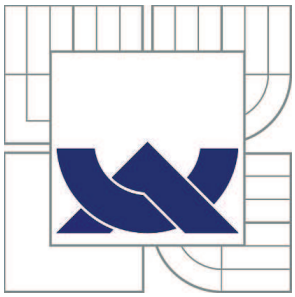
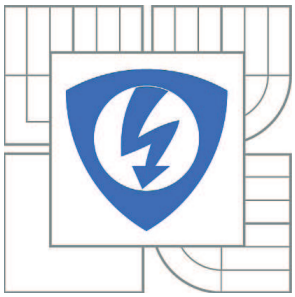


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

KALIBRACE KAMERY POMOCÍ EVOLUČNÍCH ALGORITMŮ

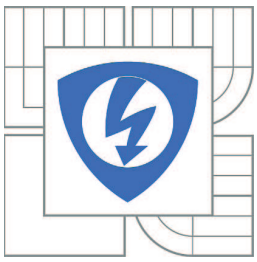
CAMERA CALIBRATION BY EVOLUTIONARY ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN KLEČKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ BABINEC
SUPERVISOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Jan Klečka	***ID:***	126769
Ročník:	3	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Kalibrace kamery pomocí evolučních algoritmů

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s problematikou kalibrace kamer pro rekonstrukci 3D tvaru objektů.
2. Na základě rešerše v oblasti umělé inteligence zvolte evoluční metodu vhodnou pro stanovení hodnot vnitřních i vnějších parametrů kamery (poloha, natočení v prostoru,…).
3. V jazyce C/C++ implementujte knihovnu pro práci se zvoleným evolučním algoritmem.
4. Zhodnoťte možnosti realizace evolučních výpočtů s využitím procesoru grafické karty a své závěry ověřte praktickou implementací.
5. Dokončete a odlaďte implementaci vámi zvolené evoluční metody pro řešení úlohy kalibrace kamery.
6. Funkci vytvořených algoritmů ověřte na syntetických i reálných datech. Vyhodnoťte a prezentujte dosažené výsledky.

DOPORUČENÁ LITERATURA:

[1] KRAUS K.: Photogrammetrie 1 und 2, Ummler / Bonn, 1996, ISBN: 3110177080
[2] SONKA M., HLAVAC V., BOYLE R.: Image Processing, Analysis and Machine Vision. 3rd edition. Toronto : Thomson, 2008. 829 s. ISBN 978-0-495-08252-1.
[3] MAŘÍK V.: Umělá inteligence 3, Praha: Academia, 2001, ISBN: 80-200-0472-6
[4] MAŘÍK V.: Umělá inteligence 4, Praha: Academia, 2003, ISBN: 80-200-1044-0

Termín zadání:	6.2.2012	***Termín odevzdání:***	28.5.2012

Vedoucí práce: Ing. Tomáš Babinec
Konzultanti bakalářské práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

Abstrakt

Práce se zabývá vývojem a zhodnocením programu pro výpočet vnitřních a vnějších parametrů kamery, pomocí evolučních algoritmů resp. pomocí algoritmu diferenciální evoluce. Zároveň popisuje možnost využití grafické karty pro paralelní výpočet.

Klíčová slova

Kalibrace kamery, diferenciální evoluce, evoluční algoritmy, grafická karta, paralelní zpracování programu

Abstract

This paper describes the possibility of using evolutionary algorithms (specifically the differential evolution) to figure out interior and exterior parameters of camera. It is an easy and an effective way to solve this problem. Also describe possibility of using graphics processor unit to parallel computing.

Keywords

Camera calibration, evolutionary algorithms, differential evolution, parallel computing

Bibliografická citace:

KLEČKA, J. *Kalibrace kamery pomocí evolučních algoritmů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 40s. Vedoucí bakalářské práce byl Ing. Tomáš Babinec.

Prohlášení

„Prohlašuji, že svou bakalářskou práci na téma „Kalibrace kamery pomocí evolučních algoritmů“ jsem vypracoval samostatně, pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **28. května 2012**

……
podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Tomáši Babinci za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **28. května 2012**

……
podpis autora

Obsah

1 ÚVOD

Předmětem této práce bylo vytvořit metodu resp. program, který dokáže pomocí evolučních algoritmů vypočítat vnitřní a vnější parametry kamery. Řešit kalibraci kamery touto metodou je v dnešní době velice výhodné. Dále jsem měl zhodnotit a praktickou realizací ověřit možnosti využití grafické karty k akceleraci výpočtu jeho paralelizací.

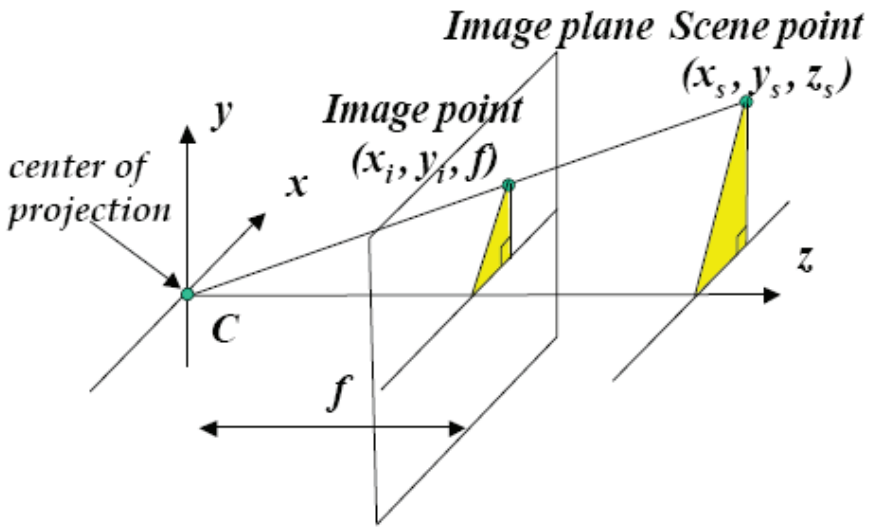
Má práce by se dala rozdělit na dvě části: teoretickou a praktickou. Za teoretickou můžeme označit první tři kapitoly (kapitolu 2,3 a 4). V této části se pokusím shrnout veškeré důležité teoretické poznatky, které jsem musel nastudovat pro zvládnutí tohoto problému. V praktické části mé práce se potom zaměřím na můj program. Pokusím se zhodnotit přesnost, shrnu své poznatky z oblasti paralelních výpočtů na GPU a prezentuji jádro mého programu.

2 KALIBRACE KAMERY

Kalibrace kamery je zjištění vnitřních a vnějších parametrů kamery. Je to velice důležitý proces pro jakékoli měření pomocí kamery. Význam tohoto procesu se pokusím vysvětlit na zjednodušeném modelu kamery, protože kalibrace kamery by se dala jinými slovy také vysvětlit jako nastavení parametrů modelu kamery tak, aby měl při stejných vstupních datech stejný výstupní obraz.

2.1 Matematický model kamery

Model kamery nám určuje na který pixel dvourozměrného obrazu se nám promítnou body ze snímaného trojrozměrného prostoru. Grafické znázornění tohoto modelu můžeme vidět na Obr 1.



Obr. 1: Grafické znázornění modelu kamery převzaté z [3]

C na obrázku značí ohnisko kamery, písmeno f je ohnisková vzdálenost, souřadnice s indexem i jsou obrazové souřadnice tedy souřadnice pixelu na obrázku a souřadnice s indexem s jsou prostorové souřadnice. Je jasně vidět, že převod z prostorových do obrazových souřadnic je možné snadno a jednoznačně vypočítat. Opačně dostáváme nekonečně mnoho řešení.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_S \\ Y_S \\ Z_S \end{bmatrix} \tag{2.1}$$

Kde potom:

$$x_i = \frac{x}{z}$$

$$y_i = \frac{y}{z}$$

Rovnice 2.1 je pouze pomocí matic zapsaný model kamery znázorněný na Obr 1. Tento model využívající ke své funkci jen podobnost trojúhelníků je sice funkčním modelem, ale pro kalibraci kamery se nehodí. Důvodů, proč se nehodí, je hned několik. Např.: Výsledné souřadnice promítnutých bodů jsou v prostorových souřadnicích a nikoliv v souřadnicích pixelových. Zároveň neumožňuje žádný posun či rotaci mezi kamerou a sledovaným objektem a veškeré souřadnice sledovaných objektů by bylo nutno brát jako vzdálenost od ohniska promítání.

Vzhledem k těmto důvodům jsem pro kalibraci kamery použil principielně stejný avšak trošku složitější model, který matematicky popisuje rovnice 2.2.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{f}{p_x} & 0 & c_x & 0 \\ 0 & \frac{f}{p_y} & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 000 & 1 \end{bmatrix} \begin{bmatrix} X_S \\ Y_S \\ Z_S \\ 1 \end{bmatrix} \tag{2.2}$$

Kde potom $x_i = \frac{x}{z}$, $y_i = \frac{y}{z}$

Kde: p_x, p_y jsou velikosti pixelu, R je matice rotace, T matice translace a $\lfloor c_x, c_y \rfloor$ jsou souřadnice středu obrázku.

U tohoto modelu už vidíme, že výsledek je v pixelových souřadnicích, jelikož ohnisko f je podělené velikostí pixelu v dané ose (protože pixel obecně vzato nemusí být čtvercový). Souřadnice středu obrázku $\lfloor c_x, c_y \rfloor$ v modelu plní funkci jistého offsetu. Resp. zajišťují, aby byly body, které leží v prostoru na ose z, zobrazeny na střed obrazu. Dále model obsahuje matici transformací homogenních souřadnic pro translaci a rotaci ve třech osách.

Také zde můžeme hezky vidět, již výše zmíněné rozdělení na vnitřní a vnější parametry kamery. Matice v součinu úplně vlevo představuje projekční matici a obsahuje vnitřní parametry kamery. Prostřední matice obsahuje vnitřní parametry kamery, tedy polohu v prostoru.

Samozřejmě existuje ještě celá řada dalších možností, jak model kamery rozšířit, aby počítal s více a více vlivy, např.: je možné počítat s pixely, které nejsou ani obdélníkové ale zkosené nebo je možné vzít v potaz i možnosti zkreslení. Tyto modely jsem ve své práci nepoužil.

2.2 Transformace homogenních souřadnic

Běžně se používají při přípravě scény v počítačové grafice nebo v robotice pro plánování pohybu robota.

Jsou to transformace, které se aplikují na bod v prostoru za účelem změny polohy tohoto bodu resp. spíše transformují naše vnímání prostoru kolem bodu.

Existuje mnoho druhů těchto transformací. Jde o např.: Změnu měřítka, zkosení, rotaci, translaci atd. Z těchto transformací se budu zabývat pouze translací a rotací, protože jiné ve své práci nepoužívám.

Všechny tyto transformace jsou zapsané pomocí matic. Díky tomuto zápis i složitých kombinací transformací může být pro člověka snadno srozumitelný. Jejich použití je velmi snadné. Přesně tak jak vidíme na rovnicovém zápisu modelu kamery 2.2. Vezmeme vektor souřadnic bodu, na jeho konec přidáme standardně hodnotu 1 (jiná hodnota se používá při změně měřítka nebo třeba chceme-li výsledek v jiných jednotkách) a tímto vektorem vynásobíme transformační matice.

