pipeline“ pre vytváranie mozaiky obrázkov

Detekcia a popis príznakov

Pri hľadaní príznakov medzi obrázkami sa snažíme hľadať charakteristické rysy, ktoré prejavia na obidvoch obrázkoch na rovnakých miestach. Obrázky sa môžu medzi sebou líšiť rôznymi polohami, pozíciami fotoaparátu, taktiež sveteľnosťou, rozmazaním a farbami. Preto je táto úloha pomerne zložitá, a existuje pre ňu mnoho riešení. Ak hovoríme o **detekcii príznakov (feature detection)**, tak máme dostupný napríklad Harris Corner Detector, ktorý jednoducho autokoreláciou zisťuje rohy, ale taktiež zložitejšie algoritmy ako napríklad FAST[2], SIFT[3] a SURF[4].

Pri našej úlohe nám ale nájdenie iba príznakov na obidvoch obrázkoch nepostačuje, pretože v následujúcom kroku budeme musieť vedieť, ktoré príznaky sú si medzi sebou podobné medzi obrázkami. To nám zaručuje **popis príznakov (feature description)**. Keďže nám napríklad SIFT alebo SURF vytvára aj popisy príznakov, tak je možné ich použitie. My sme sa ale rozhodli pre **ORB (Oriented FAST and Rotated BRIEF)**[5], ktorý nám dokáže detekovať kľúčové body, a taktiež k nim vytvoriť deskriptory, ktoré využijeme v následujúcom kroku. Výber tohto algoritmu podporilo to, že našou úlohou je optimalizácia rýchlosti riešenia.

Pri výbere týchto algoritmov je dôležité zohľadiť niekoľko aspektov:

- **Rýchlosť vs. presnosť** - SIFT a SURF nám poskytujú vyššiu presnosť, ale za cenu potreby vyššieho výpočetného výkonu.
- **Invariácia ku transformácii** - voči rotácii a posunu SIFT a SURF sú robustnešie keď sa jedná o väčšie posuny a rotácie obrázku, no v našom prípade sa rotácia pohybuje často v menších mierach.

Matching kľúčových bodov

Po tom ako sme našli kľúčové body na obidvoch obrázkoch, tak potrebujeme vytvoriť dvojice kľúčových bodov medzi týmito obrázkami, ktoré popisujú rovnaké miesto (resp. rovnaký objekt na obrázku). Na túto úlohu sme si vybrali **Brute-Force Matcher**, ktorý je pomalší ako iné alternatívy (FLANN[6]). Hrubou silou prechádza každú dvojicu bodov, no za to nám prináša väčšie možnosti pri implementácií paralelizácie na GPU. Keďže máme binárny deskriptor príznakov (ORB), tak používame Hammingovu vzdialenosť ako vzdialenostnú metriku pre náš matcher.

V tomto bode máme možnosť filtrovať výsledky z tohto matcheru, aby sme dosiahli lepších výsledkov pri hľadaní homografie. Keďže sa potenciálny spoločný región medzi obrázkami nachádza len v jednej oblasti, tak vieme očakávať, že väčšina bodov bude mať medzi sebou podobnú vzdialenosť, a ostatné vieme vyfiltrovať. V našej implementácii takáto filtrácia neprinášala významné zisky, a preto sme sa ju rozhodli netestovať v každom prípade.

Odhad homografie

Táto úloha spočíva v zistení transformácie (homografie) medzi bodmi dvoch potenciálne prelínajúcich sa obrázkov. Technikou, ktorú sme si zvolili mi, je použitie algoritmu **Random Sample Consensus (RANSAC)**[7] v spojení s **Direct Linear Transform (DLT)** pre odhad tejto matice.

Sekvenčná implementácia

Ako prvé sme sa rozhodli implementovať algoritmus RANSAC v sekvenčnej verzii, aby sme následne vedeli porovnať rýchlosť jednotlivých riešení. Taktiež máme dostupnú variantu, ktorá je implementovaná v knižnici OpenCV[8], ktorú taktiež porovnávame vo výsledkoch.

Paralelizácia v OpenCL

Pomocou knižnice OpenCL, v ktorej si vieme vytvárať kernely, ktoré sa následne spúšťajú na grafickej karte, sme vedeli vylepšiť rýchlosť výpočtu vďaka využitiu dostupných zdrojov GPU.

Implementácia riešenia a meranie rýchlosti

Táto časť sa zameriava vysvetlením implementácie RANSAC prístupu v spojení s DLT algoritmom na určenie homografie medzi fotkami.

