Vysoké učení technické v Brně
Fakulta strojního inženýrství
Ústav mechaniky těles, mechatroniky a biomechaniky

Bakalářská práce

Brno 2014

Bibliografická citace

KUMPÁN, P. Vytvoření 3D modelu prostředí pomocí senzoru Kinect. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2014. 25 s. Vedoucí bakalářské práce Ing. Michal Růžička.

Abstrakt

Tato práce se zabývá užitím senzoru Microsoft Kinect pro vytváření trojrozměrného modelu prostředí. Cílem práce je navrhnout, implementovat a otestovat metodu, který toto bude realizovat a porovnat výkon při implementaci pro procesor a pro grafickou kartu. V první části je popsáno zařízení Kinect, druhá část přibližuje fungování programu. Třetí část shrnuje dosažené výsledky.

Abstract

This thesis

Poděkování

Děkuji tomu že Lorem Ipsum má kořeny v klasické latinské literatuře z roku 45 před Kristem, což znamená, že je více jak 2000 let staré. Richard McClintock, profesor latiny na univerzitě Hampden-Sydney stát Virginia, který se zabýval téměř neznámými latinskými slovy, odhalil prapůvod slova consectetur z pasáže Lorem Ipsum. Nejstarším dílem, v němž se pasáže Lorem Ipsum používají, je

Čestné prohlášení

Prohlašuji na svou čest, že bakalářskou práci na téma *„Vytvoření 3D modelu prostředí pomocí senzoru Kinect“* jsem vypracoval samostatně, pod vedením vedoucího bakalářské práce pana Ing. Michala Růžičky a s použitím uvedených literárních a internetových zdrojů.

V Brně dne 5. 5. 2014 .....

Pavel Kumpán

Obsah

Úvod

Hlavním cílem práce bylo navrhnout, implementovat a otestovat metodu pro rekonstrukci prostředí za pomoci hloubkového senzoru Kinect. U částí systému kde to bylo vhodné a možné, byla metoda implementována jak ve verzi pro obvyklý procesor x86, tak i ve verzi s využitím možnosti výpočtu na grafické kartě.

V současné době se touto problematikou zajímají tři projekty:

- Kinect Fusion vyvinutý v Microsoft Research v roce 2011, použité algoritmy byly veřejně publikovány na konferencích o počítačovém vidění a zpracování obrazu. Kinect Fusion je společností Microsoft často používán pro efektní demonstraci možností Kinectu.
- KinFu je otevřený projekt založený na knihovně Point Cloud Library a algoritmech Kinect Fusion.
- ReconstructMe přestavuje komerční software vyvinutý rakouskou společností Profactor.

Kapitola 1

Zařízení Kinect

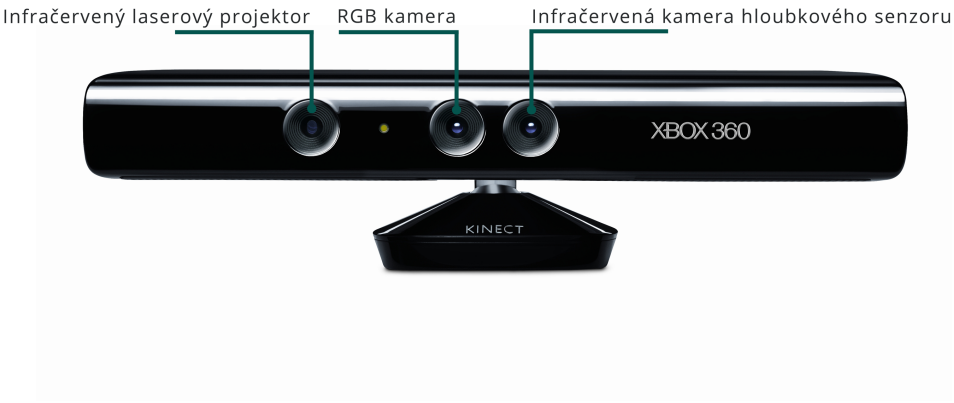
Kinect je hardwarový ovladač, který byl uveden jako doplněk ke hrací konzoli Xbox 360 společností Microsoft v listopadu roku 2010. SDK pro tvorbu aplikací třetích stran pro nekomerční užití Microsoft vydal v červnu 2011. V únoru 2012 byla uvolněna verze Kinectu přímo určená pro operační systém Windows a použití s PC. V roce 2014 se očekává uvedení nové verze Kinect 2.0 s vylepšenými parametry hardwaru.

1.1 Hardwarové parametry

Zařízení obsahuje RGB kameru o 8-bitovém maximálním rozlišení 1280×960 pixelů schopnou pracovat s frekvencí snímkování 12 Hz, případně lze použít rozlišení 640×480 a nižší s frekvencí 30 Hz. Dále se v zařízení nachází infračervený hloubkový senzor o maximálním rozlišení 640×480 pixelů a 11-bitové obrazové informaci s účinným snímáním objektů ve vzdálenosti 0,4 - 4 m. Zorný úhel je 43 ° vertikálně a 53 ° horizontálně. Maximální frekvence snímkování je 30 Hz.

