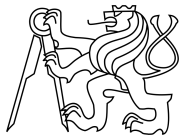
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Vytváření hloubkové mapy pro 3D zobrazení
Student:	Bc. Miroslav Lhoťan
Vedoucí:	Dr. Ing. Sven Ubik
Studijní program:	Informatika
Studijní obor:	Počítačové systémy a sítě
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Proveďte rešerši metod vytváření hloubkové mapy pro 3D zobrazení z 2D obrazových zdrojů pro levé a pravé oko. Zvolte metodu vhodnou pro rychlé zpracování obrazových sekvencí v reálném čase. Analyzujte možnost paralelního zpracování ve více vláknech. Implementujte tuto metodu v software. Navrhněte architekturu distribuovaného systému pro vzdálenou prezentaci 3D obrazu přes počítačovou síť v reálném čase. Ověřte funkčnost pomocí dvou kamer a autostereo 3D monitoru. Vyhodnoťte rychlost zpracování, stabilitu a kvalitu hloubkové mapy pro různé scény. Potřebné technické vybavení je k dispozici v Síťové multimediální laboratoři fakulty.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 5. prosince 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Diplomová práce

Vytváření hloubkové mapy pro 3D zobrazení

Bc. Miroslav Lhoťan

Vedoucí práce: Dr. Ing. Sven Ubik

2. května 2016

Poděkování

Poděkovat bych chtěl zejména svému vedoucímu práce panu Dr. Ing. Svenu Ubikovi a panu Ing. Zdeňku Trávníčkovi za veškerou ochotu a pomoc při psaní práce a implementaci. Dále pak musím poděkovat své sestře Anně, která mi pro testování a měření zapůjčila výkonný počítač, kterým já bohužel nedisponuji.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 2. května 2016 .....

Odkaz na tuto práci

Lhoťan, Miroslav. *Vytváření hloubkové mapy pro 3D zobrazení.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

V této práci jsou popsány základy problematiky konstrukce hloubkové mapy ze stereoskopického obrazu. Dále jsou zde popsány principy čtyř algoritmů řešících tuto úlohu a jejich implementace. Implementace je realizována formou modulů multimediální knihovny libyuri, a to s použitím knihovny OpenCV a platformy CUDA. Implementované moduly jsou pak otestovány z hlediska kvality jejich výstupů.

Klíčová slova stereoskopie, hloubková mapa, mapa disparit, OpenCV, libyuri, CUDA

Abstract

This thesis describes basics of depth-map construction from provided stereoscopic images. It contains description of four algorithms designed for this task and describes their implementation. Implementation is carried out in form of modules for multimedia library libyuri, with use of the OpenCV library and CUDA platform. Implemented modules are tested concerning the quality of their outputs.

Keywords stereoscopy, depth-map, disparity map, OpenCV, libyuri, CUDA

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Hlavním cílem této práce je nalezení způsobu jak ze stereoskopického obrazu získaného ze dvou vedle sebe umístěných kamer získat hloubkovou mapu. Tato mapa je představována černobílým snímkem, u něhož jas jednotlivých pixelů indikuje jejich vzdálenost od pozorovatele. Tato reprezentace prostorových dat má oproti použití běžného stereoskopického obrazu jisté výhody. První takovou výhodou, je fakt, že černobílá hloubková mapa je v porovnání s běžným videem mnohem lépe komprimovatelná a při přenosu nepotřebuje tak široké pásmo jako standardní snímek. Jako další motivaci pro tuto můžeme uvést skutečnost, že některá zařízení pro svou funkci hloubkovou mapu přímo vyžadují. Pro získání hloubkové mapy snímané scény je nutné provést párování odpovídajících bodů v rámci obou snímků, neboli takzvaný Stereo Matching.

Jako první bude provedena analýza problematiky spojené s prostorovým viděním a budou popsány obecné kroky, které jsou pro konstrukci hloubkové mapy třeba. Tedy kalibrace snímacích zařízení a rektifikace obrazu, které je nutné provést před samotným párováním pixelů v rámci obou snímků.

Pro tento úkol již existuje značné množství algoritmů. Další částí této práce tedy bude provedení rešerše již existujících algoritmů a případně hotových, již použitelných řešení. Bude popsán způsob jejich fungování a rovněž bude zhodnocena jejich vhodnost pro použití při řešení hlavního problému. Je totiž třeba použít algoritmus dostatečně efektivní na zpracování živého video vstupu, takže bude třeba brát ohled například na možnost paralelního zpracování a celkové škálování při zvyšování objemu zpracovávaných dat. Důležitá je rovněž stabilita získávaných výsledků v čase. Stabilitou rozumíme skutečnost, že změny ve vypočítaných hloubkách odpovídají pouze skutečným změnám v prostoru snímané scény a nebude docházet k jiným nežádoucím změnám. Všechny popsané způsoby budou pro toto použití v rámci možností vyzkoušeny a následně bude vybrán ten nejvhodnější.

Vybraný způsob tvorby hloubkové mapy pak bude použit pro implementaci distribuovaného systému, použitelného pro zpracování a následné vzdálené zobrazení výsledného obrazu. Tato část bude zpracována jako moduly pro

multimediální knihovnu libyuri s tím, že bude využito jejích již existujících součástí. Pro kalibraci a další předzpracování obrazu, bude rovněž použito knihovny OpenCV. Výsledné řešení pak bude otestováno na zařízeních, která jsou k dispozici v Síťové multimediální laboratoři (SAGElab).

Kapitola 1

Analýza problému

První částí této práce je vysvětlení základních principů prostorového vidění a kroků nutných pro správnou konstrukci hloubkové mapy. V tomto případě je hloubková mapa vypočítána ze dvou obrazů, které byly pořízeny dvěma kamerami snímajícími stejnou scénu ze dvou různých úhlů. Pokud tedy hledáme polohu nějakého bodu v prostoru, lze ji v ideálním případě vypočítat z jeho polohy v rámci obou obrazů a z parametrů kamer (jejich polohy, vzájemné vzdálenosti a ohniskové vzdálenosti). Ve skutečnosti je ovšem celá věc problematičtější, protože polohu bodu v rámci obrazů v kamerách zkreslují další faktory. To mohou být například zkreslení způsobená čočkami nebo, v případě digitálních kamer, diskrétní povaha snímače.

Abychom ovšem mohli potřebné informace o jednotlivých bodech scény získat, je nutné nejprve provést mezi oběma snímky jejich spárování. A právě toto párování je nejdůležitějším krokem v celé konstrukci hloubkové mapy. V této kapitole tedy budou dále popsány kroky, které je pro úspěšné párování třeba provést a které jsou uvedeny v dokumentaci knihovny OpenCV. Tato knihovna bude posléze použita pro implementaci těchto společných přípravných kroků.[3][4]

1.1 Kalibrace

Prvním důležitým krokem, který je třeba provést, je kalibrace kamery. Ta je důležitá k tomu, abychom mohli provádět porovnání skutečných rozměrů s rozměry předmětů na snímku a abychom mohli dále vycházet z teoretického modelu dírkové kamery. Za prvé je třeba získat informace o vnitřním nastavení kamery, zejména ohniskovou vzdálenost a souřadnice středu kamery. Vnitřní nastavení kamery je pak představováno tzv. maticí kamery (1.1):

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1.1}$$

Zde hodnoty f_x a f_y představují ohniskové vzdálenosti a c_x a c_y je vektor posunutí středu snímače, oproti jeho průsečíku s osou čočky. Pro ohniskovou vzdálenost pak máme dva údaje pro případ, že pixel na snímači nemá zcela přesně čtvercový tvar.[1, 373]

S pomocí této matice je možné přepočítat souřadnice bodu v prostoru na souřadnice bodu v rámci snímku získaného z kamery. Rovnice pro tento výpočet pak ve zjednodušené podobě vypadá následujícím způsobem.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A[R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1.2}$$

V tomto případě je s koeficient zvětšení, u a v jsou souřadnice bodu obrazu v pixelech a X, Y a Z jsou souřadnice bodu v rámci prostoru. $R|t$ je potom matice zobrazení, která převádí souřadnice bodu do souřadnic vztažených k pozici kamery. Tyto hodnoty jsou pro jedno nastavení kamery neměnné a po jejich získání je lze uložit a dále používat.[1, 386]

Jak již bylo zmíněno, obraz získaný z kamery je při průchodu čočkou zkreslen. Zde uvedeme dva hlavní druhy projevujících se zkreslení. První je takzvané radiální zkreslení. To má za následek, že linie, které jsou ve skutečnosti rovné, se na obraze zobrazí jako křivky (známý efekt rybího oka). Tato skutečnost pak velmi komplikuje párování bodů mezi oběma obrazy. Samotné zkreslení pak lze popsat následujícími rovnicemi [5][4].

$$\begin{aligned} x_d &= x(1 + k_1 r^2 + k_2 r^4 \ldots) \\ y_d &= y(1 + k_1 r^2 + k_2 r^4 \ldots) \end{aligned} \tag{1.3}$$

Hodnoty k_n odpovídají koeficientům zkreslení a r je vzdálenost bodu od středu snímku. Pro zpřesnění je možné přidávat další stupně k, knihovna OpenCV pak používá pro výpočet tohoto zkreslení stupně 3[4].