2.2.1 Rotace

Bereme-li v úvahu trojrozměrný prostor, existují přirozeně při elementární rotace a to podle každé že tří os. Tyto tři elementární transformace vidíme na rovnicích 2.3, 2.4, 2.5.

$$R_X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.3}$$

$$R_Y = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

$$R_Z = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.5}$$

Rotaci ve třech osách tedy můžeme realizovat pomocí těchto tří elementární transformací a to pouhým zařazením matic za sebe jinak řečeno budeme rotovat napřed podle jedné, druhé a nakonec podle třetí osy.

Ze znalosti násobení matic ale víme, že tato operace není komutativní, tedy záleží na pořadí matic. Pro přesné určení modelu kamery bude tedy nutné určit i pořadí, ve kterém aplikujeme rotační matice.

Mnou zvolené pořadí je: Nejprve rotuji podle osy X, následně podle Y a nakonec podle Z. Samozřejmě by šlo použít i jakékoliv jiné pořadí, jen by kalibrace kamery dávala obecně jiné výsledky.

2.2.2 Translace

Translace je ještě jednodušší jak rotace. Translační matici umisťuji jako poslední z transformačních matic, přirozeně by šla použít i jako první, ale můj subjektivní názor je, že je přirozenější a snáze představitelné rotovat předmět kolem jeho os a poté ho umístit na požadované místo. Translační matice je popsána rovnicí 2.6.

$$T = \begin{pmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

2.3 Jiné metody kalibrace kamery

Jak pro kalibraci kamery tak pro měření pomocí kamery existuje celá řada jiných metod než je ta, kterou využívám já. Nemám žádné data pomocí, kterých bych mohl srovnat přesnost nebo rychlost těchto metod s mojí. Jediné co můžu srovnat na první pohled je složitost, protože při využití evolučních algoritmů je problém, narozdíl od těchto metod, velmi názorný a srozumitelný, není třeba studovat nic složitého.

Existují tři velké skupiny metod využívaných pro měření pomocí kamery. Každá určená pro jiné vzdálenosti měření. Pro nejmenší vzdálenosti je tu interferometrie, která využívá fázového posunu světla. Dále jsou tu metody využívající triangulaci a tedy geometrické vlastnosti pozorovaných objektů. A pro největší vzdálenosti jsou zde metody měřící dobu letu světla.

Má metoda je co do oblasti použití a částečně i funkce nejvíce podobná prostřední skupině metod využívající triangulaci.

V této množině je mnoho různých metod, mimo jiné i například měření pomocí teodolitu, metody nejpodobnější té mojí jsou ze skupiny nazvané „pasivní triangulace“. Využívají obrazů ke kalibraci kamery, měření a rekonstrukci zobrazených objektů. Všechny tyto metody ale využívají více pohledů na objekt, ať už pomocí více kamer nebo pomocí přesunování jedné, při jediném pohledu nemůžou fungovat. [7]

Více informací o této problematice je možné najít v [7].

3 EVOLUČNÍ ALGORITMY

3.1 Stručná historie evolučních algoritmů

První pokusy byly uskutečněny v první až druhé polovině 60.let. Úplně první zaznamenané použití bylo v Berlíně třemi studenty, kteří se zabývali konstrukcí převodovek. Dalším příkladem je evoluční programování publikované v roce 1966 (Fogel a kol.). V roce 1975 americký teoretický biolog John Holland pokládá ve své knize základ genetickým algoritmům. V roce 1989 kniha Davida Goldberga nahlíží na genetické algoritmy z technického pohledu jako na obecně aplikovatelnou techniku na širokou třídu úloh. [1]

3.2 Charakteristika evol. algoritmů

Evoluční algoritmy se používají pro prohledávání určitého prostoru. Většinou prostoru rozsáhlého – takového, který nejde prozkoumat úplným prohledáním.

Jsou to scholastické algoritmy. Jejichž prohledávací schopnost je posílena modelování genetické dědičnosti a Darwinovským zápasem o přežití.

Adaptace musí být „výhodná“ ve smyslu nějakého kritéria – v přírodě např. množství potravy. Z toho plyne podmínka že kvalitu jedince musí jít vždy určit.

Tradiční optimalizační postupy vycházejí z vhodného (nebo i náhodně zvoleného) počátečního odhadu řešení, které se iterativně vylepšují. Evoluční algoritmy naproti tomu pracují s celou množinou možných řešení tzv. populací. Vývojový čas běží zpravidla v diskrétních vývojových krocích a proto hovoříme o posloupnosti populací jako o generacích.

Každý jedinec populace je ohodnocen svou mírou kvality. Touto kvalitou může být pro technické použití téměř libovolné ohodnocení jedince.

Pro tyto vlastnosti se evoluční algoritmy používají často jako optimalizační metody.[1][4]

3.3 Diferenciální evoluce

3.3.1 Historie diferenciální evoluce

Poměrně nový typ evolučního algoritmu založený na tzv. genetickém žíhání vyvinutým v roce 1994 K.V. Pricem, který následně začal žíhání měnit z binární do dekadické reprezentace. Krátce na to byla vyvinuta tzv. diferenciální mutace. Kombinací genetického žíhání a diferenciální mutace vzniká to co dnes označujeme jako diferenciální evoluce. Později byl původní složitý název tohoto algoritmu zkrácen na diferenciální evoluci a v roce 1997 byl publikován (Stron a Price). Můžeme najít podobnost s genetickým algoritmy, jelikož genetické žíhání bylo původně jejich součástí. Podobnost je v tom, že nový jedinec je tvořen pomocí rodičů avšak na rozdíl od genetických algoritmů jsou 4.[2]

3.3.2 Vlastnosti a oblast použití

Oblast použití diferenciální evoluce je poměrně rozsáhlá. Na rozdíl od většiny evolučních algoritmů, které jsou klasicky schopné pracovat jen s omezenou množinou problémů (např.: Ant Colony Optimalization, který je v podstatě předurčen na řešení problémů s hledáním optimální cesty), lze prohlásit, že diferenciální evoluce je velmi výkonným algoritmem vhodným pro globální optimalizaci. Může být použita téměř na jakoukoli funkci, která vrací hodnotu, díky níž může být určena kvalita jedince. [2]

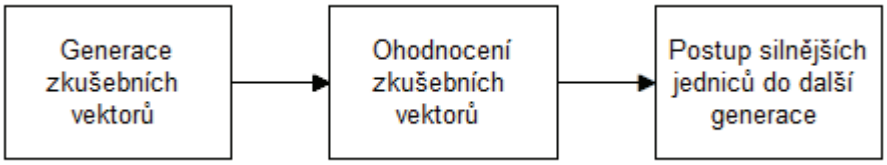
Vlastnosti diferenciální evoluce [2]:

- Jednoduchost .. Algoritmus je velmi jednoduchý
- Hybridnost čísel .. Argumenty fitness funkce mohou být téměř libovolného tipu.
- Používání dekadických čísel .. Pro účely mutace není třeba převod do binární soustavy
- Rychlost
- Nezávislost křížení na kvalitě rodičů .. To je významné z hlediska hledaného globálního extrému funkce. Díky této vlastnosti je snížena pravděpodobnost zaseknutí v lok. extrému.
- Částečná schopnost nalézt extrém .. Algoritmus pracuje i s problémy tipu „Jehla v kupce sena“ tedy funkce je plochá a extrém je jen úzkou dírkou. Ale nalezení tohoto extrému je zpravidla dílem náhody.
- Schopnost dát vícenásobné řešení .. Pokud má funkce více lokálních extrémů algoritmus by je měl najít v takovém případě je ovšem nutno sledovat celou populaci řešení a ne jen nejlepšího jedince.
- Účinnost při řešení nelineárních problémů s ohraničením.

3.3.3 Princip diferenciální evoluce

Diferenciální evoluce má několik základních parametrů:

- CV – práh křížení, pravděpodobnost že bude parametr ze zkušebního vektoru zaměněn za parametr ze šumového vektoru.
- NP – počet jedinců v populaci
- F – mutační konstanta
- D – dimenze problému, počet parametrů Fitness funkce



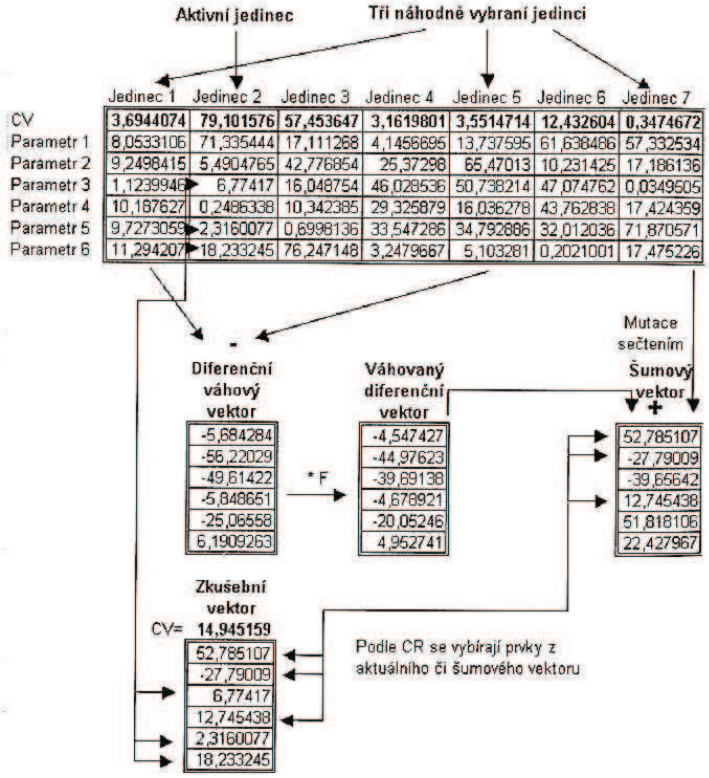
Obr. 2: Blokové schéma tvorby nové generace

Toto schéma nám ukazuje 3 základní kroky, které je třeba provést v každé iteraci programu.

Na následujících řádcích bude podrobně rozebrán každý z těchto kroků.

3.3.3.1 Generace zkušebních vektorů

V této fázi je pro každého jedince vygenerován tzv. zkušební vektor. Postup je názorně vidět na Obr.3. Pro každého jedince jsou náhodně vybraní tři různí jedinci (záleží i na pořadí vybrání). Od prvního vybraného je pak odečten druhý vybraný a tento rozdíl ve vynásoben mutační konstantou F. Výsledek této operace je pak přičten k třetímu vybranému jedinci, takovýto vektor se pak nazývá šumový. Následně je z hodnot šumového vektoru a jedince, pro kterého tento problém řešíme, vytvořen zkušení vektor tak, že pro každý parametr je vygenerováno náhodné číslo od 0 do 1, je-li toto číslo menší než CR, bude hodnota tohoto parametru zkušebního vektoru naplněna ze šumového vektoru, v opačném případě je hodnota zkopírována z původního jedince.



Obr. 3: Postup při generování zkušebního vektoru převzatý z [2]

Pozn.: V Obr.3 je mimo parametrů i hodnota označena CV jde o hodnotu fitness funkce (cost value).

3.3.3.2 Ohodnocení zkušebních vektorů

V této fázi je každý jedinec ohodnocen pomocí účelové funkce. Toto hodnocení je vhodné uložit ke každému ze zkušebních vektorů, postoupí-li pak daný vektor do další generace jako jedinec, nebude už nutné ho znovu hodnotit.