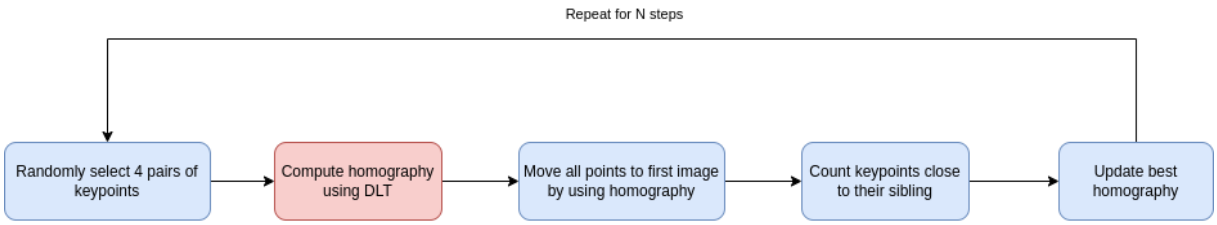

Figure 2: Diagram algoritmu RANSAC spolu s DLT

RANSAC s DLT

Random Sample Consensus (RANSAC) je algoritmus na určovanie modelu pomocou výberu náhodných bodov, a následného výpočtu tzv. inliers - čo znamenajú body náležiace do modelu. V našom prostredí to znamená, že náhodne vyberieme 4 dvojice kľúčových bodov z obrázkov, a následne pre nich vypočítame homografiu pomocou Direct Linear Transform (DLT). Keď máme homografiu medzi

týmito štyroma dvojicami, tak vieme následne prejsť všetky ostatné kľúčové body z druhého obrázka, posunúť ich do toho prvého, a pomocou jednoduchej euklidovskej vzdialenosti porovnať, či je daný bod dostatočne blízko svojmu páru, alebo nie. Ak je, tak inkrementujeme počet inliers. Tento počet je naša metrika pre vyberanie najlepšej homografie spomedzi `n` náhodných.

Ak by sme algoritmus RANSAC nepoužili, museli by sme počítať homografiu pre všetky kľúčové body, ktorých je počet síce závislý od typu a parametrov feature detektoru, ale je veľmi náročný na výpočet, a nereálny na praktické využitie.

V priložených zdrojových kódoch sa nachádza trieda `Homography`, ktorá nám slúži ako hlavný prostriedok pre výpočet homografie `H`. Táto trieda v konštruktore akceptuje funkciu, ktorou si určujeme zvolený spôsob výpočtu DLT. **Dostupné možnosti sú**:

- `Homography::CV` - použitie zabudovanej metódy OpenCV, ktorá používa pre výpočet vlastnú implementáciu DLT za pomocou metódy najmenších štvorcov pre odhad homografie.
- `Homography::DLT` - naša sekvenčná implementácia DLT za použitia OpenCV SVD.
- `Homography::eigenDLT` - naša sekvenčná implementácia DLT za použitia SVD z `Eigen` knižnice.
- `Homography::GPU` - naša implementácia za využitia výpočtu na grafickom procesore.

Okrem vyššie spomenutej triedy, zdrojové kódy obsahujú aj implementácie celej pipeline pre hľadanie homografie, a nástroje pre vizualizáciu jednotlivých krokov pred výpočtom homografie.

Meranie výkonnosti

Meranie výkonnosti je zabudované do triedy `Homography`, ktorá pri volaní hlavnej funkcie `Homography()::find()` vypočíta dĺžku výpočtu RANSAC s DLT, a uloží to do členskej triedy `Performance` - `perf`, ktorá nám taktiež drží informácie o nájdených bodoch, alebo počtu inliers.

Práve vďaka tomuto vieme spustit `Homography()::find()` na množstve dát, a mať výsledky výkonnosti pri zmene rôznych parametrov. Jedným z takých parametrov je napríklad maximálny počet iterácii RANSAC algoritmu. Vizualizáciu a porovnanie dát môžeme vidieť v následujúcich grafoch.

Experimenty

V experimentoch sme porovnávali hlavne časovú náročnosť na vypočítanie rovnakej úlohy medzi viacerými verziami `Homography` „finderov“. Pri použití väčšieho datasetu fotografií sa medzi fotografiami rovnakých rozlíšení rýchlostne nelíšili, a preto sme sa rozhodli ladiť parametre. Vo výsledných grafoch a prílohe môžeme vidieť merané parametre. Taktiež sme sa rozhodli vyhodnotiť aj presnosť nášho riešenia, keďže určitá presnosť je potrebná, aby náš model bol použiteľný. Presnosť sme vyhodnocovali porovnaním voči referenčnému riešeniu v `OpenCV`, kde sme vybrali súbor bodov, ktoré sme posunuli referenčnou homografiou, a taktiež az homografiou našou získanou. Následne sme porovnávali priemernú vzdialenosť medzi korešpondujúcimi dvojicami.