Druhým typem zkreslení je pak zkreslení tangenciální, které je způsobeno tím, že plocha snímače nesvírá s osou čočky přesně pravý úhel. Toto má za efekt zkreslení perspektivy, takže například čtverec se pak na snímku může jevit jako lichoběžník. Tangenciální zkreslení lze popsat pomocí těchto dvou rovnic:

$$\begin{aligned} x_d &= x + [2p_1 xy + p2(r^2 + 2x^2)] \\ y_d &= y + [p_1(r^2 + 2y^2) + 2p_2 xy] \end{aligned} \tag{1.4}$$

Stejně jako v předchozím případě, lze přidávat další stupně p ale OpenCV v tomto případě používá stupně 2. Všechny parametry zkreslení jsou pak stejně jako parametry matice kamery pro jedno nastavení pevné a rovněž je lze uložit pro pozdější použití.

Pokud budeme chtít výše uvedené parametry získat a provádět pomocí nich korekce, můžeme opět použít funkce knihovny OpenCV. Pokud těmto funkcím

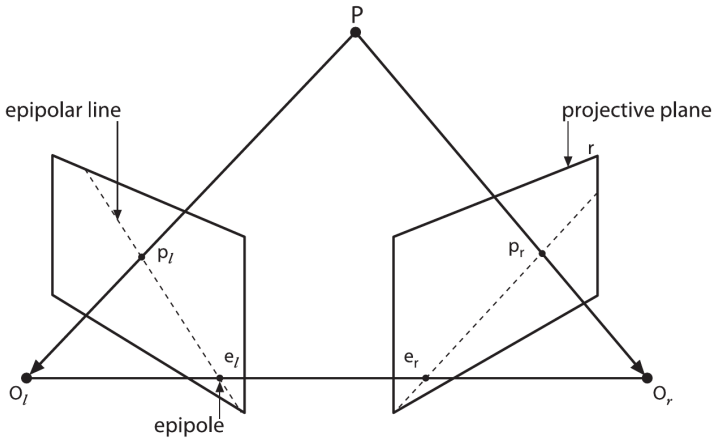
poskytneme snímky obsahující kalibrační obrazec, což je v případě OpenCV šachovnice přesně daných rozměrů, všechny potřebné parametry budou těmito funkcemi pomocí řešení soustav lineárních rovnic vypočítány a my je můžeme dále používat.

OpenCV dokonce nabízí i funkce pro společnou kalibraci dvou kamer, která pomocí rozpoznání polohy šachovnice určí jejich vzájemnou polohu. S pomocí této informace pak poskytne i informace další, které budou potřebné v dalším kroku.

1.2 Rektifikace

Poté co je obraz z obou kamer zbaven nežádoucích zkreslení lze přistoupit k dalšímu důležitému kroku, který se nazývá rektifikace. Tato operace je důležitá pro urychlení párování bodů mezi oběma snímky. Zajistí totiž horizontální zarovnání obou snímků[1, 427]. Díky tomu pak stačí korespondující body hledat pouze na stejných řádcích obou snímků. Toho bude dosaženo transformací obou snímků do podoby odpovídající stavu, ve kterém byly středy obou kamer ve stejné výšce a jejich osy rovnoběžné. Fyzicky dosáhnout tohoto stavu je totiž velmi obtížné, ale pomocí těchto transformací je možné tyto nepřesnosti velmi dobře odstranit.

Pro pochopení těchto operací je však třeba vysvětlit principy takzvané *epipolární* geometrie, která popisuje vztahy při snímání scény více kamerami. Základní pojmy budou ilustrovány na následujícím obrázku:



Obrázek 1.1: Epipolární rovina Zdroj: [1, 420]

Na obrázku vidíme dvě kamery, které jsou určeny jejich rovinami projekce a jejich ohnisky (O_l a O_r). Poté zde existuje bod P, což je bod v prostoru a p_l a p_r jsou jeho průměty v levé a pravé kameře. Dále vidíme body e_l a e_r, které jsou definovány jako obrazy ohnisek druhých kamer. E_l je tedy obrazem

bodu O_r promítnutým levou kamerou a obráceně. Rovina v prostoru, která je určena body O_l, O_r a P, se pak nazývá *epipolární rovina* a body e_l a e_r *epipóly*. Přímky $p_l e_l$ a $p_r e_r$ se označují jako epipolární přímky.

Pro pochopení účelů epipólů si povšimněme skutečnosti, že všechny body v prostoru, nacházející se na přímce $O_l P$, se v levé kameře zobrazí do bodu p_l. Stejně tak je tomu v případě pravé kamery. Celá přímka $O_r P$ se zobrazí do bodu p_r. Důsledkem této skutečnosti je pak fakt, že přímka $O_r P$ se v levé kameře zobrazí právě přesně do epipolární přímky $p_l e_l$.

Nyní tedy můžeme s jistotou tvrdit, že pokud máme v levé kameře obraz nějakého bodu v prostoru (X), jeho obraz v pravé kameře se nachází na epipolární přímce určené ohnisky a bodem X. Tento fakt nám tedy významně usnadňuje párování bodů mezi obrazy, neboť nyní postačuje prohledat pouze odpovídající epipolární přímku. To jednak snižuje složitost výpočtu, ale také velmi omezuje možnost výskytu chybného párování.

Dalším zajímavým důsledkem je skutečnost, že je zachováno horizontální pořadí. Pokud tedy máme body A a B, oba tyto body jsou v obou obrazech viditelné a A je v jednom z obrazů výše než B, pak platí že A je výše než B i v obrazu druhém.

V tuto chvíli již známe význam epipolárních přímek pro párování bodů ve snímcích, zbývá tedy provést jejich horizontální zarovnání. K tomuto úkonu jsou nutné dva prostředky. Těmi jsou dvě matice zobrazení, které nazýváme *esenciální* (essential) a *fundamentální* (fundamental) matice. Označujeme je písmeny E a F. Matice E pak obsahuje informace o vzájemné pozici obou kamer, jedná se tedy o zobrazení translace a rotace. Matice F obsahuje jednak stejné informace jako E a navíc obsahuje i vnitřní charakteristiky kamer. Určuje tak vztah obou kamer v pixelových jednotkách.[1, 421]

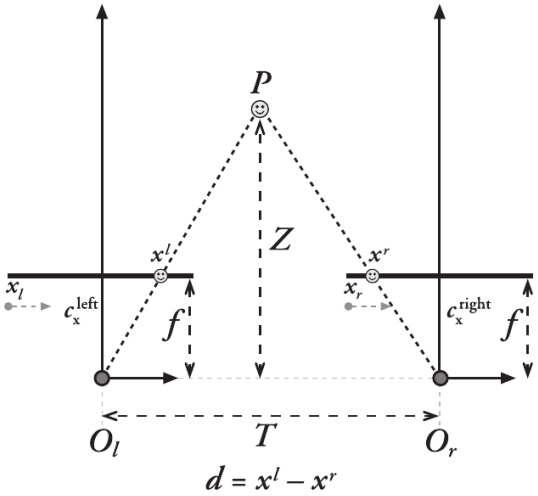
Pro ilustraci významu matice E budeme uvažovat ohnisko levé kamery jako počátek soustavy souřadnic a P_l jsou souřadnice bodu P vzhledem k tomuto počátku. Matice E pak funguje jako matice zobrazení, která nám souřadnice P_l transformuje na souřadnice s počátkem v ohnisku druhé kamery. Matice F funguje podobným způsobem, pouze, jak již bylo řečeno, přidává interní charakteristiky kamer.