3.3.3.3 Postup silnějších jedinců do další generace

Jsou porovnány hodnoty účelové funkce u odpovídajících dvojic zkušebních vektorů a původních jedinců. Do další generace pak postupuje ten vektor, který je z hlediska fitness funkce výhodnější.

3.3.3.4 Tvorba první generace

Před spuštěním vlastní evoluce musí být náhodně vygenerováni a ohodnoceni jedinci v patřičných mezích, stanovených našimi požadavky.

3.4 Genetické algoritmy

Představují první a asi nejobecnější evoluční optimalizační techniku. Jsou inspirované představou Darwinovské evoluce, tedy že nejsilnější jedinec má největší šanci na přežití, všichni členové populace jsou proto podrobeni selekci. Následně se předpokládá, že pokud se množina „přeživších" spolu spáří, v nové populaci se vyskytnou kvalitnější jedinci, než byli generaci rodičů.

Samozřejmě mimo hlavní myšlenku bylo zjištěno, že tyto algoritmy pracují lépe, když do evoluce přidáme i mutaci nebo elitismus (nejsilnější jedinec nebo množina nejsilnějších jedinců postupují do další generace beze změny).

Ve snaze vylepšit standardní genetické algoritmy vznikla celá řada metod, jak jedince mezi sebou křížit jak navrhnout selekci apod. Např.: Selekce- standardní algoritmy počítaly s tím, že se vezme množina nejsilnějších jedinců, ale ukázalo se, že toto řešení často končí v lok. extrému, proto byly vyvinuty jiné způsoby selekce, jako je turnajová selekce, kdy jsou jedinci rozděleni na N skupinek s n (často jen dvěma) jedinci a z každé skupiny postupuje pouze jeden nebo dva nejlepší jedinci, nebo tzv. ruletové kolo kde si můžeme představit, že každý jedinec na základě své kvality zabírá různě velkou část ruletového kola.

Podobné příklady by se dali najít na křížení dvou jedinců a na mutaci. Vzhledem k velkému množství různých parametrů a postupů jsme se rozhodl, že genetické algoritmy nepoužiji.

3.5 Příklady jiných evolučních algoritmů

Jak již bylo řečeno, existuje celá řada evolučních algoritmů. Podrobně byly rozepsány pouze dva, jelikož tyto dva jsem studoval a rozhodoval jsem se který z nich použiji. Nicméně existuje i řada algoritmů, které se na první pohled nehodili pro řešení mně zadaného problému, zde je několik příkladů:

- Ant Colony Optimalization – metoda založená na pozorování mravenců a jejich metody hledání cesty. Tato metoda je výhodná pro problémy hledání optimální cesty.
- Imunology system metod – systém inspirován fungování imunitního systému v živých organismech. Lze použít jako antivirovou ochranu.
- Scatter search – Vektorově orientovaný algoritmus generující nové algoritmy na základě heuristických technik.
- Particle swarm – je založen na práci s populací jedinců, jejichž pozice v prostoru je měněna za základě rychlostního vektoru přičemž dochází k vzájemnému ovlivňování mezi jedinci v oblastech zvaných sousedství.

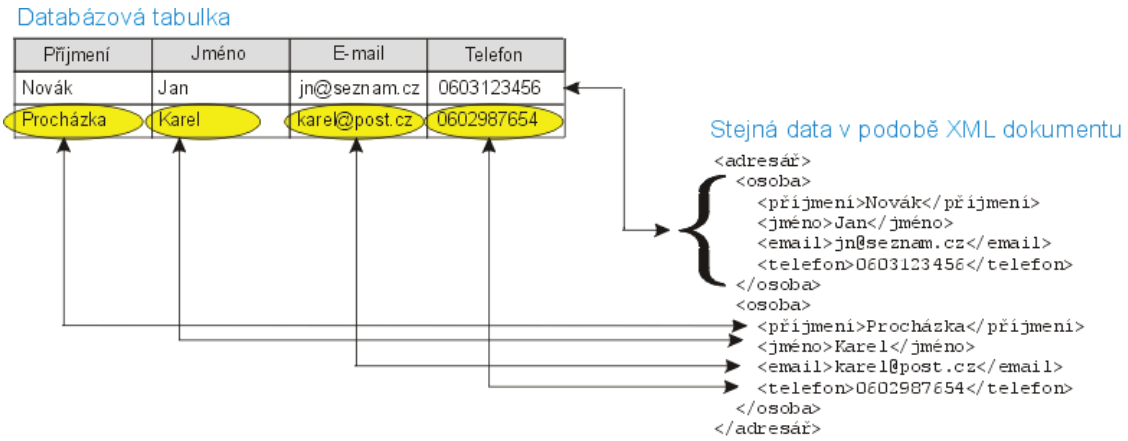
4 DALŠÍ POUŽITÉ TECHNOLOGIE

Při řešení své bakalářské práce jsem využil i jiné pro práci méně významné technologie. Nicméně jsou použity jak v samotném programu tak jsou zmíněny i v této práci a proto bude v této kapitole popsaná teorie k takovýmto prvkům mé práce.

4.1 XML

Jazyk XML (eXtensible Markup Language) je normalizovaným textovým formátem pro ukládání struktur dat. Stylem může připomínat HTML, a také ho lze použít i pro tvorbu webových stránek. Hlavní rozdíl mezi HTML a XML je v tom že v XML si můžeme definovat vlastní tagy. Důvodem podobnosti XML s HTML je podobná historie oba vznikly z SGML (Standard Generalized Markup Language) tedy formátu, což je obecný jazyk, který vznikl a byl normalizován v roce 1986 z potřeby po standardizovaném ukládání formátovaného textu. [6]

Ale XML neslouží pouze k ukládání textu jak můžeme vidět na Obr. 4, snadno si poradí i s jakýmikoliv daty, např.: při psaní programu v knihovnách .NET je podporované ukládání veškerých uživatelských nastavení ukládána právě ve formě XML.



Obr. 4: Příklad uložení databázové tabulky do XML převzatý z [6]

Na Obr. 4 též vidíme typickou strukturu XML dokumentu. Celý dokument je uzavřen v hlavním tagu a každý prvek který dokument označuje je uzavřen ve vlastním tagu a stejně tak každý atribut kteréhokoliv prvku.

XML plně vyhovuje standartu unicode a tak nevzniká žádný problém s českými znaky.

Já jsem XML využil ve své práci pro ukládání modelu tělesa, jak popisuje kapitola 5.4.4, a pro ukládání výsledku. U obou těchto případů využívám knihovny .NET k vytvoření XML dokumentu.

4.2 Využití GPU k paralelním výpočtům

Důvod pro využití GPU k výpočtům místo CPU stojí hlavně na rozdílech mezi těmito dvěma typy procesorů. GPU narozdíl má narozdíl od CPU méně vnitřní logiky, nemá

např.: logikou pro obsluhu přerušení či pro ochranu paměti, stejně tak logikou pro různé dynamické předvídání instrukcí. Hlavní rozdíl je však v počtu výpočetních jednotek. Zatím co současná moderní CPU mají 2 až 8 jader, na GPU jich najdete kolem 800. Nejde však o plnohodnotná jádra, jako naleznete na CPU. [5]

Jelikož GPU obsahuje velké množství jader umožňuje program paralelizovat do obrovského množství vláken, což se projeví při práci s velkými poli dat. Např.: Máme-li pole které má 10000 prvků a je třeba je všechny vynásobit dvěma CPU v jednom taktu dokáže vynásobit jednu hodnotu, kdežto GPU může v jednom taktu vynásobit všech 10000 hodnot. Samotný takt ale vzhledem k nižší taktovací frekvenci GPU tvá trošku déle než u CPU.

Vidíme, že GPU je vhodné použít pro úkoly, které je možné dobře paralelizovat, jako jsou operace nad poli a maticemi prvků. Podmínkou ovšem je, že se nad všemi prvky bude provádět stejná operace a, že jednotlivé položky jsou na sobě nezávislé! I na GPU je sice možné provádět nad každým prvkem jinou operaci, ale v takovém případě bude výkonnostní přínos malý, nebo žádný. [5]

Zásadním problémem je u GPU již zmíněná chybějící ochrana paměti a především pak propustnost a zpoždění na sběrnici mezi GPU a CPU. [5]

Dříve se na programování aplikací využívajících GPU používaly knihovny poskytující API pro práci s počítačovou grafikou jako jsou Open GL, či DirectX. Tento přístup nebyl zrovna nejvhodnější, protože programátoři museli svůj program přetransformovat tak, aby jeho výpočty odpovídali grafickým výpočtům poskytovaných těmito knihovnami. Potřebu vhodnějšího prostředku pro práci s GPU, který by umožnil využívat jeho vysoký výkon, si uvědomily i firmy vytvářející grafické čipy a knihovny pro práci s nimi. [5]

Výsledkem jsou tři balíky knihoven, pomocí kterých můžete programovat aplikace využívající GPU:

- DirectCompute
- CUDA
- OpenCL

Za prvním jmenovaným stojí firma Microsoft a za druhým firmu nVidia. Za poslední jmenovanou OpenCL stojí konsorcium Khronos Group, které sdružuje desítky firem a jeho účelem je tvorba otevřených standardů. [5]

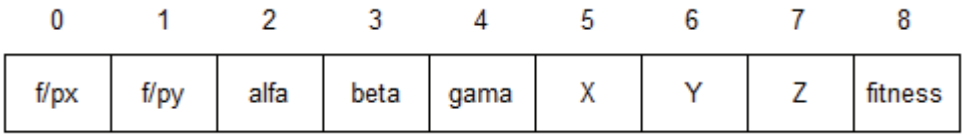
Já jsem si pro svou práci zvolil technologii CUDA a o z několika důvodů jako je dostupnost dokumentace a nejrůznějších příkladů, dostupnost výkonné nVidia karty v laboratoři a v neposlední řadě taky na doporučení vedoucího mé práce.

5 VÝPOČETNÍ JÁDRO PROGRAMU

Pro kalibraci kamery jsem se rozhodl použít algoritmus diferenciální evoluce. V úvahu jsem bral ještě genetické algoritmy. Nicméně mě od nich odradil fakt že mnohdy značně obtížné zvolit správný algoritmus té či oné fáze(selekce, mutace …) a vhodná volba těchto algoritmů je správnou funkci naprosto kritická. Naproti tomu diferenciální evoluce je jasně daná a doladění se provádí pomocí několika parametrů, navíc je celý algoritmus rychlý, jednoduchý a dosahuje dobrých výsledků.

5.1 Jedinec

Elementární prvek diferenciální evoluce, jde o jednu sadu proměnných parametrů účelové funkce. V našem případě jde o jedno nastavení modelu kamery. K jedinci je zároveň připojena i hodnota účelové funkce, aby nemusela být počítána pokaždé znovu. Graficky je datová struktura jedince znázorněna na Obr. 5.



Obr. 5: Datová struktura jedince

Úhly alfa, beta, gama značí rotace v osách x,y a z. X,Y,Z označují translaci.

V našem případě se tedy jedinec skládá z 9 čísel typu double. Můžeme zde tedy názorně vidět že dimenze problému (nebo-li parametr D) je pro zvolený model kamery rovna 8.