Firmware Kinectu umožňuje detekci osob ve scéně i jednotlivých gest. Zařízení také obsahuje pole mikrofonů pro snímání zvuků okolí a je vybaveno servomotory pro autonomní změnu náklonu.

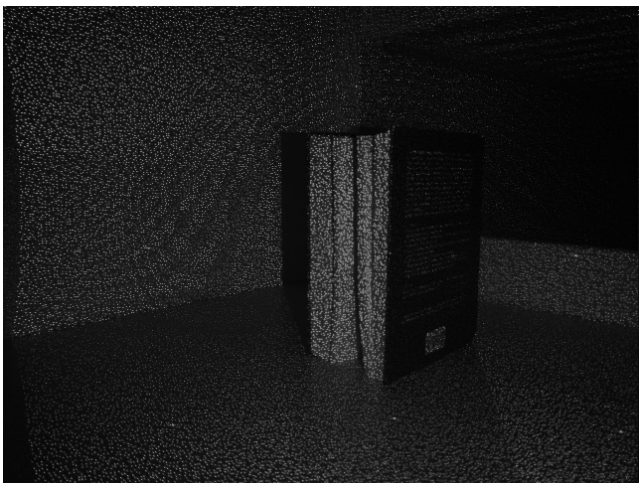
Pro tvorbu softwaru lze využít dvou nejrozšířenějších SDK. Oficiální Kinect SDK od společnosti Microsoft a komunitní otevřené SDK OpenKinect. Pro účely této práce bylo použito oficiální Kinect SDK.



Obrázek 1.1: Zařízení Kinect

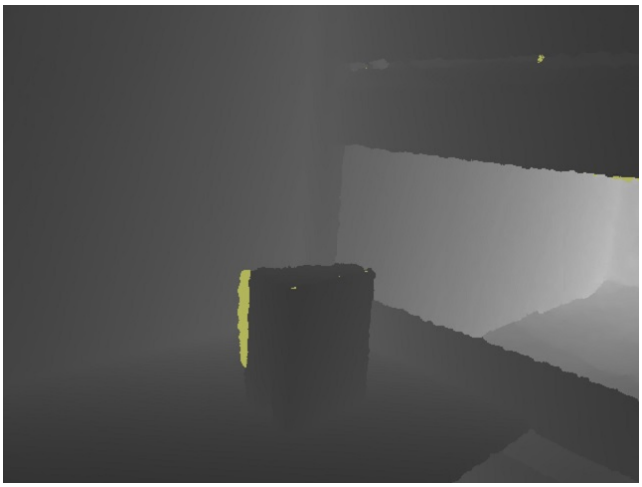
1.2 Hloubkový senzor

V této práci byl primárně využíván hloubkový senzor zařízení. Skládá se z infračerveného laserového emitoru a monochromatického CMOS senzoru. Díky použití infračerveného rozsahu spektra jej lze využívat relativně nezávisle na světelných podmínkách. Emitor na snímanou scénu stále promítá vzor bodů.

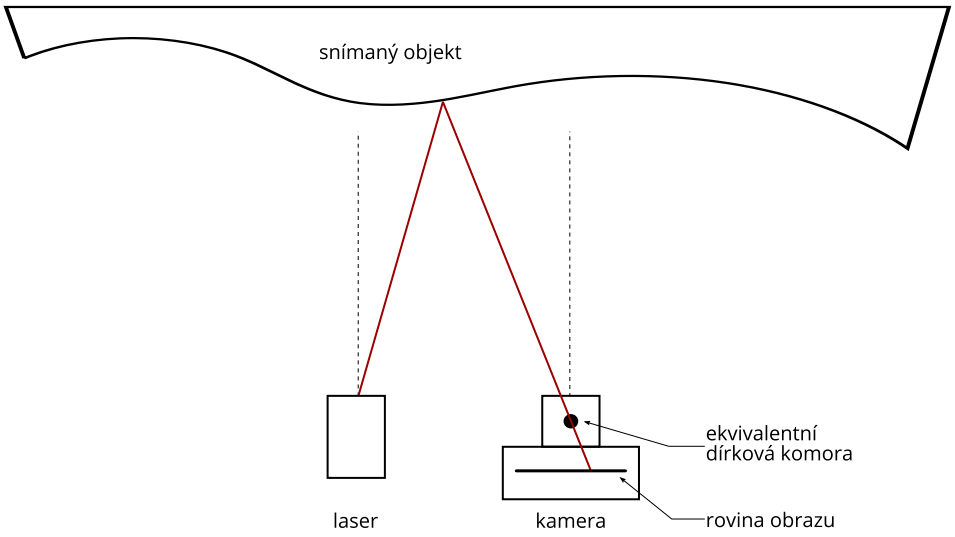


Obrázek 1.2: Snímek promítaného vzoru v infračerveném spektru

Body je následně snímán CMOS senzorem a z porovnání rozdílu mezi vzorem snímaným a promítaným jsou pomocí stereo triangulace vypočteny vzdálenosti pro jednotlivé pixely snímaného obrazu. Ty jsou poté do déle předány jako pole 16-bitových hodnot které tvoří takzvanou hloubkovou mapu. První tři bity obsahují informaci o hráči, nachází-li se bod na nějakém. Horních třináct bitů pak obsahuje vzdálenost bodu scény od senzoru v milimetrech.