Knihovna OpenCV nám nabízí funkci, která je již připravena kalibrovat dvě kamery najednou a je schopna vypočítat přímo matice E a F. S pomocí těchto matic pak další funkce obsažené v knihovně mohou připravit zobrazení, které dokáže dané snímky transformovat do žádoucího stavu s rovnoběžnými epipolárními přímkami. Pro tento okamžik zde již nebudeme uvádět další detaily, o těchto funkcích se však ještě zmíníme v části, která se věnuje implementaci.

1.3 Mapa disparit

V tuto chvíli již máme oba snímky správně připravené pro provedení párování jejich bodů. Tato operace je nejkritičtější částí celého procesu a budeme se jí podrobně věnovat v další kapitole. Nyní nás ovšem bude zajímat její výsledek a jak tento výsledek interpretovat.

Jako výsledek v ideálním případě získáme pro body v levém snímku pozice odpovídajících pixelů ve snímku pravém. Pokud vyjdeme z principu, na jakém funguje prostorové vidění, můžeme říci, že čím blíže je pozorovaný bod pozorovateli, tím dále se jeho obrazy nacházejí, pokud bychom oba snímky překryli. Jsme tedy v tuto chvíli schopni vypočítat vzdálenost bodu od snímací soustavy.



Obrázek 1.2: Ilustrace odlišných souřadnic objektu Zdroj: [1, 416]

Pokud tedy ke každému bodu přiřadíme vzdálenost jeho korespondujícího bodu z druhého snímku získáme takzvanou *mapu disparit*. Tu potom lze jednoduchým způsobem vizualizovat. Vzdálenost můžeme interpretovat jako stupně šedi a výsledný obrázek nám tak podá vizuální informaci o prostorovém rozmístění objektů ve scéně. Tento obrázek rovněž může sloužit i jako hloubková mapa pro hlavní cíl této práce.

Pokud získáme mapu disparit vzhledem k levému i pravému snímku, můžeme pak provést kontrolu konzistence a zjistit v jakých oblastech nebylo možné párování provést kvůli překryvům objektů. Pixel mapy disparit p je neplatný v případě, že rozdíl jeho disparity v levé mapě a disparity odpovídajícího bodu v pravé mapě je vyšší než stanovená hranice. V opačném případě je pixel platný.

V tomto okamžiku již máme popsány teoretické základy tvorby hloubkové mapy. V následující kapitole tedy popíšeme základní principy, kterými

lze k párování bodu mezi snímky přistoupit a představíme několik konkrétních algoritmů. Mezi nimi provedeme srovnání a ohodnotíme jejich vhodnost pro řešení hlavního cíle práce. Jako zkoumané algoritmy byly zvoleny ty nejrychlejší uvedené ve srovnání publikovaném na stránkách skupiny počítačového vidění Middlebury College.[6] Ty se jeví pro zpracování živého videa jako nejvhodnější. Jedná se zejména o lokální algoritmy, které jsou ze své povahy velmi vhodné pro masivně paralelní zpracování například pomocí GPU a mělo by tak být možné dosáhnout výkonu dostatečného pro zpracování živého video vstupu.

Kapitola 2

Algoritmy pro stereo matching

2.1 Základní rozdělení algoritmů

Samotné algoritmy pro stereo matching se stejně jako celý proces tvorby hloubkové mapy skládají z několika kroků. Každý z těchto kroků se pak vyznačuje způsobem jakým je prováděn, což nám umožňuje provést kategorizaci těchto algoritmů podle nich. Podle [7] jsou pak ony zmíněné kroky tyto:

1. Výpočet ceny párování
2. Agregace ceny v podpůrném regionu
3. Výpočet / optimalizace disparity
4. Zjemňování disparity

Každý algoritmus pak provádí podmnožinu těchto kroků, přičemž se může lišit i jejich pořadí. Některé algoritmy například vynechávají krok s agregací ceny, protože k problému přistupují jako k optimalizační úloze (tzv. *globální* metody). Jiné algoritmy pak mohou některé z kroků kombinovat do jednoho. Lokální algoritmy založené na porovnávání posuvného okénka pak mohou spojovat výpočet a agregaci ceny do jednoho kroku. My si nyní jednotlivé kroky a možnosti jejich provedení rozebereme.

2.1.1 Výpočet ceny párování

Zde nejprve vysvětlíme co rozumíme pojmem cena. K tomu ovšem potřebujeme definovat pojem *obraz prostoru disparit* (*DSI* – disparity space image). To je jakákoliv funkce nebo obraz definovaný nad prostorem disparit (x, y, d), který obsahuje všechny možné hodnoty disparity pro všechny pixely obrázku. Výsledek DSI pak představuje „vhodnost“, jinak řečeno cenu daného pixelu pro danou hodnotu disparity.

Mezi nejběžnější druhy výpočtu ceny, patří výpočty prováděné na základě intenzity pixelů. Může se tak jednat o výpočty pomocí čtverce rozdílu intenzit (SD – squared intensity differences) nebo absolutní hodnoty rozdílu intenzit (AD – absolute intensity differences). Mezi další způsoby pak patří použití normalizované vzájemné korelace, nebo binární reprezentace ceny.

Výsledkem výpočtu ceny přes všechny pixely a hodnoty disparity je pak iniciální DSI. To je náš základní objekt na kterém se pak provádí agregace či optimalizace.[7]

2.1.2 Agregace ceny

Agregace ceny se provádí zejména u lokálních algoritmů. Ty pak agregují cenu pomocí součtu nebo průměrování v rámci takzvaného podpůrného regionu DSI, ten pak může mít dva nebo tři rozměry. Záleží, zda budeme mít okénko s konstantní disparitou, či ne. Samotná agregace pak může být provedena například konvolucí nebo jiným efektivním způsobem.

2.1.3 Výpočet a optimalizace disparity

Lokální metody kladou důraz na agregaci cen a optimalizaci většinou vynechávají. Samotný výpočet disparity je pak poměrně jednoduchý. Pro každý pixel se vybere hodnota disparity, která byla oceněna nejnižší hodnotu. Takový přístup ovšem trpí nedostatkem v podobě rizika jevu, při kterém bude některý bod přiřazen k referenčnímu snímku vícekrát.

Globální metody naopak hlavní práci vykonávají při optimalizaci. U mnoha globálních metod jako problém minimalizace energie. Cílem je pak nalézt funkci disparity označovanou jako d, která minimalizuje globální energii. Optimalizace pak může být realizována například pomocí simulovaného ochlazování nebo markovských náhodných polí. Existují pak i metody využívající grafových řezů a toků v sítích. Jiné globální metody pak mohou využívat dynamického programování.

Posledním typem algoritmů, které využívají optimalizace, jsou pak takzvané *kooperativní* algoritmy. Ty mohou iterativně provádět lokální výpočty, ale tváří se spíše jako metody globální.

2.1.4 Zjemňování disparity

Vzhledem k tomu, že většina algoritmů vypočítá odhady disparit v diskretizované podobě (vetšinou obsahuje několik „hladin“ disparit), je vhodné na závěr provést nějakou formu vyhlazení. Toho může být docíleno jednak použitím většího množství stupňů disparit, nebo proložením těchto stupňů křivkou. Při použití tohoto postupu, tak může být s malou výpočetní náročností dosaženo vizuálně dobře působícího výsledku. Ovšem za předpokladu, že původní rozmístění a spojitá povaha ploch odpovídá skutečnosti.

Je rovněž možné, a mnoho algoritmů tento postup používá, provést segmentaci mapy disparit a zneplatnit malé oblasti, jejichž disparita se výrazněji odlišuje od okolí (takzvané *peaky*). Takové chybné oblasti v mapě disparit často vznikají jako následek šumu, který se objevil při snímání obrazu.

Dalším dílčím krokem v rámci této části algoritmu pak může případně být začištění oblastí, které nebyly úspěšně spárovány, což bývá velmi často způsobeno překryvy objektů ve scéně. Tento krok může být realizován například porovnáním obou map disparit vzhledem k levé a pravé kameře jak již bylo zmíněno, nebo jednoduše pomocí distribuce sousedních hodnot disparit do okolí.