5.2 Populace

Populace je množina jedinců. Velikost této množiny značně ovlivňuje chování diferenciální evoluce. V zásadě platí že čím větší je populace tím větší je odolnost proti zaseknutí se v lok. extrému problému. V [2] se můžeme dočíst, že je vhodné volit velikost populace v rozmezí od 10D do 100D. Z hlediska vyšší výpočetní rychlosti jsem zvolil 10D, ukázalo se že tato velikost populace je náš problém dostačující. Parametr NP je tedy 80.

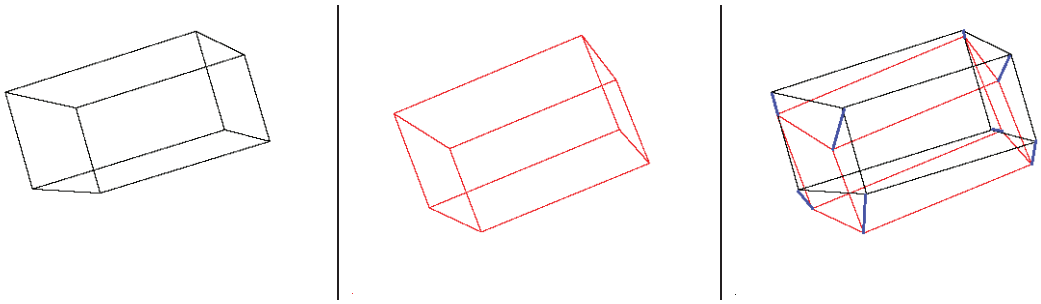
5.3 Fitness funkce

Jak už zde bylo mnohokrát psáno výše, fitness funkce má za účel ohodnotit kvalitu každého jedince. Je naprosto klíčovou funkcí celé diferenciální evoluce, a proto musí být nejen vhodně napsaná, tak aby rozlišila dobré a špatné jedince, ale zároveň dostatečně rychlá, protože bude volána v průběhu evoluce neustále.

Jak tedy vyhodnotit správné nastavení kamery? Řešení je nasnadě, musíme porovnat obraz, který nám dává model kamery, se skutečným obrazem z reálné kamery. Resp. nemusíme porovnávat celý obraz, ale pouze body objektu u něhož známe jeho reálné rozměry (prostorové souřadnice) a souřadnice pixelů na něž je reálná kamera zobrazila.

$$F = \sqrt{\sum_i \left(x_{Si} - x_{Zi}\right)^2 + \left(y_{Si} - y_{Zi}\right)^2} \tag{5.1}$$

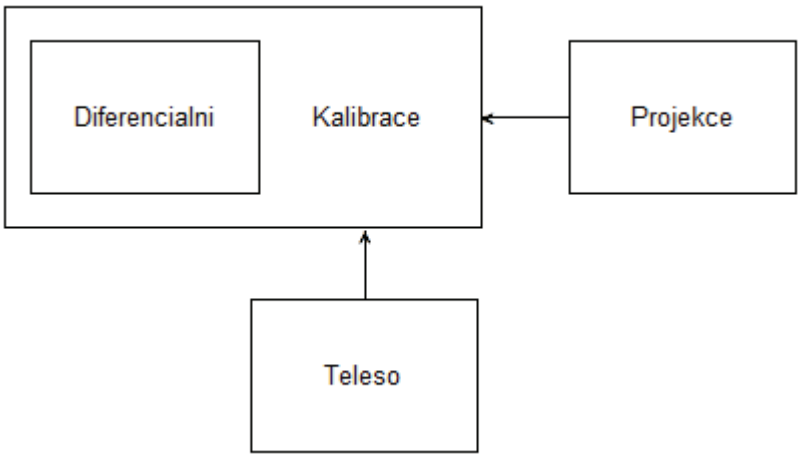
Matematicky fitness funkci popisuje rovnice (5.1), kde souřadnice s indexem S jsou skutečné, nasnímané reálnou kamerou, souřadnice s indexem Z jsou souřadnice zobrazené pomocí modelu kamery nastaveným podle dat příslušného jedince. Sčítám mocniny těchto vzdáleností, protože tak dostávám podobnou vlastnost, pro kterou se používá metoda nejmenších čtverců. Výsledný součet odmocňuji z důvodu větší srozumitelnosti výsledku. Samotná suma je totiž v jednotkách pixel^2 odmocněním dostávám výsledek v pixelech.



Obr. 6: Grafické znázornění fitness funkce

Obr.6 znázorňuje graficky princip fitness funkce na obrázku úplně vlevo vidíme objekt pro nějž hledáme parametry kamery. Na prostředním obrázku je tento objekt promítnutý z dat nějakého jedince. Na posledním obrázku pak vidíme vyznačené vzdálenosti bodů, jejichž mocniny následně sečteme a součet odmocníme.

5.4 Implementace do C++



Obr. 7: Blokové schéma programu pro kalibraci kamery

Na Obr 7. vidíme grafické znázornění mého programu. V následujících podkapitolách se budu věnovat každé ze 4 tříd mého programu.

5.4.1 Třída Diferencialni

Tato třída se stará o běh diferenciální evoluce. Je napsána obecně a tak aby bylo možné ji v budoucnu aplikovat i na jiné problémy. Dalo by ses svým způsobem říct že se jedná o abstraktní třídu (i když tomu není úplně tak, protože její instanci je možné vytvořit), jelikož jediné smysluplné využití této třídy tkví v jejím zdědění. Obsahuje virtuální metodu Fitness, která obsahuje Testovací účelovou funkci pro libovolnou dimenzi problému.

Nejdůležitější metodou obsaženou metodou je metoda Reprodukce, která provede jednu iteraci dif. evoluce. Při psaní této metody jsem se dopustil chyby. Tato metoda totiž ještě před doběhnutím iterace ukládá již vyřešené prvky zpět do populace a proto jsou jedinci v generaci s číslem X, kteří jsou v populaci „v zadu“ resp. jedinci s vyšším indexem, generováni z jedinců z nové generace s číslem X+1. Tuto chybu jsem si uvědomil, když jsem v [2] studoval možné úpravy dif. evoluce, kde je tato úprava popsána jako výhodná u některých problémů. Rozhodl jsem se tedy srovnat oba přístupy a zjistil, že při řešení kalibrace kamery je o trošku výhodnější původní přístup. Evoluce se s touto „chybou“ ustálí o přibližně 10 iterací rychleji.

5.4.2 Třída Kalibrace

Tato třída dědí od třídy Diferenciální, narozdíl od třídy Diferenciální už není obecná ale je zaměřená na řešení kalibrace kamery. Má vhodnou fitness funkci, kterou nahrazuje testovací fitness funkci třídy kalibrace. Dále obsahuje instance tříd Projekce a Těleso a umožňuje provést potřebná nastavení těchto tříd (např.: u třídy Projekce je nutné nastavit střed obrazu $\lfloor c_x, c_y \rfloor$ a u třídy Těleso je nutné vytvořit kalibrační objekt). Zároveň tato třída automaticky nastavuje parametry a meze dif. Evoluce.

5.4.3 Třída Projekce

Tato třída obsahuje model kamery zapsaný rovnicí 2.2. pro zvýšení výpočetní rychlosti jsem matice roznásobil čímž z maticové rovnice vznikly dvě rovnice:

$$x_i = \frac{f}{p_x} \frac{\cos(\gamma)\cdot\cos(\beta)\cdot X_s + \cos(\gamma)\cdot\sin(\alpha)\cdot\sin(\beta)\cdot Y_S + \cos(\gamma)\cdot\cos(\alpha)\cdot\sin(\beta)\cdot Z_S - \sin(\gamma)\cdot\cos(\alpha)\cdot Y_S + \sin(\alpha)\cdot\sin(\gamma)\cdot Z_S + X_{trans}}{-\sin(\beta)\cdot X_S + \sin(\alpha)\cdot\cos(\beta)\cdot Y_S + \cos(\alpha)\cdot\cos(\beta)\cdot Z_S + Z_{trans}} + c_x \quad (5.2)$$

$$y_i = \frac{f}{p_y} \frac{\sin(\gamma)\cdot\cos(\beta)\cdot X_S + \sin(\gamma)\cdot\sin(\alpha)\cdot\sin(\beta)\cdot Y_S + \sin(\gamma)\cdot\cos(\alpha)\cdot\sin(\beta)\cdot Z_S + \cos(\alpha)\cdot\cos(\gamma)\cdot Y_S - \sin(\alpha)\cdot\cos(\gamma)\cdot Z_S + Y_{trans}}{-\sin(\beta)\cdot X_S + \sin(\alpha)\cdot\cos(\beta)\cdot Y_S + \cos(\alpha)\cdot\cos(\beta)\cdot Z_S + Z_{trans}} + c_y \quad (5.3)$$

Výpočet těchto rovnic je ještě urychlen tím, že hodnoty goniometrických funkcí se počítají předem, což se projeví především u zobrazování těles s velkým množstvím bodů.

Samozřejmě můžeme vidět potenciální problém v tom, že jmenovatel zlomku může být roven nule. Roven nule bude v okamžiku kdy bude zobrazovaný bod po rotaci a translaci v rovině obsahující ohnisko a rovnoběžné s rovinou obrazu. Toto není v této

třídě ošetřeno. Ošetřeno je to vhodným nastavením mezí ve třídě kalibrace díky čemuž k tomuto stavu nemůže dojít.

5.4.4 Třída Teleso

Tato třída spravuje informace o kalibračním objektu. Obsahuje informace jak o bodech v prostoru tak i o souřadnicích pixelů kam se dané body zobrazily. Zároveň pro přehlednější zobrazení předmětu umožňuje udržovat informace i o tom které body jsou opticky spojené.

Těleso je možné ukládat do xml souboru a následně ho odtud načítat. Při psaní těchto metod jsme využil knihovny .NET. Abych ale nepřišel o platformní nezávislost pro věc, která není zcela nutná, bude tato část kódu preprocesorem vyloučena, jestliže nebude při deklaraci třídy použito #define NET. Pro představu jak vypadá struktura tělesa uloženého do xml je na dalších řádcích ukázka úsečky (větší těleso by zabralo moc místa).

```
<?xml version="1.0" encoding="utf-8"?>
<Teleso>
  <Bod>
    <X>1</X>
    <Y>1</Y>
    <Z>1</Z>
    <ImageX>0</ImageX>
    <ImageY>0</ImageY>
    <Spojeni />
  </Bod>
  <Bod>
    <X>-1</X>
    <Y>1</Y>
    <Z>-1</Z>
    <ImageX>0</ImageX>
    <ImageY>0</ImageY>
    <Spojeni>
      <Bod>True</Bod>
    </Spojeni>
  </Bod>
</Teleso>
```

5.5 Nastavení parametrů diferenciální evoluce

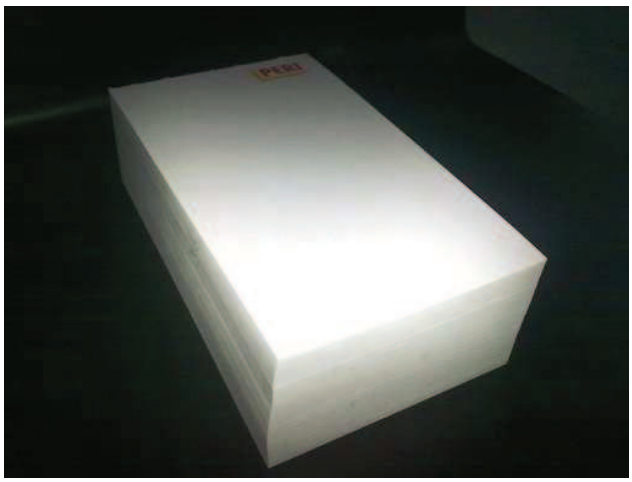
Jak již bylo řečeno výše, diferenciální evoluce má čtyři základní parametry: Jde o dimenzi problému D, počet jedinců v populaci NP, mutační konstantu F a křížící konstantu CR. Význam těchto konstant je vysvětlen v kapitole 4.3.3. Jako parametr evoluce by se dala brát i fitness funkce, ale ta je již jasně daná problémem. Dalšími parametry jsou meze ve kterých hledáme řešení.