Obrázek 1.3: Hloubkový snímek s hloubkou vyjádřenou pomocí odstínů šedi



Obrázek 1.4: Princip hloubkového senzoru

Kapitola 2

Popis metody

V této kapitole bude zevrubně popsána implementovaná metoda.
Metoda má několik částí

1. Příprava snímku,
2. slícování snímku s předchozími,
3. začlenění snímku do modelu.

Příprava snímku zahrnuje aplikaci filtrů na vstupní data hloubkové mapy za účelem vyhlazení a odstranění šumu. Hloubková mapa je následně převedena na množinu souřadnic bodů v prostoru (bodové mračno).
Připravené bodové mračno je poté třeba slícovat s body z předcházejících snímků.

2.1 Filtrování obrazových dat

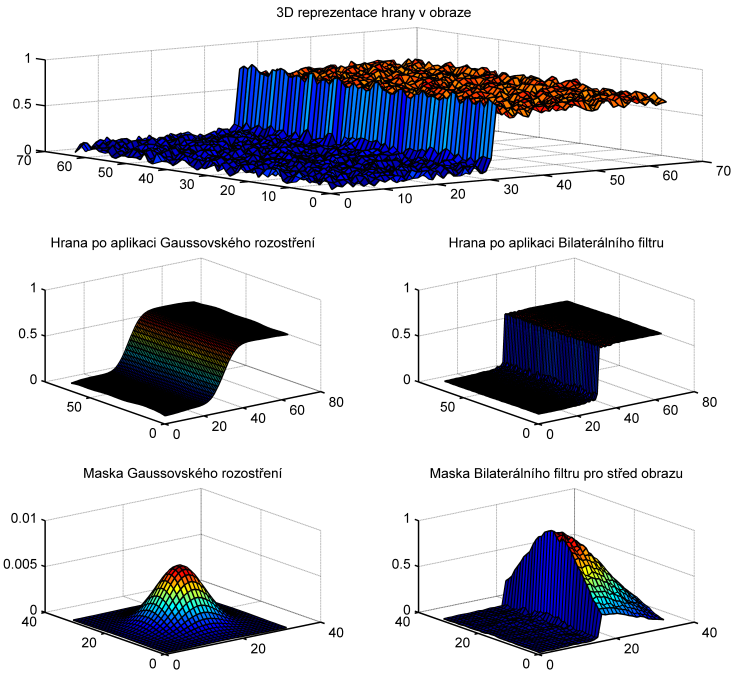
Nasnímaná data nejprve zpracujeme pomocí filtru, ukázalo se že tento krok výrazně zlepšuje rychlost konvergence a kvalitu výstupů dalších kroků algoritmu. Na hloubkový sken v pravidelném rastru se můžeme dívat jako na obrázek o jednom barevném kanálu, což nám umožňuje použít grafické filtry běžně používané pro zpracování obrazu. Naším hlavním cílem je redukce šumu. K tomu se často používá Gaussovo rozostření. Nedostatkem tohoto filtru je však to, že dochází ke ztrátě informace o hranách v obraze. Tento nedostatek řeší bilaterální filtr, který zachovává ostré hrany, ale odstraňuje šum.

2.2 Převod hloubkové mapy na body v prostoru

Data dodaná senzorem Kinect ve formě pole hloubek je pro další zpracování nutné převést na souřadnice prostorových bodů. K provedení výpočtu je senzor nahrazen zjednodušeným modelem dírkové komory. Předpokládá se, že každý reálný bod $\mathbf{p}$ o souřadnicích (x, y, z) je promítán na obrazovou rovinu ležící ve vzdálenosti f od dírky. Průmětem je bod $\mathbf{p}'$ o souřadnicích (u, v, d). Mezi souřadnicemi reálného bodu $\mathbf{p}$ a jeho obrazu na obrazové rovině $\mathbf{p}$ platí následující vztahy

$$\begin{aligned} \mathbf{p}(x) &= \frac{\mathbf{p}'(d) \cdot (\mathbf{p}'(u) - c_x)}{f_x} \\ \mathbf{p}(y) &= -\frac{\mathbf{p}'(d) \cdot (\mathbf{p}'(v) - c_y)}{f_x} \\ \mathbf{p}(z) &= \mathbf{p}'(d) \end{aligned} \tag{2.1}$$

kde (c_x, c_y) jsou souřadnice osy ohniska infračervené kamery Kinectu na snímku - tedy střed snímku. Souřadnice bodu $\mathbf{p}'$ - u a v představují pozici pixelech na hloubkové mapě.