2.2 SSD Block Matching

V této sekci si představíme základní algoritmus pro stereo matching. Tento by se dal označit jako „naivní“ nebo jako brute-force přístup a můžeme ho použít pro srovnání s dalšími algoritmy, které budou představeny dále. Podle rozdělení, které bylo představeno výše, můžeme jeho dílčí kroky popsat následovně.[7]

1. Výpočet ceny párování – čtverec rozdílů
2. Agregace ceny – součet ve čtvercovém okénku
3. Výpočet / optimalizace disparity – nejlepší bere vše

Způsob jeho fungování je jednoduchý, pro každý pixel p o souřadnicích x, y a hodnotu disparity d se vypočítá cena následujícím způsobem.

$$C(p_{x,y}, d) = \sum_{x,y \in W} (I^l_{x,y} - I^r_{x-d,y})^2 \tag{2.1}$$

Kde W je množina souřadnic pixelů v zadaném okénku a $I^l_{x,y}$ a $I^r_{x,y}$ jsou intenzity pixelů v levém a pravém snímku na daných souřadnicích.

Pro každý pixel tedy získáme několik hodnot výsledné ceny a pro určení hodnoty disparity vybereme tu, pro kterou byla vypočítána nejnižší cena. Tento postup opakujeme pro všechny pixely ve snímku. Místo SSD je možné použít i jiné způsoby výpočtu ceny, například normalizovanou korelaci nebo součet absolutních hodnot rozdílů (SAD). Jako výhody tohoto algoritmu pak můžeme uvést jednoduchou implementaci nebo snadnou možnost paralelizace. OpenCV pak obsahuje CUDA implementaci tohoto algoritmu s použitím SAD pro výpočet a agregaci ceny.

2.3 Semiglobal Matching

Dalším algoritmem, který je implementován v rámci knihovny OpenCV, je algoritmus *Semiglobal Block Matching* (SGBM). Ten je založen na algoritmu

Semiglobal Matching[8] (SGM) představeném Heiko Hirschmüllerem v roce 2008 a který bude v této části přiblížen. OpenCV implementace obsahuje oproti tomuto algoritmu jisté rozdíly, mezi něž patří například fakt, že SGBM páruje bloky pixelů zatímco SGM pracuje s jednotlivými pixely. Je však možné SGBM spustit i v režimu, který Hirschmüllerově algoritmu odpovídá, ovšem za cenu velké paměťové náročnosti. Algoritmus SGM je pak založen jednak na párování pixelů a za druhé na aproximaci globálního dvourozměrného požadavku na hladkost (2D smoothness constraint) pomocí kombinace většího množství požadavků jednorozměrných.

Vstupem algoritmu jsou dva snímky se známou epipolární geometrií. Rektifikace nutná není. Cena párování dvou pixelů p a q pro danou disparitu d, přičemž q se nachází na epipolární přímce odpovídající bodu p, může být vypočítána dvěma způsoby. Prvním způsobem je použití techniky nezávislé na vzorkování podle Birchfielda a Tomasiho, kde je cena určena jako nejmenší absolutní hodnota rozdílu intenzit v okolí půl pixelu oběma směry na epipolární přímce. Tento způsob je pak použit ve výchozí OpenCV implementaci SGBM s tím, že cena je rovněž agregována v rámci bloku. Druhou možností je výpočet takzvané *společné informace* (Mutual Information), jenž vychází z entropie intenzit pixelů ve snímcích a sdružené entropie obou snímků.

Při použití takového výpočtu cen ovšem získané informace nejsou dostatečně přesné. Špatná párování tedy mohou mít nižší cenu než párování správná, což může být zapříčiněno například šumem v obrazu. Proto je u globálních metod zavedeno další omezení, které vylepšuje hladkost výsledného obrazu penalizováním ceny větších změn disparity v sousedících pixelech. Celkovou energii obrazu pak udává suma cen všech pixelů, k nimž jsou přičteny jejich penalizace zvyšující jejich cenu. Tato celková energie je pak optimalizována na co nejnižší hodnotu, pomocí výběru vhodných hodnot disparit pro jednotlivé pixely.

Protože je ovšem taková optimalizace výpočetně náročný problém, používá se u semiglobální metody přístup odlišný. Pro každý pixel je vypočítána nová cena $S(p, d)$, která je výsledkem optimalizace jednorozměrných cest, které končí v zadaném pixelu p s hodnotou disparity d. Optimalizace jednorozměrných cest je totiž snadnější než optimalizace dvojrozměrného snímku. Ve výsledku pak může být použito až 16 cest z různých směrů. V OpenCV implementaci je použito směrů 5.

Jako výsledná disparita pro první snímek je pak pro každý pixel či blok určena ta s nejnižší cenou S. Získat můžeme i mapu disparit vzhledem k druhému (pravému) snímku. Stačí když pro každý pixel q pravého snímku nalezneme v levém snímku pixel s nejnižší cenou S, který se nachází na epipolární přímce odpovídající pixelu q. Jeho disparita pak odpovídá disparitě pixelu q. Tento způsob ovšem není tak přesný, jako kdybychom celý algoritmus provedli vzhledem k pravému snímku. V okamžiku kdy máme obě mapy, lze provést kontrolu konzistence a odstranění peaků tak, jak bylo popsáno v předchozí kapitole.

2.4 Dvouúrovňová korelační metoda

Další metodou, která zde bude představena, je Dvouúrovňová korelační metoda. Ta využívá výpočtu normalizované vzájemné korelace (summed normalized cross-corelation — SNCC) a byla představena roku 2010 Nilsem Eineckem a Julianem Eggertem z Honda Research Institute Europe v Německu[9].

Tento algoritmus se řadí mezi lokální algoritmy a jak je z názvu zřejmé, pro výpočet cen používá normalizovanou vzájemnou korelaci, která je dána následujícím vztahem.

$$\rho_x = \frac{\frac{1}{|p(x)|}\sum_{x' \in p(x)}(I^L_{x'} - \mu^L_x)(I^R_{x'+d} - \mu^R_{x+d})}{\sigma^L_x \sigma^R_{x+d}} \tag{2.2}$$

Kde μ a σ jsou střední hodnota respektive směrodatná odchylka, hodnota d pak udává aktuálně vypočítávanou hodnotu disparity. Tento vztah bychom použili při provádění block-matchingu pomocí NCC. V dvouúrovňové korelační metodě je ovšem přístup jiný.

Jeho dvouúrovňovost spočívá v následujícím principu. Nejprve jsou pro všechny pixely z jejich malého okolí (3×3 nebo 5×5) předpočítány hodnoty μ a σ. Poté již můžeme začít testovat jednotlivé hodnoty disparity. Pro danou hodnotu disparity vypočítáme korelaci všech pixelů snímku s pixely druhého snímku posunutého právě o onu hodnotu disparity. K tomu použijeme stejně velké okénko jako v předchozím kroku, a tedy i předpočítané hodnoty μ a σ. To je první úroveň výpočtu korelace. Druhá úroveň pak spočívá v další aplikaci box-filteru, který počítá průměrnou hodnotu v rámci okénka. Získaná průměrná hodnota je pak úroveň korelace pixelu pro danou disparitu. Jako nejlepší je tedy vybrána hodnota disparity s nejvyšší korelací.

Podle autorů tento přístup pomáhá se zpřesněním odhadu disparity v oblastech s velkým kontrastem, tedy v oblastech, kde dochází k ostrým změnám hloubky. Celá metoda je pak efektivně implementovatelná například za použití integrálního obrázku pro implementaci box-filtrů, kterými jsou počítány μ a σ v rámci jednotlivých okének. V článku je ovšem pro efektivní výpočet box-filtru využito jeho separability.

Po výpočtu cen je jako druhý krok provedeno zpřesnění hodnoty disparity pomocí proložení kvadratické křivky body v okolí nalezené maximální hodnoty korelace. Třetím krokem je kontrola konzistence levého a pravého snímku. Čtvrtý a pátý krok pak představuje zneplatnění malých regionů jako v předchozím případě, respektive interpolace neplatných pixelů pomocí sousedních platných hodnot. Jako maximální velikost oblastí pro zneplatnění autoři uvádí hodnotu 200 pixelů.

2.5 Iterativní metoda zjemňování pro korespondence s adaptivní podpůrnou vahou

Poslední v této práci představenou metodou, je metoda založená na použití *adaptivních podpůrných vah* (dále ASW – Adaptive support weights), jejichž přesnost je dále iterativním způsobem zpřesňována[2]. Tato metoda byla publikována v článku z roku 2013, jehož autory jsou Jędrzej Kowalczuk, Eric Psota a Lance Pérez z University of Nebraska-Lincoln. Ve srovnávacích testech University of Middlebury[6] je pak paralelní implementace této metody za použití technologie CUDA vyhodnocena jako jedna z nejrychlejších. Pro hlavní cíl práce se tedy jeví jako velmi vhodná.