Zhodnotenie výsledkov

Výkon sekvenčný vs. paralelný

V počiatočnej fázi sme použili sekvenčný algoritmus ako baseline riešenie pre ladenie, a vylepšovanie paralelného algoritmu. Bohužial, kvôli komplexite výpočtu SVD v OpenCL knižnici (taktiež veľmi náročné aj sekvenčne), sa nám nepodarilo plnohodnotne využiť schopnosti grafickej karty, a v tomto aspekte je ešte priestor na vylepšenie doterajšieho riešenia.

Presnosť riešenia

Presnosť nášho riešenia sme porovnávali s referenčným riešením `cv::findHomography()`. Obidva riešenia nám poskytujú homografickú maticu, ktorú sme vedeli následne porovnať, a tak veľmi spoľahlivo určit presnosť nášho riešenia voči referenčnému.

Škálovateľnosť

Pri zväčšovaní počtu bodov, ktoré vstupovali do algoritmu sme nezaznamenali výrazný zisk zo škálovania, hlavne kvôli vyššie zmienenému nedostatku využitia GPU.

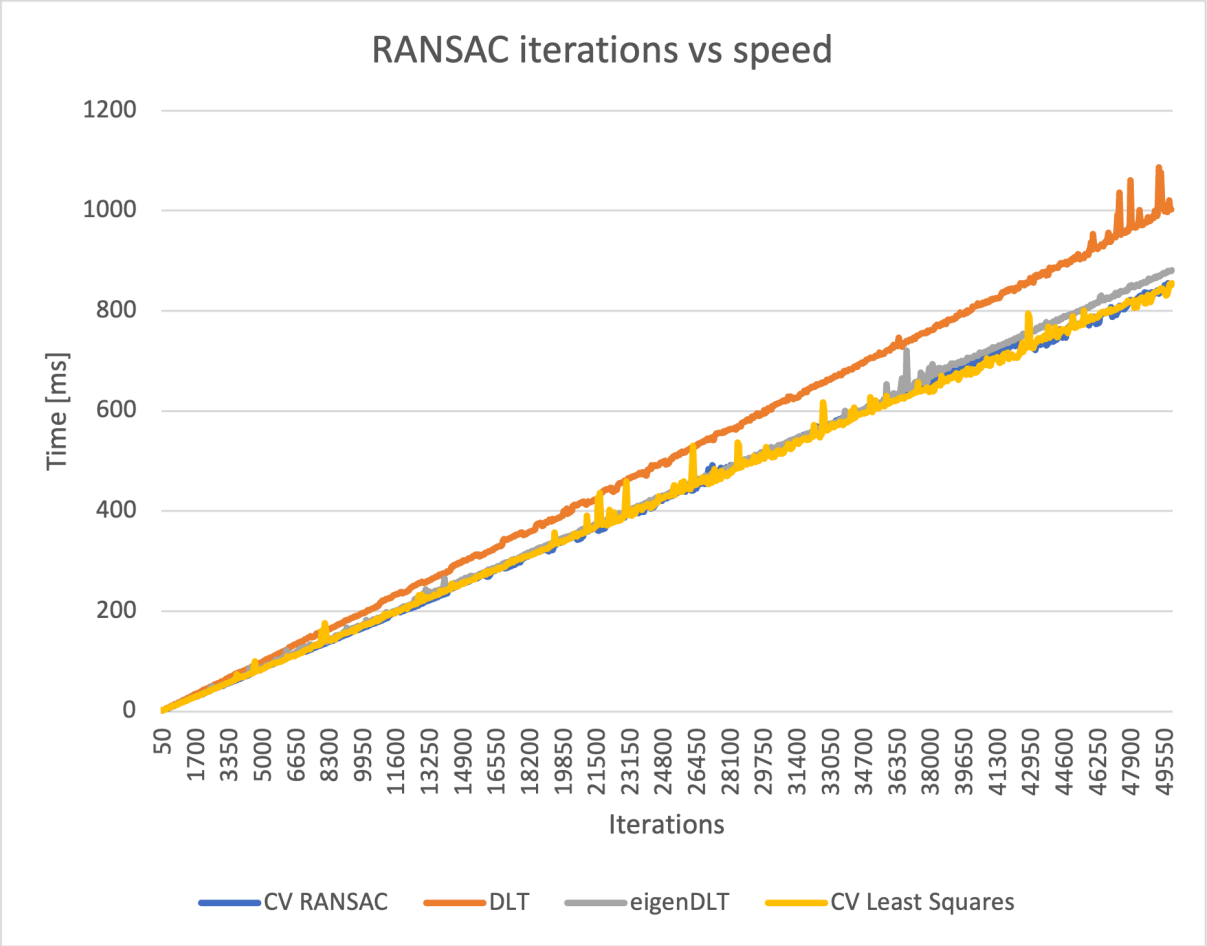

Figure 3: Porovnanie rýchlosti jednotlivých algoritmov pre výpočet homografie v závislosti od počtu iterácii RANSAC

Záver

Riešenie úspešne demonštruje metódy pre hľadanie homografie pomocou RANSAC algoritmu, ako aj sekvenčné, taktiež aj dátovo paralelné na grafickej karte. Integrácia grafickej karty do výpočtu môže priniesť mnoho priestoru pre urýchľovanie výpočtu. Vďaka možnosti pre použitie omnoho viac kľúčových bodov, môžeme zjednodušiť kroky pred samotným výpočtom homografie, čo nám môže zaručiť ešte vyšší zisk v celkovej rýchlosti výpočtu. Zrýchlenie, ktoré sme očakávali sa nám nepodarilo dosiahnuť v plnej miere, a vďaka návrhom, ktoré sme popisovali vyššie, by sme vedeli tieto ciele dosiahnuť.