Obrázek 2.1: Porovnání masky Gaussovského rozostření a Bilaterálního filtru a jejich vlivu na hranu

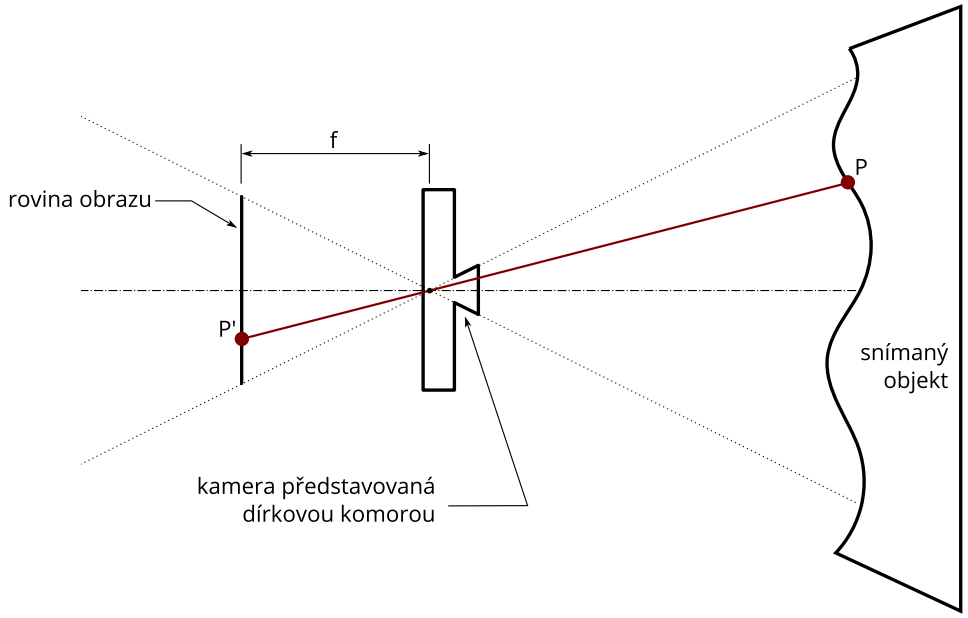
(a) Původní obrázek (b) Gaussovo rozostření (c) Bilaterální filtr

Obrázek 2.2: Porovnání Gaussova rozostření a bilaterálního filtru

Hodnota d reprezentuje hloubku, tedy reálnou vzdálenost mezi senzorem a bodem. Souřadnice bodu $\mathbf{p}$ v milimetrech se vztahují k lokálnímu souřadnému systému jehož počátek představuje právě dírka modelové dírkové komory.

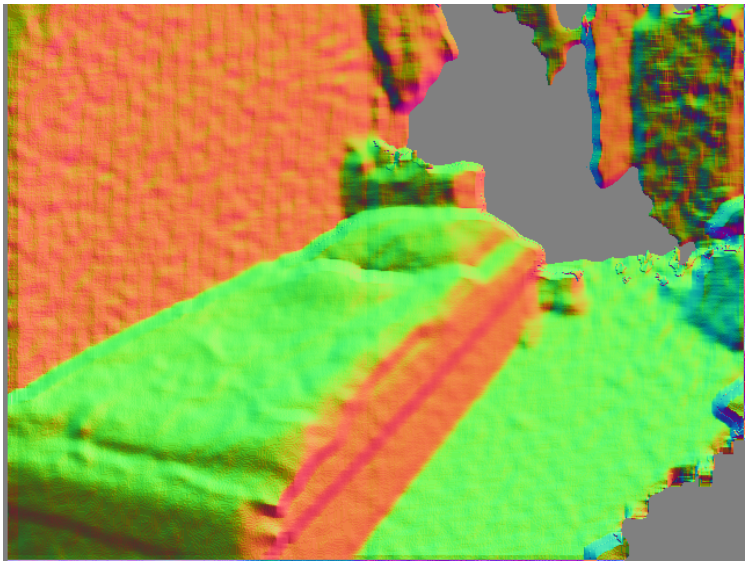
2.3 Vytvoření normálové mapy

K dalšímu zpracování je krom souřadnic jednotlivých bodů ve snímku třeba znát i hodnoty normálových vektorů ploch snímané scény v těchto bodech. Pro zjednodušení výpočtu určíme jako normálu plochy scény v daném bodě jako normalizovaný normálový vektor roviny určené tímto bodem a sousedními body v horizontálním a vertikálním směru.



Obrázek 2.3: Geometrický model Kinectu

$$\begin{aligned} \mathbf{u} &= \mathbf{p}_{i+1,j} - \mathbf{p}_{i,j} \\ \mathbf{v} &= \mathbf{p}_{i,j+1} - \mathbf{p}_{i,j} \\ \mathbf{n} &= \frac{\mathbf{u} \times \mathbf{v}}{|\mathbf{u} \times \mathbf{v}|} \end{aligned} \tag{2.2}$$



Obrázek 2.4: Normály scény znázorněné pomocí virtuálních barev

2.4 Spojování snímků

Souřadnice bodů nového snímku se vtahují k lokálnímu souřadnému systému s počátkem v poloze senzoru při pořízení snímku. Pro sestavení celkového modelu scény je však třeba

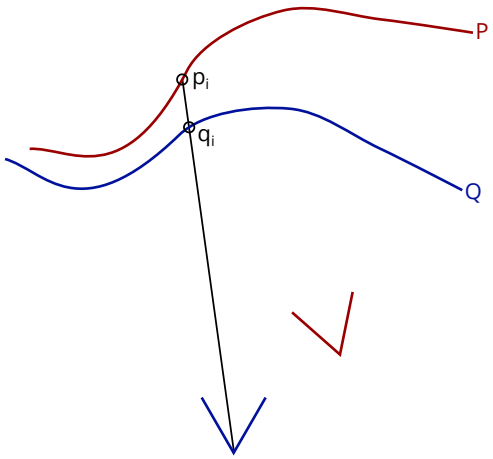
všechny snímky převést do jednotného globálního souřadného systému. Pro jednoduchost jej ztotožníme se souřadným systémem prvního zpracovávaného snímku.