První součástí celé metody je metoda ASW, která byla poprvé uvedena v roce 2005. I když se jedná o lokální metodu, projevuje se velmi dobrou přesností, která se blíží i metodám globálním. Metoda ASW se pokouší napodobovat principy, na jakých pracuje i lidské vidění. Tedy například princip blízkosti (proximity principle) a princip podobnosti (similarity principle). Zjednodušeně řečeno, princip blízkosti tvrdí, že pravděpodobnost zda pixel p patří ke stejné ploše jako pixel q, klesá s rostoucí vzdáleností těchto dvou pixelů. Princip podobnosti pak říká, že pokud pixely p a q mají podobnou barvu, je pravděpodobné, že patří do stejné plochy ve scéně.

Samotné ASW je pak definováno následujícím způsobem. Každý pixel p ve snímku má své podpůrné okolí Ω_p, což je čtvercové okénko se středem v p. Pro každý pixel $q \in \Omega_p$ je pak následujícím vztahem definována váha $w(p, q)$.

$$w(p,q) = \exp\left(-\frac{\Delta_c(p,q)}{\gamma_c} - \frac{\Delta_g(p,q)}{\gamma_g}\right) \tag{2.3}$$

Kde Δ_c a Δ_g je rozdíl intenzit respektive euklidovská vzdálenost pixelů p a q. Konstanty γ_c a γ_g jsou pak konstanty, jež je nutné empiricky určit.

Cenu párování dvou pixelů p a p', kde p je pixel v levém snímku a p' ve snímku pravém, pak určuje vztah následující.

$$C(p,p') = \frac{\sum\limits_{q\in\Omega_p, q'\in\Omega_{p'}} w(p,q)w(p',q')\delta(q,q')}{\sum\limits_{q\in\Omega_p, q'\in\Omega_{p'}} w(p,q)w(p',q')} \tag{2.4}$$

Kde $\delta(q, q')$ je suma absolutních hodnost rozdílů intenzit.

Pro urychlení zmíněné agregace pak autoři metody uvádějí ještě zjednodušenou formu agregace, která použití čtvercového okénka aproximuje. Místo celého okénka se váhy agregují pouze v jednorozměrném sloupci, respektive řádku, které obsahují pixel pro nějž hodnotu w počítáme. Za cenu snížení přesnosti se totiž složitost agregace řádově sníží z $O(\omega^2)$ na $O(\omega)$, kde ω je velikost strany okénka. Existují pak i další metody jak agregaci zrychlit ovšem pro v článku popsanou implementaci její autoři zvolili tuto.

Jako u ostatních lokálních algoritmů by pak jako finální párování pixelů bylo určeno to s nejnižší cenou. V tomto případě ovšem nastupuje iterativní část algoritmu, která získané disparity dále zpřesňuje a nalezené hodnoty jí slouží jako výchozí. Její princip spočívá v dodatečném penalizování těch párování, která se jeví jako nevhodná, a zvyšování jejich ceny. Iterativní postup pak lze popsat následujícím vztahem.

$$C^i(p, p') = C^0(p, p') + \Lambda^i(p, p') \tag{2.5}$$

Kde právě funkce $\Lambda^i(p, p')$ je ona zmiňovaná penalizace a definována je takto.

$$\Lambda^i(p, p') = \alpha \sum_{q \in \Omega_p} w(p, q) F_q^{i-1} |D_q^{i-1} - d(p, p')| \tag{2.6}$$

Hodnota D_q^{i-1} je vypočítaná hodnota disparity pro pixel q z předchozí iterace. F_q^{i-1} je pak procentuální hodnota udávající rozdíl mezi první a druhou nejlepší nalezenou cenou, dá se tak o ní hovořit i jako o míře důvěryhodnosti daného párování. Pokud je rozdíl mezi první a druhou hodnotou velký, je pravděpodobné, že nalezené párování je vhodné. V tomto případě se rovněž jedná o hodnotu vypočítanou v předchozí iteraci. Funkce $d(p, p')$ je horizontální vzdálenost pixelů pro něž je penalizace vypočítávána a poslední hodnota α je empiricky nalezená konstanta.

Pokud tedy na algoritmus nahlédneme jako na celek, je možné ho popsat následujícím způsobem.

1. r iterací pro požadované hodnoty disparity
   a) Vertikální agregace cen
   b) Horizontální agregace cen
2. Výběr disparity s nejlepší cenou vzhledem k levému i pravému snímku
3. Kontrola konzistence
4. i iterací zjemňování
   a) Vertikální zjemnění disparity vzhledem k levému i pravému snímku
   b) Horizontální zjemnění disparity vzhledem k levému i pravému snímku
   c) Výběr disparity s nejlepší cenou
   d) Kontrola konzistence
5. Dodatečné zpracování

Kontrolou konzistence zde opět rozumíme porovnání disparit spárovaných pixelů vzhledem k levému a pravému snímku. Pokud platí, že $|D_p^R - D_{p'}^L| > 1$, je pixel označen jako neplatný a jeho míra důvěryhodnosti F_p^i je okamžitě nastavena na nulu. Dodatečné zpracování pak představuje aplikace mediánového filtru o velikosti 3×3 a zaplnění neplatných pixelů hodnotami aproximovanými z okolí.

Kapitola 3

Implementace

3.1 Použité technologie

V první části kapitoly věnující se samotné implementaci budou popsány technologie, jež jsou pro implementaci použity. Jedná se konkrétně o multimediální knihovnu libyuri, CUDA, což je platforma pro paralelní výpočty a programování za pomoci grafických procesorů a knihovnu OpenCV. Tyto knihovny budou postupně blíže představeny a budou uvedeny důležité funkce, které jsou pro implementaci řešení našeho problému použity.

3.1.1 libyuri

Libyuri[10] je multimediální knihovna vyvíjená Institutem Intermédií, který je společným pracovištěm Českého vysokého učení technického v Praze a Akademie múzických umění. Knihovna libyuri je pak vlastně platformou pro zpracování multimediálního obsahu, zejména pak videa. Tato technologie pak byla vybrána zejména díky velkému množství dostupných modulů. Je tak velmi snadno možné do v této práci implementovaných modulů předávat obrazová data z různých zdrojů. To mohou nýt například různé kamery (V4L zařízení, Decklink zařízení, ...) nebo i síťové streamy (např. technologie Ultragrid). Bude pak velmi snadné vytvořit systém pro vzdálené zpracování a zobrazení snímků získaných z kamer.

Hlavní součástí knihovny libyuri je jádro, které poskytuje základní funkcionalitu a hlavně definuje rozhraní pro vytváření dalších modulů. Každý modul je pak představován třídou, která implementuje nějakou operaci zpracovávající multimediální obsah. Existují tedy moduly, které fungují například jako zdroje videa a které jsou producenty jednotlivých snímků videa. Tento zdroj může představovat například síťový stream, zařízení připojené pomocí USB nebo soubor na disku počítače. Takový zdroj pak vstupní data dekóduje a jednotlivé snímky odesílá dalším modulům ke zpracování.

Těmito moduly pak mohou být moduly realizující například zpracování

obrazu. Z těch jednodušších můžeme zmínit například modul *flip*, který obraz otáčí kolem svislé nebo vodorové osy, nebo modul *contrast* upravující kontrast obrazu. K dispozici pak jsou i moduly umožňující výstup zpracovávaných snímků, například do okna na obrazovce, nebo enkódování do výstupního souboru.