V budúcnosti by sme sa radi zamerali na väčšiu utilizáciu hardvéru grafickej karty, hlavne pri výpočte SVD, čo sa ukázalo ako najťažšia úloha spomedzi ostatných častí celej práce.

Bibliografia

[1] R. I. Hartley a A. Zisserman, *Multiple View Geometry in Computer Vision*. v Cambridge Computer Science Texts. Cambridge University Press, 2004.

[2] E. Rosten a T. Drummond, "Machine learning for high-speed corner detection", v *European Conference on Computer Vision*, 2006.

[3] D. G. Lowe, "Distinctive image features from scale-invariant keypoints", v *International journal of computer vision*, Springer, 2004.

[4] H. Bay, T. Tuytelaars, a L. Van Gool, "SURF: Speeded up robust features", v *European conference on computer vision*, 2006.

[5] E. Rublee, V. Rabaud, K. Konolige, a G. Bradski, "ORB: An efficient alternative to SIFT or SURF", v *2011 International Conference on Computer Vision*, 2011. [Online]. Available at: https://ieeexplore.ieee.org/document/6126544

[6] Muja, Marius and Lowe, David G, "FLANN - Fast Library for Approximate Nearest Neighbors". [Online]. Available at: https://github.com/mariusmuja/flann

[7] M. A. Fischler a R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography", *Communications of the ACM*, roč. 24, č. 6, 1981, [Online]. Available at: https://dl.acm.org/doi/10.1145/358669.358692

[8] G. Bradski, A. Kaehler, a others, "OpenCV: Open Source Computer Vision Library". 2023.