5.5.1 Nastavení základních parametrů

Parametry D a NP jsou popsány v kapitolách 5.1 a 5.2. D je pevně zvolen a NP je nastaven na optimální hodnotu z hlediska rychlosti a odolnosti vůči stagnaci nebo-li zaseknutí v lok. extrému. Ze stejného hlediska jsem hledal optimální nastavení parametrů F a CR.

Jako první věc jsem si musel stanovit způsob, kterým budu hodnotit rychlost evoluce při daném nastavení F a CR. Zde se naprosto jednoznačně nabízí porovnávat počet iterací nutných k ustálení populace v extrému. Tento bod můžeme snadno poznat pomocí sledování směrodatné odchylky z hodnot fitness funkce všech jedinců v populaci.

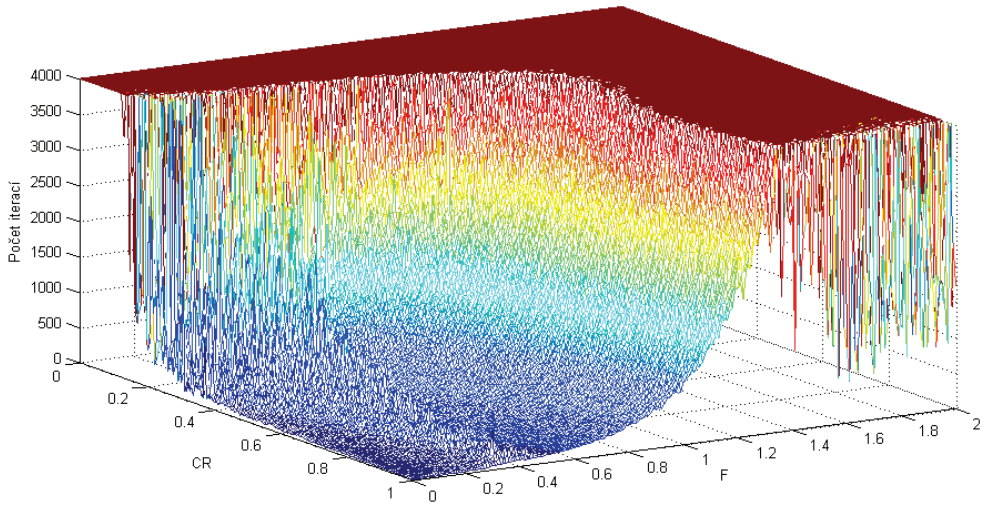
Jako testovací data, na nichž jsem pokus prováděl, jsem si zvolil fotku bločku (Obr.6) s reálnými rozměry: a= 87,5 mm, b= 42,2 mm a c= 148,1 mm. Souřadnice pixelů zobrazených rohů jsem do programu zadal ručně.



Obr. 8: Testovací obraz

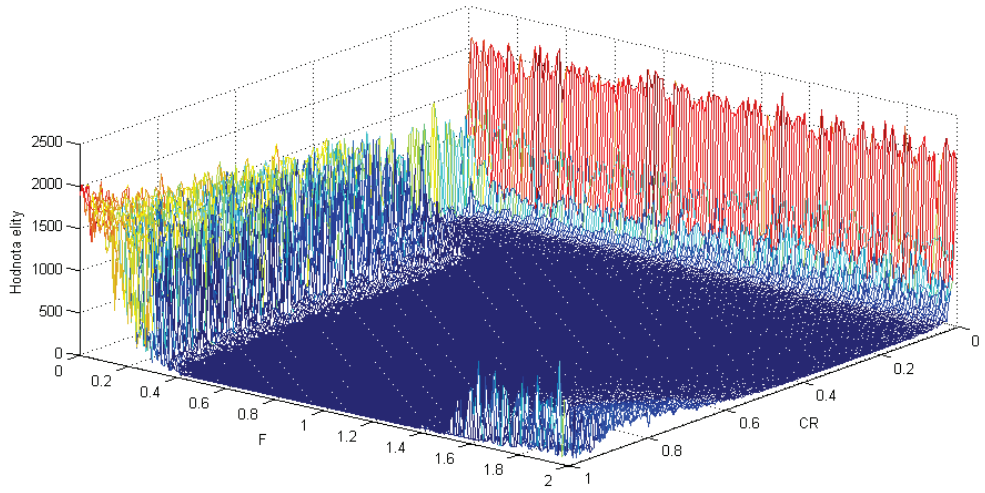
Následně jsem napsal program, který spouštěl kalibraci kamery s všemi kombinacemi $F \in \langle 0,2 \rangle$ a $CR \in \langle 0,1 \rangle$ pro F s krokem 0,01 a pro CR s krokem 0,005 a následně zaznamenával výsledky – počet iterací nutných ke ztrátě směrodatné odchylky (resp. poklesu pod hodnotu 0,01) a hodnotu fitness funkce nejlepšího jedince v populaci.

Přirozeně jsem musel omezit maximální počet iterací pro případy nastavení u kterých nebude možné najít výsledek. Maximální počet iterací jsem omezil na 4000.



Obr. 9: Počet iterací nutných k ustálení populace v závislosti na F a CR

Na Obr.9 vidíme při které iteraci se populace přestala vyvíjet z tohoto grafu by jsme mohli usuzovat, že nejlepší bude CR co největší a F co nejmenší. Ale musíme se ještě podívat na Obr.10, aby jsme zjistili kde se populace přestala vyvíjet.



Obr. 10: Hodnota fitness funkce nejlepšího jedince v závislosti na CR a F

Zde vidíme že v oblasti, kde by podle Obr.7 bylo nejlepší umístit pracovní bod, evoluce skončila v lokálním extrému a nejvhodnější bude volit F větší než 0,4.

Proto jsem pro kalibraci kamery zvolil $F = 0{,}5$ a $CR = 0{,}99$ u tohoto nastavení je třeba pro výpočet kalibrace kamery pomocí testovacího objektu 110-130 iterací.

Pozn.: Při prvních pokusech jsem používal, podle [2], „univerzální“ (univerzální samozřejmě do jisté míry) nastavení F a CR. F = 0,8 a CR = 0,85. Tyto hodnoty jsou nastaveny konstruktorem třídy Diferencialni a jsou požity v případě, že nejsou změněny.

5.5.2 Nastavení mezí

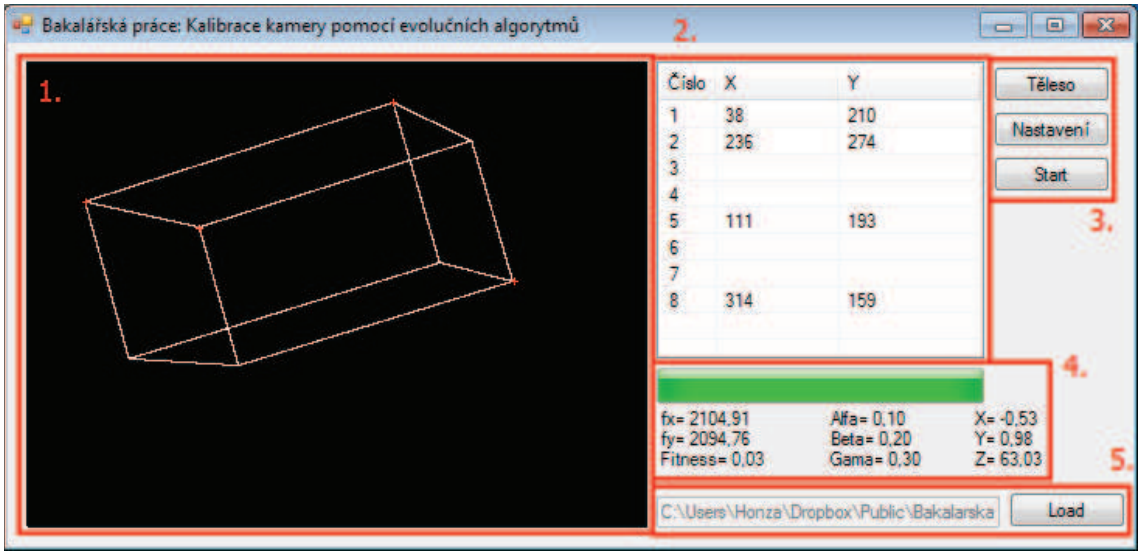
Nastavení mezí je nutné pro funkci dif.evoluce. Používá se při generovaní každé generace , u první se generují všechny prvky v těchto mezích a u dalších generací je v těchto mezích náhodně vygenerován prvek, který danou mez překročil.

Meze by samozřejmě měli být vždy určené řešeným problémem v mém případě jsou naprosto jasně určeny pouze 3 meze a to jsou meze u rotací, které jsou v rozmezí $\langle -\pi; \pi \rangle$. Další důležitá mez je spodní mez u Z translace tu je vzhledem k problému nutnou vyčíslit tak, aby se po translaci nemohl žádný bod promítnout na (nebo dokonce za) rovinou rovnoběžnou se zobrazovací rovinou a obsahující ohnisko. Ostatní meze je nutné určit podle obrázku tak, aby se mezi nimi nacházelo řešení.

Meze jsou implicitně přiřazené konstruktorem jejich hodnoty můžeme vidět na Obr.13.