Požadované zapojení modulů je definováno pomocí XML konfiguračního souboru, který je předložen hlavnímu spustitelnému souboru platformy libyuri. Libyuri s jednotlivými moduly pracuje jako s uzly orientovaného grafu a konfigurační XML soubor je jeden ze způsobů jak takový graf definovat. Jednoduchý konfigurační soubor si můžeme prohlédnout na následujícím příkladu.

```
1  <?xml version="1.0" ?>
2  <app name="webcam" xmlns="urn:library:yuri:xmlschema:2001"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4       <variable name="device">/dev/video1</variable>
5       <node class="v4l2source" name="webcam">
6            <parameter name="path">@device</parameter>
7            <parameter name="fps_stats">1</parameter>
8       </node>
9       <node class="sdl2_window" name="sdl"/>
10      <link name="yuyv_image" class="single"
11           source="webcam:0" target="sdl:0"/>
12 </app>
```

Jak vidíme na ukázce, ve schématu se nachází 5 různých druhů elementů, které budou nyní blíže popsány.

app Kořenový element schématu, který uzavírá všechny ostatní elementy.

variable Představuje proměnnou, na kterou se lze dále v konfiguraci odkazovat. Obsah elementu představuje výchozí hodnotu proměnné, kterou lze nastavit při spuštění z příkazové řádky.

node Představuje instanci libyuri modulu. Atribut *class* udává třídu modulu a atribut *name* název uzlu v rámci konfigurace. Element *node* pak může obsahovat elementy *parameter*.

parameter Nastavuje hodnoty parametrů jednotlivých instancí modulů.

link Definuje spoje mezi jednotlivými uzly, kterými procházejí zpracovávaná data. Jeho atributy *source* a *target* obsahují názvy příslušných uzlů a index jejich vstupu či výstupu.

V ukázce tedy máme dva uzly, představované instancemi modulů *v4l2source* a *sdl2window*. Tyto uzly jsou nazvané jmény *webcam* respektive *sdl*. Uzel *webcam* má v sobě dva elementy *parameter*, které této instanci modulu nastavují

zadané hodnoty. U parametru *path* je jako hodnota použit odkaz na proměnnou *path*. U uzlu *sdl* pak žádné parametry nastavovány nejsou. Element *link* pak realizuje spoj mezi uzly *webcam* a *sdl*, kterým po spuštění budou proudit obrazová data.

3.1.2 CUDA

CUDA je platforma pro paralelní programování a výpočty vyvíjená společností NVIDIA Corporation a je určena pro použití na grafických procesorech tohoto výrobce. Grafické procesory jsou speciálně konstruovány pro masivně paralelní zpracování obrazových, ale i jiných dat. Jejich síla pak spočívá ve schopnosti paralelně provádět velké množství identických, na sobě nezávisejících výpočtů, což je při zpracování obrazu běžné. Zjednodušeně je tedy možné říci, že na rozdíl od CPU, grafické procesory věnují více tranzistorů samotným výpočtům, než řídícím obvodům, práci s pamětí a cachování dat.[11]

Platforma CUDA je navržena tak, aby byla schopna jednoduché spolupráce s programovacími jazyky C, C++ a Fortran. Pro tyto jazyky poskytuje jednoduchá rozšíření pomocí nichž je možné technologii CUDA používat. CUDA pak svou efektivitu staví na třech základních principech, kterými jsou: hierarchie skupin vláken, sdílená paměť a bariérová synchronizace. Tyto principy pak programátora navádějí k dekompozici řešeného problému podle následující filozofie:

1. Rozdělit problém na dílčí nezávislé podproblémy
2. Podproblémy rozdělit na jemnější části kooperativně řešené jednotlivými vlákny

První bod dekompozice v CUDA představují takzvané bloky, což jsou samostatné skupiny až 1024 vláken. Bloky jsou na sobě navzájem nezávislé a umožňují jednoduchou škálovatelnost. Čím více bloků nám hardware umožňuje spustit, tím rychleji celý výpočet proběhne. Vlákna, ze kterých jsou bloky složené, je pak možné bariérově synchronizovat a mají k dispozici 48 kilobajtů rychlé sdílené paměti, k níž mohou všechna vlákna v rámci bloku přistupovat. Tato paměť se dá v zásadě označit jako explicitně spravovaná datová cache.

Samotné programování je pak realizováno pomocí takzvaných *kernelů*. *Kernel* je procedura, která je samostatně a paralelně provedena každým ze spuštěných vláken. Definici kernelu si ukážeme na následujícím příkladu.

```
1 __global__ void myKernel(int* a, int* b){
2         int idx=threadIdx.x;
3         a[idx] = a[idx] + b[idx];
4 }
```

Jak vidíme, kód kernelu je uvozen klíčovým slovem *__global__*, což je jedno z důležitých rozšíření, které CUDA do jazyka C respektive C++ přináší. Další nutnou podmínkou pak je, aby funkce kernelu měla návratovou

hodnotu typu *void*. Proměnná *threadIdx.x* pak označuje souřadnice vlákna v rámci bloku. Takto adresovat je možné jak vlákna, tak i samotné bloky, a to až ve třech rozměrech. V rámci kernelu je pak možné k těmto souřadnicím přistoupit pomocí proměnných *threadIdx.x*, threadIdx.y a *threadIdx.z*, respektive proměnné *blockIdx*. Rozměry bloku v patřičných rozměrech jsou pak uloženy ve struktuře *blockDim*.

Nyní si předvedeme, jak se takový kernel volá v kódu aplikace. V příkladu je naším cílem po složkách sečíst dvě pole *a* a *b* a výsledek uložit do pole *a*.

```
1  int main(){
2          int a[5] = { 1, 2, 3, 4 , 5 };
3          int b[5] = { 6, 7, 8, 9 , 10 };
4          size_t size = sizeof(int);
5          int *a_device,*b_device;
6
7          cudaMalloc(&a_device, size);
8          cudaMalloc(&b_device, size);
9
10         cudaMemcpy(a_device, a, size, cudaMemcpyHostToDevice);
11         cudaMemcpy(b_device, b, size, cudaMemcpyHostToDevice);
12         myKernel<<<1,5>>>(a_device, b_device);
13         cudaMemcpy(a, a_device, size, cudaMemcpyDeviceToHost);
14         return 0;
15 }
```

Jak je patrné, před samotným voláním kernelu na řádku 12, je potřeba provést ještě další důležité úkony. Jako první je třeba alokovat paměť v *globální* paměti grafické karty. To se děje za pomoci funkce *cudaMalloc*. V příkladu jsme tedy alokovali prostor pro pole *a* a *b* a funkce *cudaMalloc* nám na ně vrátila ukazatele. Tyto ukazatele ovšem představují adresy v paměti na grafické kartě a mimo kernely k nim nelze přistoupit. Na řádcích 10 a 11 pak prostřednictvím funkce *cudaMemcpy* kopírujeme obsah obou polí do globální paměti GPU.

V tuto chvíli již má GPU potřebná data, a my můžeme spustit náš kernel. Ve špičatých závorkách udáváme v kolika blocích a vláknech se má kernel spustit. V tomto případě se spouští jeden blok o pěti vláknech. Pokud bychom například chtěli, aby kernel pomocí jednoho vlákna zpracovával jeden prvek čtvercové matice velikosti 160×160 prvků, a my jsme si chtěli usnadnit indexaci vláken a bloků, můžeme použít speciální strukturu *dim3* tak, jako v následujícím příkladu.

```
...
dim3 blocks(10,10);
dim3 threads(16,16);
...
kernel<<<blocks, threads>>>(...);
```

```
...
```

V samotném kernelu by pak výpočet indexů vypadal následujícím způsobem.

```
...
int x = blockIdx.x * blockDim.x + threadIdx.x;
int y = blockIdx.y * blockDim.y + threadIdx.y;
...
```

Po takovém výpočtu by pak každé vlákno vědělo, ke kterému prvku matice přistoupit.

Další užitečnou možností, je pak možnost definovat funkce, které lze volat v rámci kernelů a které mají běžnou návratovou hodnotu. V takovém případě je nutné funkci uvést klíčovým slovem *__device__* stejným způsobem jako je použito *__global__* u definice kernelu. Funkce je pak přeložena spolu s kernely do strojového kódu GPU a je možné ji volat pouze v rámci kernelů.

Ve chvíli kdy máme připraven program obsahující kernely, zbývá než ho zkompilovat a vytvořit spustitelný binární soubor. Kompilace zdrojových kódů využívajících technologii CUDA probíhá prostřednictvím kompilátoru *nvcc*, který je součástí instalace vývojového kitu CUDA.

3.1.3 OpenCV

OpenCV, celým názvem *Open Source Computer Vision Library*, je knihovna obsahující funkce z oblasti počítačového vidění a strojového učení. V současnosti do ní patří více než 2500 algoritmů z těchto dvou oblastí, které mohou být použity pro rozpoznávání různých objektů nebo lidských tváří, sledování objektů ve scéně nebo naopak sledování pohybu kamery a pro nás důležité algoritmy související s prostorovým viděním.