Vztah mezi souřadnými systémy nového snímku P a předchozího snímku snímku Q lze vyjádřit pomocí rigidní transformace $[\mathbf{R}, \mathbf{t}]$, která se skládá z matice rotace $\mathbf{R}$ a vektoru translace $\mathbf{t}$.

Pro nalezení takovéto transformace byl použit algoritmus ICP (Iterative closest point - *iterace přes nejbližžší bod*). Jedná se o iterační algoritmus často používaný v 3D grafice pro spojování prostorových objektů. V práci byla použita varianta algoritmu známá jako "bod - rovina"("Point-to-plane").

Prvním krokem v algoritmu ICP je nalezení shodných bodů mezi snímky P a Q. Je známo několik efektivních strategií pro hledání takovýchto shod. Nejjednodušší možností je pro každý bod z množiny snímku P projít všechny body snímku Q a vybrat nejbližší. Metody tohoto typu jsou vhodné při zpracování snímků mezi kterými je velká transformace a dávají velmi dobré výsledky. Jejich nevýhodou je však značná časová složitost (při naivní implementaci $O(N^2)$, při použití stromových datových struktur $O(n \cdot logn)$).

Je-li však mezi oběma snímky transformace malá, lze s výhodou využít předpokladu, že hledaný nejbližší bod se na snímku Q nachází na podobných souřadnicích $[u, v]$ jako na snímku P. Metoda založená na tomto principu a použita v práci je označována jako projektivní asociace (*Projective data association*).



Obrázek 2.5: Princip projektivní asociace

Po nalezení párů bodů můžeme určit chybu mezi těmito snímky. Nejjednodušší způsob představuje použití Eukleidovské vzdálenosti mezi spárovanými body - tedy metrika bod - bod. Nicméně efektivnější možností je použít metriku bod - rovina. Při ní vzdálenost mezi body v páru nahradíme vzdáleností l mezi p a rovinou bodu p a jeho normály n.

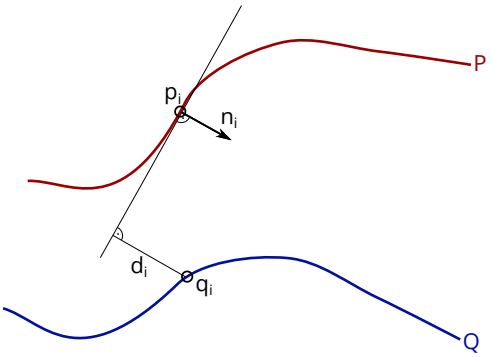
Chybu zarovnání můžeme definovat

$$e = \sum_i [((R) \cdot \mathbf{p}_i + \mathbf{t} - \mathbf{q}_i) \cdot \mathbf{n}_i]^2 \tag{2.3}$$

Snímky považujeme za zarovnané, je-li $e <= \epsilon$, kde ϵ je zvolená tolerance. Nalezení vhodné transformace $[\mathbf{R}, \mathbf{t}]$ je ve své podstatě optimalizační problém metody nejmenších čtverců.

Matice rotace $\mathbf{R}$ je tvořena kosiny a siny Eulerových úhlů, jedná se tedy o nelineární funkci. Předpokládáme-li však, že rotace mezi snímky je malá, můžeme ji aproximovat linearizovaným tvarem

$$\mathbf{R} \approx \begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix} \tag{2.4}$$



Obrázek 2.6: Metrika bod - rovina

Vektor translace má obecný tvar

$$\mathbf{t} = \begin{matrix} t_x & t_y & t_z \end{matrix}^T \tag{2.5}$$

jeho složky představují posunutí ve směru os souřadného systému.
Definujeme-li

$$\mathbf{c}_i = \mathbf{p}_i \mathbf{n}_i \tag{2.6}$$

Můžeme chybu zarovnání zapsat jako

$$e = \sum_i [(\mathbf{p}_i - \mathbf{q}_i) \cdot \mathbf{n}_i + \mathbf{t} \cdot \mathbf{n}_i + \mathbf{R} \cdot \mathbf{c}_i]^2 \tag{2.7}$$

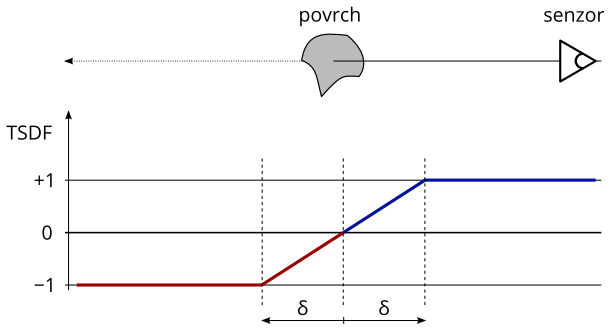
Hledanými parametry jsou úhly α, β, γ a posunutí t_x, t_y, t_z. Úlohu lze řešit pomocí metody nejmenších čtverců, která vede na řešení lineární soustavy