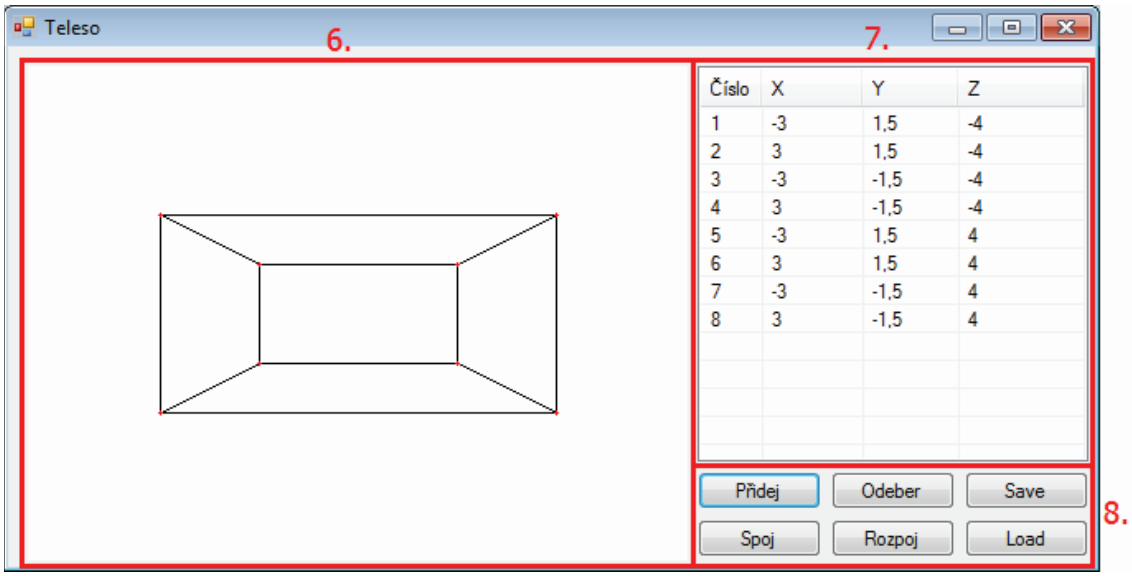
6 GRAFICKÉ UŽIVATELSKÉ ROZHRANÍ

Výpočetní jádro programu popsané v kapitole 5 je samozřejmě klíčovým prvkem celého programu, ale před zahájením výpočtu je třeba mu zadat řadu parametrů a po dokončení výpočtu, je třeba výsledná data prezentovat uživateli. A právě k tomuto účelu slouží GUI nebo-li grafické uživatelské rozhraní.



Obr. 11: Printscreen uživatelského rozhraní pro kalibraci kamery

Na Obr. 11 vidíme GUI mého programu, očíslovanými obdélníky je rozdělen do několika částí, které popíšu v následujících podkapitolách.



Obr. 12: Printscreen uživatelského rozhraní pro tvorbu kalibračních objektů

A na Obr 12. vidíme uživatelské rozhraní okna pro nastavení modelu kalibračního objektu.

6.1 PictureBox pro zobrazení obrazu z kamery

Oblast č. 1 ohraničuje pictureBox, jehož účelem je zobrazit uživateli obraz nasnímaný z kamery. Tento obraz lze pomocí kolečka myši přibližovat a oddalovat. Při přibližování se vždy zvětšuje oblast na které je kurzor.

Tento pictureBox je zároveň požíván při zadávání souřadnic pixelů, na nichž se zobrazují body vyfoceného objektu. Potom co je v listView (kap. 6.2) označen nějaký bod bude kliknutím na picturebox zaznamenána jeho poloha a na obrazu se zobrazí červený křížek označující pozici bodu.

6.2 ListView pro zobrazení obrazových souřadnic

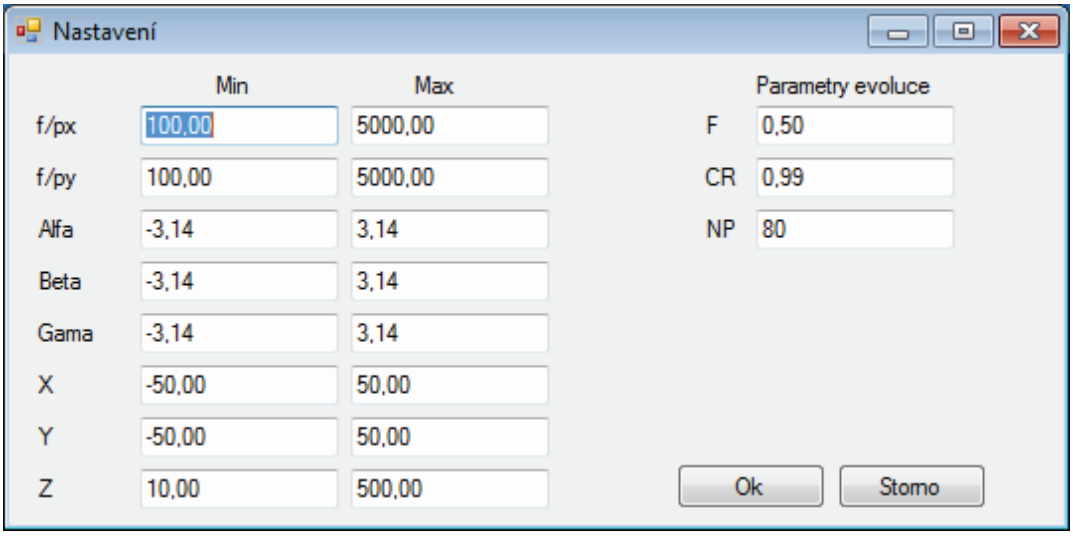
V oblasti č. 2 se nachází tabulka, která obsahuje informace o poloze pixelů, na kterých jsou pomítnuté body objektu, na němž kalibrujeme kameru.

Označíme-li některý bod v této tabulce může jej buď stiskem klávesy del smazat (budou vymazány pouze obrazové souřadnice tohoto bodu a nebude potom využitý při výpočtu fitness) nebo následným kliknutím na picturebox určit jeho polohu v obrázku.

6.3 Ovládací prvky hlavního okna

V této oblasti se nachází tlačítka s jejichž pomocí ovládáme program. Po tlačítka těleso vyvoláme okno zobrazené na Obr. 11, určené na tvorbu a úpravu kalibračních objektů. Stisknutím tlačítka Start spustíme diferenciální evoluci.

Stiskem klávesy nastavení vyvoláme okno (Obr. 13), ve kterém můžeme nastavit meze a ostatní parametry dif. evoluce. Funkci tohoto okna snad není třeba ani vysvětlovat.



Obr. 13: Okno pro nastavení parametrů dif. evoluce

6.4 Zobrazení průběhu evoluce a výsledků

Oblast označena č. 4 obsahuje místo pro zobrazení konečných výsledků. Zároveň obsahuje i progressbar, který zobrazuje průběh evoluce.

Při vypisování výsledku jsou výsledky zároveň zapisovány do xml souboru s názvem Výsledek.xml.

6.5 Načítání obrázku

V oblasti s č. 5 najdeme tlačítko, které vyvolá klasický openfiledialog pomocí něhož můžeme načíst obraz z kamery. Vstup obrazu je chráněný proti vložení nevhodného formátu souboru.

Textbox je stále zašedlý a pouze zobrazuje cestu k načtenému souboru nebo k souboru, který jsme se neúspěšně pokusili načíst.

6.6 PictureBox pro zobrazení kalibračního objektu

Oblast 6. se už nachází na okně tvorby modelu objektu pro kalibraci. Automaticky zobrazuje tvořený předmět. Využívá perspektivního promítání, a adaptabilně si nastavuje parametry zobrazování tak, aby byl objekt vždy celý v obrázku. Body, které se bude později možné využít ke kalibraci, jsou vyznačeny červeným křížkem. Pro přehlednější zobrazení je možné body navíc spojit přímkou a vytvořit tak drátový model reálného předmětu.

6.7 ListView pro zobrazení prostorových bodů modelu předmětu

Tabulka v oblasti č. 7 je určena k zobrazování prostorových bodů modelu předmětu pomocí něhož se bude řešit kalibrace kamery. Řádky jdou přidávat a odebírat pomocí ovládacích prvků (kap. 6.8). Odebrat prvek lze rovněž odebrat pomocí klávesy del.

6.8 Ovládací prvky okna pro tvorbu modelu

Pomocí tlačítek v oblasti 8. můžeme ovládat tvorbu modelu. Při stisku tlačítka Přidej je zobrazeno jednoduché okno, do kterého můžeme zadat souřadnice přidávaného bodu. Vstupy tohoto okna jsou pochopitelně ošetřeny proti zadání špatného formátu čísla. Tlačítkem Odeber se odebírá prvek, který byl při stisku tohoto tlačítka označen v tabulce.

Tlačítka Spoj a Rozpoj fungují tak, že je třeba mít při stisku jednoho z těchto tlačítek označený jeden z bodů, které chceme spojit resp. rozpojit a po stisku jednoho z těchto tlačítek kliknout do tabulky na druhý z bodů.

Tlačítka Save a Load využívají ukládání třídy Teleso do xml popsané v kap. 5.4.4. Před tím je ale pochopitelně vyvolávají OpenFileDialog resp. SaveFileDialog k určení cesty a názvu souboru.

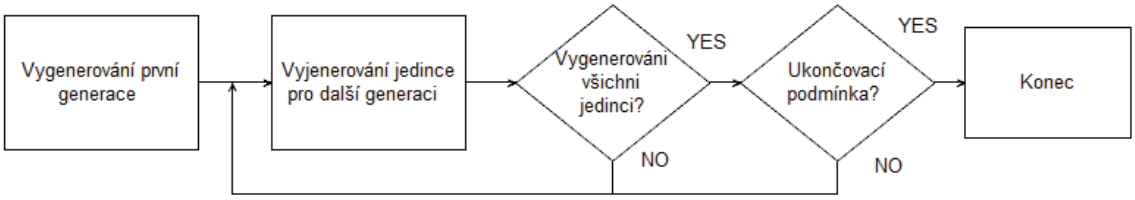
7 VYUŽITÍ GPU PRO AKCELERACI VÝPOČTU

Jedním z mých úkolů bylo i zhodnotit možnosti využití GPU a prakticky uvěřit zda mé zhodnocení bylo správné. Jak jsem již řekl pro práci s GPU jsem si vybral technologii CUDA.

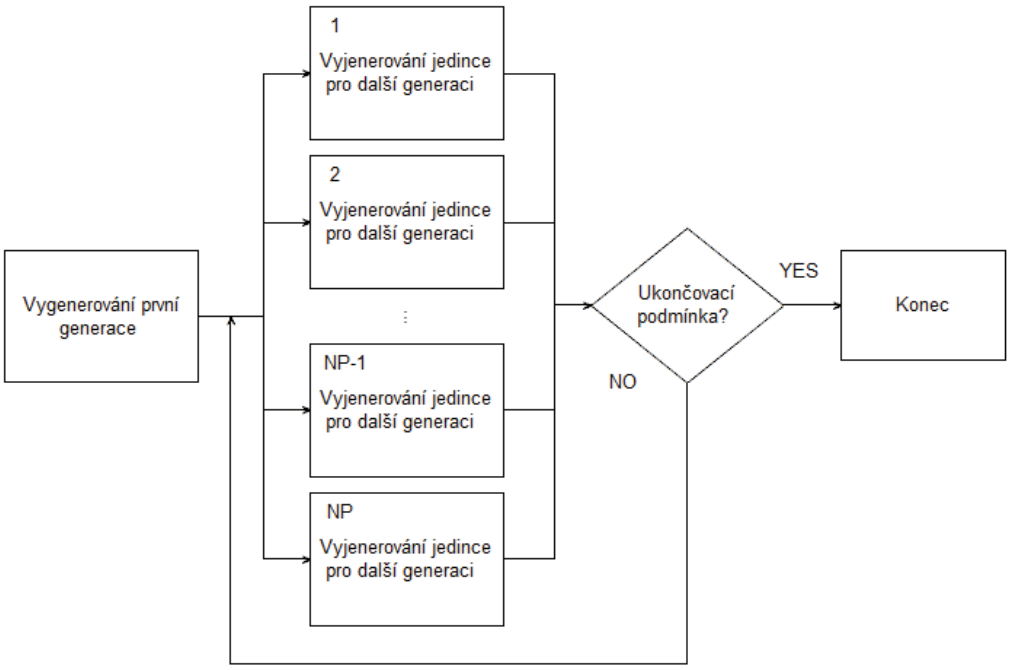
Pomocí této technologie lze spouštět kód napsaný v jazyce C na GPU, které tuto technologii podporuje, což jsou téměř všechny novější grafické karty od společnosti nVidia.