Knihovna samotná je implementována v jazyce C++, poskytuje však rozhraní pro jazyky C, Python, Java a MATLAB a je možné ji používat na všech běžných operačních systémech, a to včetně těch mobilních (např. Android). Některé algoritmy pak mohou využívat akcelerace poskytnuté technologií CUDA nebo OpenCL.

Vzhledem k širokému spektru algoritmů, které knihovna obsahuje, je knihovna rozdělena do modulů podle oblasti jejich určení. Kromě modulu *core*, který obsahuje zakladní funkcionalitu (např. reprezentace obrázků v podobě matic apod.) nás budou zajímat hlavně moduly *calib3d* a *cudastereo*.

Modul *calib3d* nám, jak jeho název napovídá, obsahuje funkce umožňující kalibraci naší soustavy kamer. Pro kalibraci dvou kamer je pak speciálně určena funkce *cv::stereoCalibrate*, která nám poskytuje jednak matice s parametry obou kamer a jejich koeficienty zkreslení, ale rovněž přímo esenciální a fundamentální matice nutné pro následnou rektifikaci. Vstupem této funkce jsou pak souřadnice kalibračních bodů v rámci snímků získaných z obou kamer.

Tyto souřadnice je možné ve snímcích nalézt použitím funkce *cv::findChessboardCorners*.

Pro přípravu rektifikační transformace jsou pak určeny funkce *cv::stereoRectify* a *cv::initUndistortRectifyMap*. Jejich konečným výstupem je pak transformační matice, která po aplikaci na snímek z kalibrované kamery za pomoci funkce *cv::remap* zajistí deformaci snímku do požadovaného rektifikovaného stavu. Použití těchto dvou funkcí i použití funkce *cv::stereoCalibrate* bude ještě podrobněji vysvětleno při popisu implementace samotného kalibračního modulu.

Další součástí modulu *calib3d* jsou i implementace několika algoritmů pro Stereo Matching. Konkrétně se jedná o klasický block-matching algoritmus a algoritmus SGBM, které byly popsány v předchozí kapitole. Použití těchto algoritmů bude rovněž dále rozvedeno u popisu implementace příslušného libyuri modulu.

3.2 Implementované moduly

V rámci práce bylo implementováno celkem 7 libyuri modulů s následujícími názvy a účely.

opencv_stereo_calib Provádí kalibraci jejímž výsledkem je transformační matice pro rektifikaci snímků

opencv_rectify Za pomoci transformační matice rektifikuje snímky získané ze zdroje videa

opencv_sgbm Za pomoci SGBM algoritmu popsaného v části 2.3 vytváří hloubkovou mapu

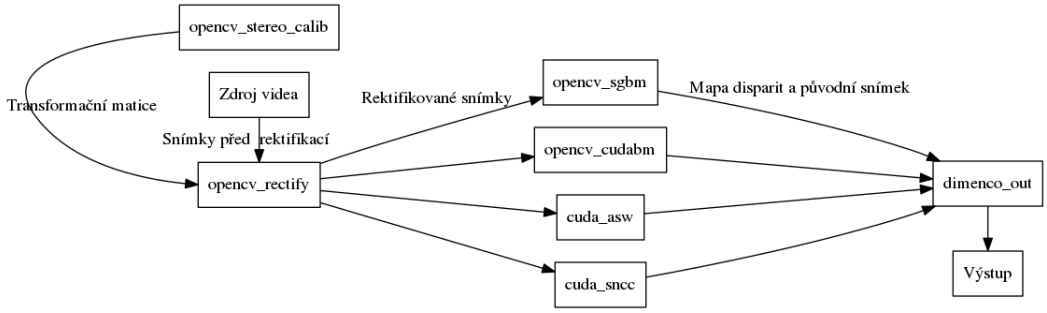
opencv_cudabm Za pomoci CUDA block-matching algoritmu popsaného v části 2.2 vytváří hloubkovou mapu

cuda_asw Za pomoci iterativního ASW algoritmu popsaného v části 2.5 vytváří hloubkovou mapu

cuda_sncc Za pomoci SNCC algoritmu popsaného v části 2.4 vytváří hloubkovou mapu

dimenco_out Připravuje side-by-side obraz pro zobrazení na autostereo monitoru Dimenco

Vazby mezi jednotlivými moduly můžeme rovněž ilustrovat na následujícím diagramu. V následujících sekcích pak bude implementace všech 7 modulů detailněji popsána.



Obrázek 3.1: Vazby mezi implementovanými moduly

Použití modulu *dimenco_out* není povinné. Snímky vycházející z modulů pro Stereo Matching je možné odeslat na výstup, i když tímto modulem neprošly. Jako výstup může být použito široké spektrum výstupních modulů, jež jsou již v libyuri implementované. Může se jednat například o SDL okno (modul *sdl_window*) pro přímé zobrazení na obrazovce, nebo modul *filedump* pro uložení snímků ve formě jednotlivých souborů. Stejně tak je možné použít i modul vytvářející RTP stream pro odeslání snímků na jiný počítač, kde je možné je dále zpracovávat nebo zobrazit. Možnost, kdy jsou síťovým streamem odeslány pouze rektifikované snímky a hloubková mapa je vytvořena až na jiném, výkonnějším stroji, je pak stejně dobře proveditelná.

3.2.1 Modul *opencv_stereo_calib*

Prvním modulem jehož implementace bude podrobněji popsána je modul *opencv_stereo_calib*. Tento modul poskytuje informace důležité pro následnou rektifikaci obrazu a využívá funkcí poskytnutých knihovnou OpenCV. V rámci implementace s pomocí knihovny libyuri je pak tento modul představován třídou *OpenCVStereoCalib*. Vzhledem k tomu, že se jedná o kalibraci pomocí stereoskopického obrazu potřebuje tento modul dva vstupy pro snímky videa. Toho je dosaženo díky třídě *SpecializedMultiIOFilter*, od které třída *OpenCVStereoCalib* dědí. Třída *SpecializedMultiIOFilter* je součástí jádra libyuri a umožňuje u modulu použití více vstupů či výstupů. V tomto případě máme dva vstupy a jeden výstup.

Modul dále disponuje následujícími parametry, které jsou použity pro jeho nastavení.

calibration_frames Počet párů snímků, které budou pro kalibraci použity

frame_delay Rozestup mezi snímky uloženými pro kalibraci

chessboard_x Počet průsečíků na kalibrační šachovnici ve vodorovné ose

chessboard_y Počet průsečíků ve svislé ose

path Cesta pro uložení výstupních souborů

Hlavní funkcionalita celého modulu je implementována v metodě *do_special_single_step*, která je volána pokaždé, když se na vstupních portech modulu objeví dvojice snímků. Pomocí parametrů této metody pak na tyto snímky obdržíme reference (typ *pRawVideoFrame*). Dále budeme očekávat, že snímek na vstupu 0 je snímek z levé kamery a snímek na vstupu 1 je snímek z kamery pravé. Postup při volání této metody je následující:

1. Převod snímků do podoby podporované funkcemi OpenCV, tedy na instance třídy *cv::Mat*
2. Konverze matic z barevných na černobílé
3. Vyhledání pozic průsečíků šachovnice v obou snímcích pomocí funkce *cv::findChessboardCorners*
4. Uložení nalezených souřadnic v případě, že byla šachovnice nalezena v obou snímcích
5. V případě, že je nasbíráno dostatečné množství souřadnic a kalibrace nebyla provedena, je volána metoda *calibrate*
6. Vykreslení pozic průsečíků do snímků prostřednictvím funkce *cv::drawChessboardCorners*
7. Zpětný převod matic *cv::Mat* na instance třídy *RawVideoFrame*
8. Odeslání nových snímků na výstup

Obrázek 3.2: Body šachovnice vykreslené ve snímcích

Hlavní část kalibrace se pak odehrává v metodě *calibrate*, což je privátní metoda třídy *OpenCVStereoCalib*. Ta je volána v případě, že již bylo nasbíráno množství souřadnic požadované parametrem *calibration_frames*. Prvním krokem v této metodě je příprava souřadnic průsečíků v rámci šachovnice. Souřadnice tedy mají v ose z vždy nulovou hodnotu. Těchto souřadnic musíme připravit stejné množství jako je množství souřadnic nasbíraných ze snímku. Pro numerický výpočet v následné kalibraci totiž potřebujeme následující trojice množin souřadnic.