$$\sum_i \begin{pmatrix} c_{i,x}c_{i,x} & c_{i,x}c_{i,y} & c_{i,x}c_{i,z} & c_{i,x}n_{i,x} & c_{i,x}n_{i,y} & c_{i,x}n_{i,z} \\ c_{i,y}c_{i,x} & c_{i,y}c_{i,y} & c_{i,y}c_{i,z} & c_{i,y}n_{i,x} & c_{i,y}n_{i,y} & c_{i,y}n_{i,z} \\ c_{i,z}c_{i,x} & c_{i,z}c_{i,y} & c_{i,z}c_{i,z} & c_{i,z}n_{i,x} & c_{i,z}n_{i,y} & c_{i,z}n_{i,z} \\ n_{i,x}c_{i,x} & n_{i,x}c_{i,y} & n_{i,x}c_{i,z} & n_{i,x}n_{i,x} & n_{i,x}n_{i,y} & n_{i,x}n_{i,z} \\ n_{i,y}c_{i,x} & n_{i,y}c_{i,y} & n_{i,y}c_{i,z} & n_{i,y}n_{i,x} & n_{i,y}n_{i,y} & n_{i,y}n_{i,z} \\ n_{i,z}c_{i,x} & n_{i,z}c_{i,y} & n_{i,z}c_{i,z} & n_{i,z}n_{i,x} & n_{i,z}n_{i,y} & n_{i,z}n_{i,z} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ t_x \\ t_y \\ t_z \end{pmatrix} = -\sum_i \begin{pmatrix} c_i, x(p_i - q_i) \cdot n_i \\ c_i, y(p_i - q_i) \cdot n_i \\ c_i, z(p_i - q_i) \cdot n_i \\ n_i, x(p_i - q_i) \cdot n_i \\ n_i, y(p_i - q_i) \cdot n_i \\ n_i, z(p_i - q_i) \cdot n_i \end{pmatrix} \tag{2.8}$$

Vzhledem k tomu, že matice je symetrické, můžeme k řešení efektivně využít LU nebo QR rozklad. Získaný vektor **x** je tvořen Eulerovými úhly a složkami translačního vektoru.

2.5 Objemová reprezentace dat

Známe-li transformaci nového snímku vůči globálního systému, můžeme jej začlenit do celkového modelu scény. Ten je reprezentován formou objemových dat. Model si můžeme představit jako krychli obsahující část snímaného prostoru. Celá krychle je pravidelně rozdělena na malé krychlové podoblasti - voxely. Vzhledem k velké datové náročnosti takovéhoto způsobu reprezentace modelu je oblast, která do něj lze zachytit silně omezena. V práci byla použita krychle o rozměrech $3m \times 3m \times 3m$, která byla rozdělena na $512 \times 512 \times 512$ voxelů. Každý voxel obsahuje dvě hodnoty. Hodnotu vzdálenostní funkce a její váhu. Oříznutá znaménková funkce vzdálenosti (truncated signed distance function) reprezentuje vzdálenost mezi středem buňky a povrchem předmětu.



Obrázek 2.7: Průběh TSDF

Hodnota 0 znamená, že hrana prochází přímo středem. Hodnota −1 reprezentuje buňky uvnitř předmětu, kterými žádná hrana neprochází. Hodnota +1 analogicky označuje buňky vně předmětu bez hrany. Hodnoty $(0, 1)$

−1	−1	−1	−1	−1	−1	−0,1	1	1	1	1	1
−1	−1	−1	−1	−1	−1	0,2	1	1	1	1	1
−1	−1	−1	−1	−1	−0,2	0,9	1	1	1	1	1
−1	−1	−1	−1	−1	0,8	1	1	1	1	1	1
−1	−1	−1	−1	−1	0,8	1	1	1	1	1	1
−1	−1	−1	−1	−0,6	0,9	1	1	1	1	1	1
−1	−1	−1	−0,6	0,8	1	1	1	1	1	1	1
−1	−1	−0,7	0,9	1	1	1	1	1	1	1	1
−1	−1	−0,7	0,6	0,9	1	1	1	1	1	1	1
−1	−1	−1	−1	−0,7	0	0,9	1	1	1	1	1
−1	−1	−1	−1	−1	−1	−0,7	0,7	1	1	1	1
−1	−1	−1	−1	−1	−1	−1	0	1	1	1	1

Obrázek 2.8: 2D křivka popsána pomocí skalárního pole hodnot TSDF

2.6 Získání hloubkového snímku z objemových dat

V podkapitole **??** byl uveden postup pro výpočet transformace nového snímku vůči předchozímu snímku. Pokud bychom použili předchozí snímek z Kinectu, došlo by poměrně brzy k akumulaci drobných chyb v zarovnání. Tento relativní způsob zarovnávání je tedy reálně nepoužitelný.