7.1 Základní princip

Základní myšlenka této kapitoly je prostá. U diferenciální evoluce probíhá spousta výpočtů, které je možno bezproblémů paralelizovat. Jejich výsledky jsou totiž použity až při další generaci. Graficky znázorněno je to na Obr.14 a Obr.15. Vidíme, že vhodným využitím grafické karty by měl být výpočet efektivnější.



Obr. 14: Zjednodušený vývojový diagram zpracování dif. evoluce sériově (CPU)



Obr. 15: Zjednodušený vývojový diagram zpracování dif. evoluce paralelně (GPU)

7.2 Úprava programu

Jako základ jsem samozřejmě použil třídu kalibrace popsanou v kapitole 5.4. Tuto třídu jsem zdědil a u klíčových funkcí jsem pozměnil funkcionalitu tak aby tato třída pracovala s grafickou kartou. První generaci jedinců generuji ještě na CPU a tyto data následně posílám na grafickou kartu se všemi ostatními daty potřebnými k chodu dif. evoluce.

Klíčová je bezpochyby globální funkce ReprodukceKernel, která obsahuje celý algoritmus tvorby nové generace včetně ohodnocení nových prvků. Tuto funkci bohužel není možné implementovat jako metodu mé upravené třídy, jelikož žádná třída v této verzi CUDy nemůže obsahovat metodu označenou jako __global__ (takto označené metody lze zavolat z hlavního programu jedoucího na CPU a vykonat na GPU).

Právě při psaní reprodukční funkce jsem narazil na největší z problémů, na které jsem narazil při používání CUDy. Šlo o to jak generovat na GPU náhodné číslo. Po dlouhém hledání a zjišťování proč na GPU nelze zavolat klasickou funkci rand() jsem zjistil že tato funkce by neměla být kvůli své podstatě volána z více threadů v jakékoliv aplikaci. Generuje totiž pseudonáhodně řadu čísel, na začátku dostane tzv. seed a z něj pak vygeneruje řadu čísel. Stejný seed stejná řada čísel. Tato část vícevláknovému zpracování ještě vůbec nevadí problém je že po každém vygenerování náhodného čísla je třeba do seedu zapsat jinou hodnotu a právě kvůli této fázi není možné funkci rand() použít.

Protože problém s generováním náhodných čísel je znám jistě již dlouhou dobu tak přímo v balíčku knihoven, které je nutno stáhnout před prací s CUDou je přiložena knihovna s názvem curand, která řeší tento problém poskytnutím funkcionality pro generování náhodných čísel vícevláknově. Na začátku programu je nutné si vygenerovat seed pro každé vlákno, které bude později generovat náhodné čísla a při samotném generování stačí funkci zavolat s příslušným seedem.

Poslední funkce, které jsem musel napsat pro GPU byli funkce pro hledání nejlepšího jedince v populaci a výpočet směrodatné odchylky populace. Tyto funkce by samozřejmě byli rychlejší na CPU, ale rozhodl jsem se tyto úlohy vykonávat na GPU sice pomaleji ale bez nutnosti přenosu dat mezi GPU a CPU.

7.3 Výsledky

A teď to nejdůležitější. Jak to tedy funguje. Co se týká přesnosti výsledků tak ta je stejná jak při zpracování programu pomocí GPU stejná jako u zpracování pomocí CPU. Co do rychlosti je vidět že GPU je při zpracování rychlejší, bezpochyby doba vykonání jedné iterace je oproti CPU asi poloviční nicméně musíme vzít v potaz, že počítání směrodatné odchylky GPU značně zdržuje.

Soustředil jsem se na výpočetní rychlost jako celek a nezabýval jsem se tím která část jak dlouho trvá. Pro měření rychlosti jsem napsal testovací program, který střídavě spouští kalibraci kamery na CPU a GPU a celý proces opakuje 10x. Samotnou výpočetní dobu měřím dobu potřebnou k ustálení populace bez počátečního nastavování a generování první generace. Výpočet směrodatné odchylky populace, vzhledem k jeho

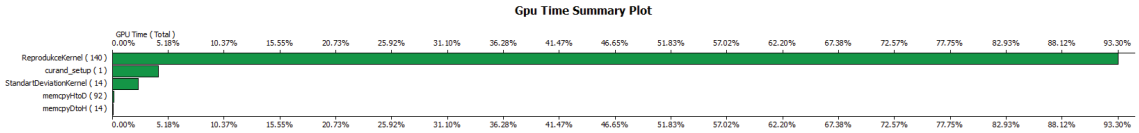
náročnosti pro GPU, provádím jednou za 10 iterací. Celé toto měření jsem provedl pro 3 uměle vytvořené obrazy těles Obr.17. Vzhledem k tomu, výhody GPU se projeví až při zpracovávání větších populací, provedl jsem toho měření jak pro velikost populace 80 tak 1000. Výsledné hodnoty jsem zprůměroval a umístil do tabulky 1.

	80 jedinců			1000 jedinců		
	8 bodů	16 bodů	32 bodů	8 bodů	16 bodů	32 bodů
GPU	38,9 ms	48,7 ms	62,5 ms	170,6 ms	209,5 ms	272,4 ms
CPU	20,3 ms	30,5 ms	47,4 ms	278,7 ms	417,2 ms	653,4 ms

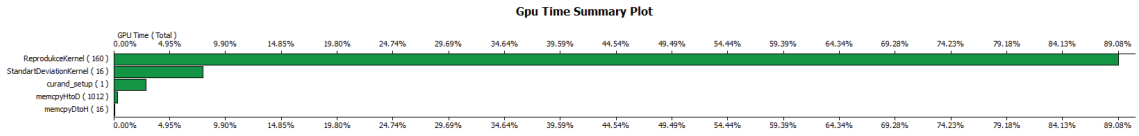
Tabulka 1: Výsledky měření časové náročnosti kalibrace kamery při zpracování pomocí CPU a GPU pro NP= 80 a NP=1000

V této tabulce vidíme důkaz, že využít GPU se vyplatí až u velkých populací. Použijeme-li implicitní nastavení (80 jedinců) zpracování na CPU bude tozhodně výhodnější, ale existují problémy kde by nám tato malá populace mohla dělat problémy a bylo by třeba ji zvětšit. K řešení takovýchto problémů by se mohlo vyplatit použít GPU, ale chce to zvážit, zda se vyplatí zrychlit program relativně sice dvojnásobně ale absolutně o desetiny sekundy.

Jak je zmíněno v kapitole 4.2, jednou z nevýhod využití GPU je nutnost posílat na GPU vstupní data a z GPU posílat zpátky ty výstupní. Vzhledem k to mu, že tato fáze programu probíhá ještě před začátkem měření, analyzoval jsem program pomocí aplikace NVIDIA Compute Visual Profiler abych zjistil, v jakém poměru je výpočetní čas GPU s dobou nutnou na přípravu výpočtu. Výsledek této analýzy vidíme na Obr. 16 a Obr. 17, je na první pohled patrné že doba pro přenos dat je naprosto zanedbatelná vzhledem k době běhu programu. Z rozdílů obou grafů si můžeme všimnout, že s zvětšující se populací se citelně zvyšuje doba nutná pro výpočet směrodatné odchylky.



Obr. 16: Graf poměru výpočetní doby funkcí vykonávaných GPU při NP = 80 a tělese s 32 body



Obr. 17: Graf poměru výpočetní doby funkcí vykonávaných GPU při NP = 1000 a tělese s 32 body

8 PŘESNOST

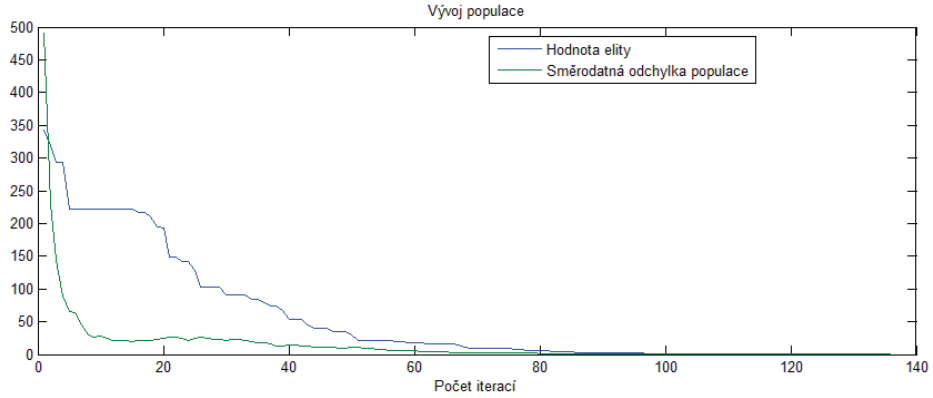
Na předchozích stránkách své práce popisuji princip funkce mého programu, v této kapitole se pokusím objektivně zhodnotit reálné výsledky, které dává můj program. Na začátku této kapitoly se budu též zabývat ukončovací podmínkou.

8.1 Ukončovací podmínka

Ač by se mohlo zdát, že tato podkapitola je umístěna ve špatné kapitole, není tomu tak. Ukončovací podmínka má totiž značný vliv na přesnost diferenciální evoluce. Evoluce je totiž schopna se vyvíjet nekonečně dlouho a, zvláště pak jsou-li parametry účelové funkce hodnoty s plovoucí desetinou čárkou, neustále zlepšovat řešení. Jde ale o zlepšení bezvýznamné a vykoupené takovým výpočetním časem, že je nutné nalézt optimum mezi přesností a výpočetní dobou a v tomto bodě evoluci ukončit.
Jak už jsem se předčasně zmiňoval v kapitole 5.5.1, hlavní vliv na to zda má evoluce dále pokračovat má v mém programu směrodatná odchylka z hodnot fitness funkce všech jedinců populace. Pro případy kdy se není dif. evoluce schopna ustálit je v programu ještě záložní podmínka s maximálním počtem iterací (v současnosti nastavená na 1000 iterací).

V průběhu evoluce se populace ve svém definičním oboru shlukuje k bodu, který má nejvýhodnější hodnotu fitness funkce. Díky tomu by se mohlo zdát, že by jsme mohli počítat směrodatnou odchylku ze všech prvků jedinců a sledovat kdy se populace shromáždí u jednoho bodu. Ale existují-li ve fitness funkci dva stejně výhodné extrémy populace se pravděpodobně rozdělí shromáždí kolem obou a evoluce by se zastavila až o maximální počet iterací, proto počítám směrodatnou odchylku pouze z hodnot fitness funkce.

Program tedy po každé iteraci počítá hodnotu směrodatné odchylky a překročí-li odchylka danou mez evoluce je ukončena. Tuto mez jsem, na základě pozorování vývoje populace při kalibraci kamery, zvolil na hodnotu 0,01. Jak vidíme na Obr. 18, který zobrazuje graf sestrojený právě z dat zaznamenaných z hledání vhodného nastavení modelu kamery. Na tomto obrázku jasně vidíme, že vývoj populace se prakticky zastavil již asi 30 iterací před ukončením evoluce.