Prílohy

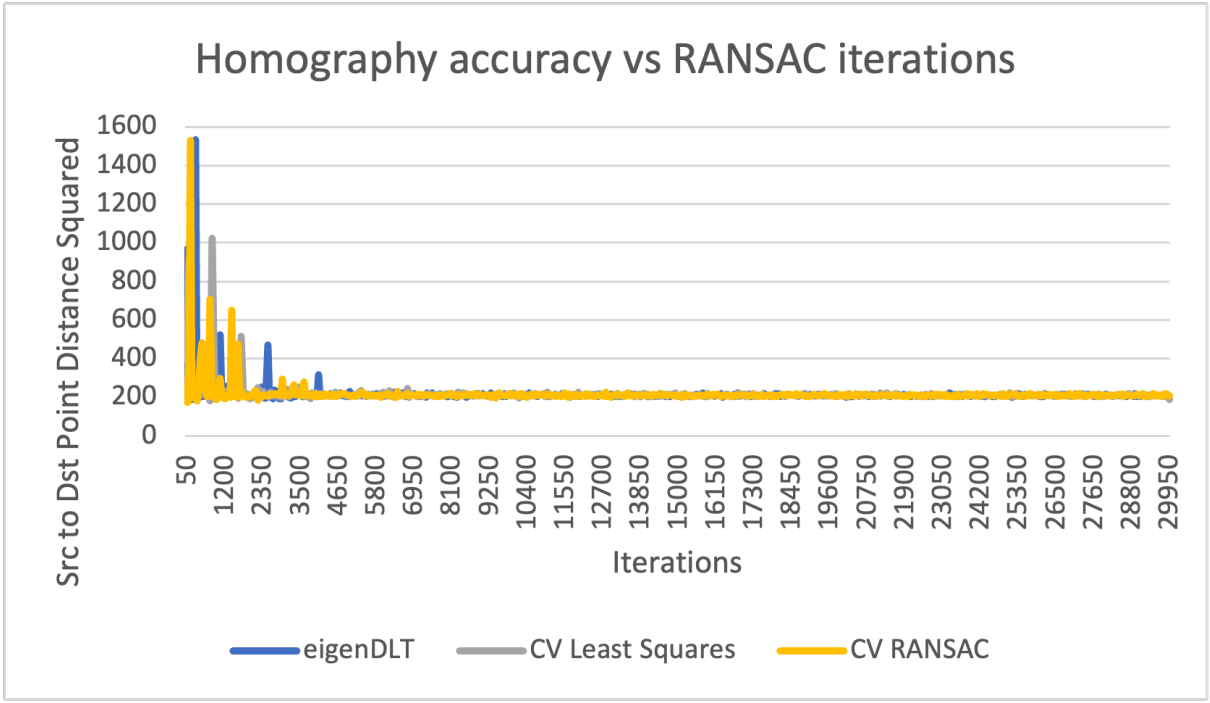

Figure 4: Porovnanie presnosti našej implementácie a referenčnej

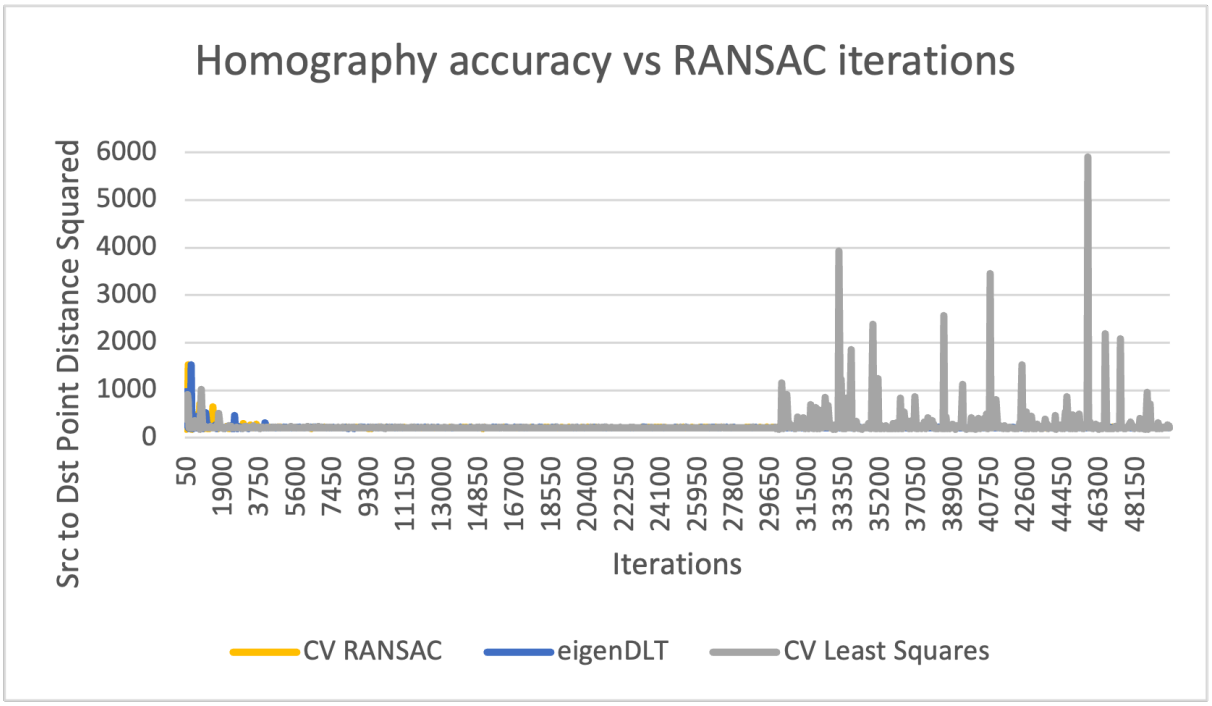

Figure 5: Porovnanie presnosti našej implementácie a referenčnej