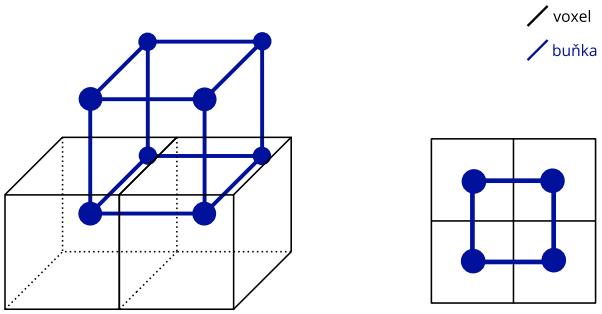
První možností, jak relativní způsob nahradit je použít absolutní zarovnávání vůči úplně prvnímu přijatému pro všechny nové snímky. To by však bylo použitelné pouze pro několik

málo snímků málo transformovaných oproti prvnímu snímku.

Jako nejrozumnější možnost se jeví použít relativní způsob zarovnávání, ale místo reálně přijatého předchozího snímku vytvořit virtuální snímek z objemového modelu pro předchozí model kamery. Pro tento úkol je mimořádně vhodná metoda trasování paprsku (ray casting).

2.7 Generování trojúhelníkové sítě

Pro vykreslení modelu jsou objemová data převedena na polygonovou síť reprezentující povrch. K vytvoření polygonové trojúhelníkové sítě je znám poměrně efektivní a jednoduchý algoritmus "pochodující kostky"(marching cubes). Kostky neboli buňky jsou tvořeny z 8 sousedících voxelů objemové mřížky. Její rohy jsou vertexy vypočtené jako střed voxelu.



Obrázek 2.9: Vztah mezi buňkou a voxely

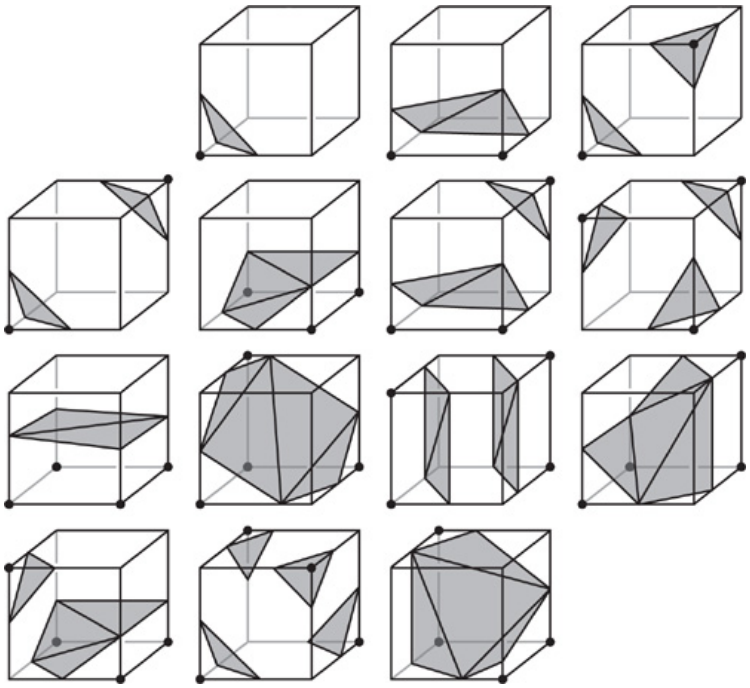
Počet buněk, nacházejících se na rozhraní mezi modelem tělesa a okolím je ve srovnání s celkovým počtem objemových buněk relativně malý. Je tedy výhodné algoritmus aplikovat pouze na takovéto buňky. K tomu byla použita jednoduchý metoda, která projde všechny buňky a na výstup zapíše indexy těch, kde je hodnota $TSDF \in (-1; 1)$. Seznam indexů buněk na rozhraní slouží jako jeden z parametrů algoritmu marching cubes.

Algoritmus určí, jakým způsobem plocha prochází buňkou a poté přejde ("pochoduje") k další kostce. K určení řezu, který plocha v buňce vytváří, nejprve určíme, které rohy buňky leží před a které za plochou. Vertexy před plochou označíme bitovou 1, vertexy za plochou označíme 0. Z bitů pro jednotlivé vertexy sestavíme 8-bitové číslo - index buňky. Protože je buňka tvořena 8 vrcholy, existuje $2^8 = 256$ možných topologických variant řezu. Jednotlivé varianty jsou předem známy a uloženy ve statické tabulce trojúhelníků. Pro přístup k topologii řezu v tabulce trojúhelníků slouží právě 8-bitový index buňky.

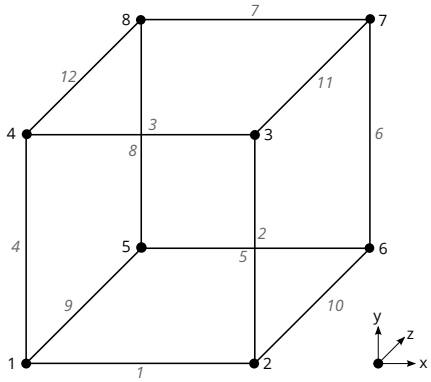
Topologie řezu popisuje definuje hrany, které obsahují vrchol některého trojúhelníku. K výpočtu reálných souřadnic vrcholů trojúhelníku je použita lineární interpolace za pomoci hodnot TSDF v jednotlivých rozích.

```
__constant int triangle_map[4096] =
{
    -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
//  trojúhelník neprotíná žádnou hranu
   0, 8, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
//  trojúhelník protíná hrany číslo 0, 8 a 3
   0, 1, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
//  trojúhelník protíná hrany číslo 0, 1, a 9
   1, 8, 3, 9, 8, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
//  první trojúhelník protíná hrany číslo 1, 8, a 3, druhý hrany 9, 8, a 1
   ...
```