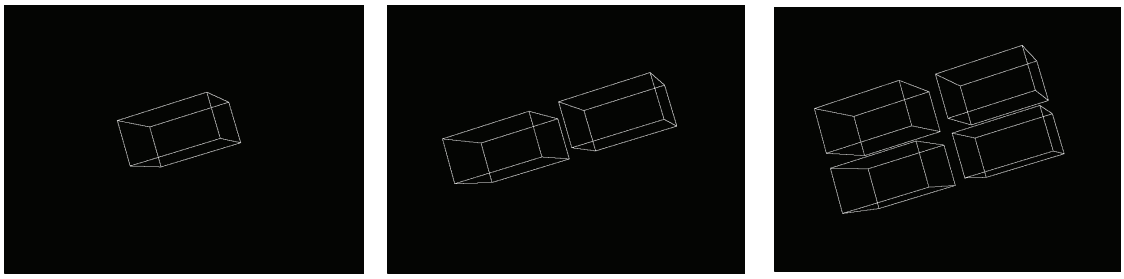
Obr. 18: Závislost hodnoty fitness funkce nejlepšího jedince a směrodatné odchylky na počtu iterací

8.2 Přesnost na umělých datech

Umělé data jsem vytvořil pomocí skriptu v matlabu s přesně určenými parametry kamery. Vytvořil jsem tři obrazy s parametry kamery:

- $\frac{f}{p_{x,y}} = 2000$
- $\alpha = 0{,}1\,;\ \beta = 0{,}2\,;\ \gamma = 0{,}3$
- $X = 0{,}1\,;\ \ Y = -0{,}5\,;\ Z = 60$

Tyto obrazy se liší pouze tělesem které na nich zobrazené resp. jde o kvádr, o těleso skládající se ze dvou kvádrů a o těleso ze čtyř kvádrů. Tyto obrazy můžeme vidět na Obr. 19.



Obr. 19: Tři obrázky s umělými daty

Na každém z obrázků jsem provedl 10 měření a výsledky nakonec zprůměroval, vypočítal absolutní odchylky a zanesl je do tabulky 2.

N [-]	f/px [-]	f/py [-]	α [rad]	β [rad]	γ [rad]	X [mm]	Y [mm]	Z [mm]
8	30,18345	23,18695	-0,00113	0,001663	0,000289	0,02852	0,039365	0,926297
16	25,86938	28,00757	-0,00055	0,00053	0,000575	0,038474	0,048379	0,736203
32	2,517208	4,025338	0,001233	0,002487	0,001112	0,0325	0,043187	0,04919

Tabulka 2: Absolutní odchylky od skutečných hodnot na umělých datech

Z hodnot můžeme vidět, že výpočet rotací je velice přesný odchylky jsou tu v řádu tisícin a desetitisícin radiánu. U translací v osách x a y dochází sice k procentuálně vysoké chybě, ale to bude způsobeno nízkými hodnotami translace v těchto osách. K největších chybám dochází u ohniskových vzdáleností a u translace v ose z, jak je z hodnot vidět zvýšenou přesnost u těchto hodnot můžeme zajistit vyšším počtem bodů tělesa, pomocí kterého kalibrujeme kameru.

8.3 Přesnost na reálných datech

Jak změřit přesnost mého programu na reálných datech byl poměrně zajímavý problém. Vytvořit totiž reálné data, u kterých bych věděl skutečné parametry kamery, tak jako jsem to udělal s umělými daty, by bylo velmi obtížné. Proto jsem se rozhodl pro jiný přístup k tomuto problému.

Jsou-li na jednom snímku dva objekty, pro něž lze provést kalibraci kamery můžeme zjistit u obou těchto těles jejich vzdálenost a natočení vzhledem ke kameře. Z těchto

hodnot poté můžeme vypočítat vzdálenost a natočení těchto těles vůči sobě. Stačí tedy jen předem znát tento vztah a můžeme tímto způsobem znázornit přesnost zjištěných údajů.

Vzhledem k tomu jak často se při kalibraci kamery využívá šachovnice rozhodl jsem se, že i já ji při tomto pokusu využiji. Vyfotil jsem si tři šachovnice, které jsou nalepené na skříňce tedy jsou navzájem kolmé. Políčko šachovnice je čtvercové se stranou 47,2mm.

Obr. 20: Obrázek tří navzájem kolmých šachovnic

Začal jsem tím, že jsem si ve svém programu vytvořil model šachovnice obsahující všech 52 bodů (rohů políček), které každá ze šachovnic obsahuje. Následně jsem spustil kalibraci kamery pro každou ze tří šachovnic. Zjištěné rotace můžeme vidět v tabulce 3. Problém u kalibrace vznikl při hledání parametrů horní šachovnice, protože vidíme, že tato šachovnice je ze všech tří nejméně zkreslená byla přesnost velice razantně ovlivněna právě nepřítomností výraznější perspektivy.

Šachovnice	α [rad]	β [rad]	γ [rad]
Horní	0,478349	-0,54503	0,878824
Levá	2,483009	-0,6868	-1,5126
Pravá	-0,98209	-0,64275	0,797989

Tabulka 3: Rotace zjištěné pro šachovnice

Těmito hodnotami rotací jsem rotoval normálový vektor roviny modelu šachovnice $n = (0,0,1)$, a vypočítal úhel mezi výslednými zrotovanými vektory. Výsledky a přesnost zobrazuje tabulka 4.

Dvojice šachovnic	φ [rad]	Δ [rad]
Horní-Levá	1,723	0,153
Horní-Pravá	1,505	-0,065
Levá-Pravá	1,545	-0,026

Tabulka 4: Úhly mezi dvojicemi šachovnic a absolutní odchylka od skutečné hodnoty

Na těchto datech můžeme jasně vidět, že přesnost kalibrace u horní šachovnice byla opravdu nižší než u zbylých dvou. Vidíme, že levá a pravá mají mezi sebou odchylku v řádu setin radiánu. Ale i tato hodnota je o řád větší než výsledky z umělých dat. To se dalo vzhledem extrémně vysoké přesnosti u umělých dat předpokládat. Pro přesnější výsledek by zde dle mého názoru nestačili větší množství bodů jelikož 52 bodů mi přijde dostatečné, zřejmě bylo nutné přidat do modelu kamery i nějaké druhy zkreslení nebo poskytnout uživateli nějaký nástroj pro přesnější zadávání bodů na reálném obrázku formou např.: nějakého předzpracování obrázku jako je zaostření , zvýraznění hran nebo detekce přímek v obrazu. Protože pokusíme-li se pro co největší přesnost zadávání těchto bodů obraz hodně přiblížit vidíme že hrany jsou rozmazané a není možné zadat naprosto přesně polohu bodu.

9 ZÁVĚR

Tato práce navazuje na semestrální projekt a shrnuje tedy dva semestry práce. V průběhu těchto dvou semestrů se celý program a i směr práce několikrát změnily. Často jsem zahodil týdny práce, když jsem zjistil že se zabývám nesmyslem. Ale teď v závěru mohu napsat, že jsem se každý bod zadání pokusil splnit co nejlépe.

Hned na začátku své práce jsem si jako evoluční algoritmus vybral diferenciální evoluci, což byla volba, která se ukázala být jako velice vhodná. I co se týče návrhu fitness funkce jsem dle mého názoru zvolil správně. U první verze jádra programu trvalo ustálení populace při kalibraci kamery více jak půl minuty. Považuji za svůj velký úspěch, že jsem jádro programu optimalizoval tak, že výpočet netrvá ani sekundu.

Později jsem se zabýval vytvořením uživatelského rozhraní. Vytvořil jsem jednoduché a funkční uživatelské rozhraní, které lze použít pro ovládání jádra programu, a tedy pro kalibraci kamery.

Dále jsem ve své práci změřil přesnost mého řešení jak na umělých, tak částečně i na reálných datech. Na umělých datech vyšla přesnost velice dobře, na těch reálný přirozeně o něco hůře, ale domnívám se, že ani tato přesnost není špatná. Bezpochyby existují přesnější metody pro měření s využitím kamery, ale také vyžadují pro svou funkci lepší vybavení a hlubší znalost problému, než je tomu u metody popsané v mé práci.

Nakonec jsem si nechal využití procesoru grafické karty, a protože jsem neměl vhodnou grafickou kartu k dispozici doma, musel jsem program testovat v laboratoři. Nakonec se mi i povedlo program odladit a naměřit výsledky. Z těchto výsledků je naprosto jasně vidět, že nemá smysl používat GPU pro kalibraci kamery s malým počtem jedinců v populaci. U velkého počtu jedinců je vidět úspora až několika desetin vteřiny. Což, dle mého názoru, za nutnost mít grafickou kartu zvládající technologii CUDA nestojí. Na druhou stranu byla práce s technologií CUDA pro mě dost přínosná, naučil jsem se mnoho nového o této oblasti a jistě se v budoucnu setkám s problémem, kde bude mít využití GPU praktický přínos.

Tato práce mě bavila a mnoho jsem se při její tvorbě přiučil jak z oblasti počítačového vidění, tak i z oblasti tvorby odborných prací.

Literatura

[1] MAŘÍK, Vladimír. *Umělá inteligence*. 1. vyd. Praha: Academia, 2001, 328 s. ISBN 80-200-0472-6.

[2] MAŘÍK, Vladimír. *Umělá inteligence*. 1. vyd. Praha: Academia, 2003, 470 s. ISBN 80-200-1044-0.

[3] POLLEFEYS, Marc. The camera model. The University of nord Carolina [online]. 2002-11-22 [cit. 2012-01-08]. Dostupné z: http://www.cs.unc.edu/~marc/tutorial/node35.html

[4] ZELINKA, Ivan. *Umělá inteligence v problémech globální optimalizace*. 1. vyd. Praha: BEN - technická literatura, 2002, 189 s. ISBN 80-730-0069-5.

[5] MLČOCH, Tomáš. Výpočty pomocí grafických procesorů GPU - GPGPU. *Tojaj's blog* [online]. 15. Listopad 2009 [cit. 2012-05-20]. Dostupné z: http://tojaj.com/pocitace/35-programovani/55-vypocty-pomoci-gpu

[6] KOSEK, Jiří. Seriál o XML pro Softwarové noviny. *Vše o www* [online]. 2000 [cit. 2012-05-22]. Dostupné z: http://www.kosek.cz/clanky/swn-xml/index.html

[7] JÄHNE, Bernd, Horst HAUSSECKER a Peter GEISSLER. *Handbook of computer vision and applications*. San Diego: Academic Press, c1999, 3 v. in container. ISBN 012379773X3.

Seznam zkratek

f	ohnisková vzdálenost
p_x	velikost pixelu v ose x
p_y	velikost pixelu v ose y
α	velikost rotace kolem osy x
β	velikost rotace kolem osy y
γ	velikost rotace kolem osy z
X,Y,Z	translace v příslušných osach
D	počet parametrů problému řešeného diferenciální evolucí
NP	počet jedinců tvořících jednu populaci
CR	práh křížení nebo křížící konstanta
F	mutační konstanta
CPU	procesor počítače (*Central Processing Unit)*
GPU	procesor grafické karty (*Graphic processing unit)*
CUDA	technologie pro zpracování programu pomocí GPU (*Compute Unified Device Architecture*)

Seznam příloh

Příloha 1. CD