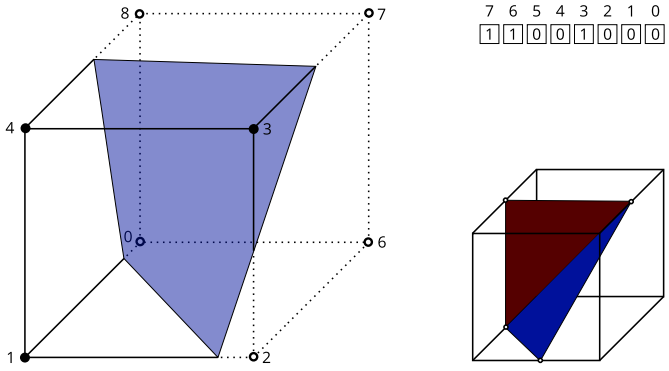
Kód 2.1: Část tabulky trojúhelníků



Obrázek 2.10: Konfigurace povrchových polygonů v buňce dle vertexů ležících mimo těleso



Obrázek 2.11: Číslování hran a vrcholů buňky



Obrázek 2.12: Buňka protnutá plochou, její index a příslušná topologie

Kapitola 3

Implementace

Navržená metoda byla implementována pod platformou .NET s použitím jazyka C#. Pro práci se zařízením Kinect bylo využito *Kinect for Windows SDK* , k maticovým výpočtům posloužila knihovna math.net. Kód pro paralelní zpracování jsem napsal pomocí OpenCL, propojení s .NET aplikací realizuje knihovna Cloo.

Podrobný popis implementace výše představené metody je mimo rozsah této práce. Zaujímá-li se čtenář o tuto oblast blíže, doporučuji nahlédnout přímo do zdrojových kódů, které jsou k práci přiloženy případně lze hlavní vývojovou vět nalézt také na GitHubu: `https://github.com/PavelKumpan/KinectDepth`. V této kapitole jsou alespoň zevrubně přiblíženy zajímavé nebo méně obvyklé části projektu.

3.1 Paralelizace

3.1.1 Krátký úvod do výpočtů na grafických kartách

I přes rapidní nárůst výkonu běžně dostupných procesorů v posledních letech jsou zde stále oblasti výpočtů u kterých je výpočetní náročnost úlohy na běžném procesoru mimo únosnou mez. Společným rysem velké části těchto úloh je to, že se jedná o úlohy paralelní, tedy takové, které je možné řešit na více výpočetních jednotkách zároveň a do jisté míry nezávisle. Přestože jsou vícevláknové procesory dnes již standardem, nejsou pro takovéto výpočty zcela ideálně použitelné. Počet vláken je zde stále relativně malý (v jednotkách až desítkách), kvůli čemu nemůžou konkurovat z principu paralelně navrženým grafickým kartám se stovkami až tisíci výpočetních jader.

Grafické karty nebyly dříve navrhovány tak, aby umožnily přímé programování.

Společnost nVidia uvedla své řešení pro programování nad grafickou kartou v roce 2007 pod označením CUDA. Konkurenční technologie Stream od AMD (ATI) spatřila světlo světa roce 2008. Nutnost standardizace vedla k vytvoření jednotného rozhraní a jazykové specifikace OpenCL spravovaného konsorciem Khronos. To je nyní podporováno grafickými kartami nVidia, AMD a Intel.

3.1.2 Struktura programu na GPU

Program který využívá grafickou kartu lze v zásadě rozdělit na tři části:

- *kernel*,
- *data*,
- *hostitelská aplikace*.

Kernel je podprogram v jazyce OpenCL C který běží na grafické kartě a realizuje potřebné výpočty. Kernely se zapisují do souborů s příponou **.cl*.

```
__kernel void vector_add_gpu (__global const float* src_a,
                              __global const float* src_b,
                              __global float* res,
                             const int num)
{
   const int idx = get_global_id(0);
   if (idx < num)
      res[idx] = src_a[idx] + src_b[idx];
}
```

Kód 3.1: Jednoduchý kernel pro součet dvou vektorů

Data se kterými je výpočet prováděn je třeba nejprve zkopírovat z operační paměti RAM do paměti grafické karty. Výsledky výpočtů jsou poté opět překopírovány z paměti GPU do RAM. *Hostitelská aplikace* se stará o agendu. Jedná se o běžnou aplikaci běžící na CPU, která pomocí rozhraní OpenCL načítá, kompiluje a spouští kernely, kopíruje data a poskytuje uživatelské rozhraní.

Kapitola 4

Výsledky měření

	Typ	**Procesor**	**Kapacita operační paměti**	**Dedikovaná grafická karta**
počítač 1	notebook	Intel core i7 3632QM	8 GB	nVidia GeForce GT740M
počítač II	notebook	Intel core i3 370M	4 GB	nemá

Tabulka 4.1: Parametry počítačů použitých k testům

Kapitola 5

Závěr

Seznam obrázků

Seznam zdrojových kódů

Seznam tabulek