- Množina souřadnic průsečíků z levého snímku
- Množina souřadnic průsečíků z pravého snímku
- Množina souřadnic průsečíků v rámci plochy šachovnice (pro všechny trojice vždy stejné hodnoty)

Ve chvíli, kdy jsou potřebné souřadnice připraveny, přichází na řadu funkce *cv::initCameraMatrix2D*. Ta nám za pomoci výše zmíněných souřadnic poskytne prvotní odhad hodnot obsažených v matici kamery. Pomocí této funkce pak opatříme matice pro levou i pravou kameru.

Nyní již může přijít na řadu funkce *cv::stereoCalibrate*. Za použití všech získaných souřadnic a matic obou kamer jejím prostřednictvím získáme následující informace.

- Zpřesněné matice obou kamer
- Koeficienty zkreslení obou kamer
- Matici rotace představující vzájemné pootočení obou kamer
- Matici translace představující posunutí mezi kamerami
- Esenciální matici
- Fundamentální matici

S těmito důležitými informacemi je nyní možné přistoupit k dalšímu kroku, jímž je získání parametrů pro rektifikaci. K tomuto účelu slouží funkce *cv::stereoRectify*. Díky maticím kamer, koeficientům zkreslení a maticím translace a otočení z ní získáme další matice zobrazení *R1*, *R2*, *P1*, *P2* a Q. R a P představují transformace uskutečňující samotnou rektifikaci a matice Q by byla užitečná, pokud bychom chtěli z hodnot disparit získat informaci o délce v délkových jednotkách. Jednou délkovou jednotkou by v tomto případě byl rozměr políčka na kalibrační šachovnici. Stejně tak tomu je i v případě rotačních a translačních matic.

Pro usnadnění operace rektifikace, která bude prováděna v samostatném modulu, nám ještě zbývá připravit transformační mapy. Princip funkce těchto

map je jednoduchý, pro každý pixel ve snímku určuje jeho souřadnice ve snímku rektifikovaném. Pro každý snímek jsou k tomuto účelu potřeba mapy dvě a obě je nám poskytne funkce *cv::initUndistortRectifyMap*. K tomuto úkonu je potřeba jí jako parametry předat matice kamer, koeficienty zkreslení a matice zobrazení R a P získané v předchozím kroku. Jak pro levý, tak pravý snímek tedy získáme dvě mapy, které spolu s dalšími nalezenými maticemi a hodnotami uložíme do YML souborů pro pozdější použití. OpenCV obsahuje třídy a metody, které umožňují jejich jednoduché ukládání i načítání, a ty jsou k tomuto účelu použity i v tomto případě.

Tímto je kalibrace naší soustavy kamer hotová a za předpokladu, že soustava nebude nijak změněna, je možné transformační mapy pro rektifikaci dále používat i v budoucnu. Při jakémkoliv posunutí nebo pootočení kamer nebo změně rozlišení snímků však bude nutné provést kalibraci znovu.

3.2.2 Modul *opencv_rectify*

Modulem, který využívá transformační matici poskytnutou předchozím modulem, je modul implementovaný třídou *OpenCVRectify*. Tento modul je oproti předchozímu poměrně jednoduchý, a dědí od třídy *SpecializedIOFilter*. Má tedy pouze jeden vstup a výstup. Pro rektifikaci stereo obrazu jsou tedy nutné dvě instance tohoto modulu. Nastavení modulu je pak provedeno pomocí dvou následujících parametrů.

map_file Cesta k YML souboru s transformační mapou

left Logický příznak zda se jedná o levý snímek, v případě pravého snímku musí být hodnota nastavena na *false*

Obrázek 3.3: Levý a pravý snímek před rektifikací

V konstruktoru této třídy jsou za pomoci uvedených parametrů načteny mapy pro levý či pravý snímek. Transformace jednotlivých snímků pak pro-

bíhá v metodě *do_special_single_step*. Nejprve je, podobně jako v předchozím modulu, vytvořena instance třídy *cv::Mat*. Vzhledem k tomu, že třída je potomkem třídy *SpecializedIOFilter*, je možné u ní nastavit podporovaný formát jednotlivých pixelů obrazu. Při vstupu do modulu jsou pak jádrem libyuri snímky do příslušného formátu automaticky konvertovány a je možné bezpečně určit formát pixelů pro matici *cv::Mat*. V tomto případě jsou snímky převáděny do formátu *BGR24*, což je formát, který OpenCV zpravidla používá. Formát snímků matice je tedy určen konstantou *cv::CV_8UC3* (tři kanály po osmi bitech).

Samotná transformace pak probíhá prostřednictvím funkce *cv::remap*. Této funkci jsou jako parametry předány následující položky.

- Matice se zdrojovým snímkem
- Matice pro výstupní snímek
- První transformační mapa
- Druhá transformační mapa
- Konstanta určující způsob interpolace pixelů, v tomto případě je s ohledem na rychlost použita bilineární metoda

Po provedení transformace jsou obrazová data z výstupní matice opět použita k vytvoření nového snímku *RawVideoFrame* a ukazatel na nějž je následně použit jako návratová hodnota metody a tímto předán ke zpracování dalšímu modulu.

Obrázek 3.4: Levý a pravý snímek po provedení rektifikace

3.2.3 Modul *opencv_sgbm*

Tento modul je prvním modulem, který poskytuje jako výsledek mapu disparit, a to za použití semi-globální metody popsané v části 2.3. Vzhledem k tomu,

že opět používá třídy a metody poskytnuté knihovnou OpenCV, je i samotná třída *OpenCVSGBM* relativně jednoduchá a podobá se předchozím modulům, které OpenCV také využívají. Pro nastavení modulu jsou v tomto případě použity parametry následující.

min_disparity Minimální hodnota disparity (posun obrazů), který bude testován

num_disparities Počet hodnot disparit, které budou testovány

window_size Velikost okénka pro agregaci

hh_mode Zapíná přesnější *HH* režim

Za pomoci uvedených parametrů je v konstruktoru třídy vytvořena instance třídy *cv::StereoSGBM*, která implementuje metodu SGBM popsanou v části 2.3. Další parametry jsou určeny podle doporučení v dokumentaci OpenCV. Parametr *hh_mode* určuje zda bude algoritmus prováděn v úplném módu tak jak ho popsal Heiko Hirschmüller v [8]. Pokud je parametr nastaven na zápornou hodnotu, je prováděna zjednodušená verze algoritmu, která je ovšem mnohem méně paměťově náročná.[12]

Při zpracování snímků v metodě *do_special_single_step* je pak postup přímočarý. Obrazová data ze snímků jsou opět převedena na matice *cv::Mat*. Díky předchozímu použití modulu *opencv_rectify*, bez něhož by další zpracování nemělo valný smysl, rovněž můžeme očekávat že data již přijdou ve formátu *BGR24* a můžeme převod bez problémů provést.

Po převodu snímků na matice je již možné vypočítat samotnou mapu disparit. K tomu slouží metoda *compute* instance třídy *cv::StereoSGBM*. Ta potřebuje tři parametry, kterými jsou:

- Matice s levým snímkem
- Matice s pravým snímkem
- Matice pro výslednou mapu disparit

Výsledná matice s disparitou je po provedení výpočtu převedena na matici s jedním barevným kanálem o hloubce 8 bitů. Z této matice je následně stejně jako v předchozím modulu vytvořen *RawVideoFrame* a tento nový snímek je dále předán dalšímu modulu ke zpracování či zobrazení.

3.2.4 Modul *opencv_cudabm*

Modul *opencv_cudabm* je velmi podobný modulu předchozímu. Princip jeho funkce je naprosto totožný, jisté rozdíly však existují a budou v této části uvedeny. Modul je implementován třídou *OpenCVCudaBM* a pro výpočet mapy disparit používá block-matchingový algoritmus (2.2) implementovaný

v CUDA. Parametry pro jeho nastavení jsou stejné, chybí zde ovšem parametr pro minimální hodnotu disparity.

Pro výpočet disparity je v tomto modulu použita instance třídy *cv::cuda::StereoBM* a její metoda *compute*. Vzhledem k tomu, že výpočet probíhá na GPU, je nutné kromě matic použitých v předchozích modulech použít i matice třídy *cv::cuda::GpuMat*. Ty představují matice uložené v globální paměti grafické karty a ke kopírovaní mezi ní a hlavní pamětí počítače slouží její třídní metody *upload* a *download*.

Poté co jsou oba snímky uloženy v globální paměti a matice pro mapu disparit je tamtéž alokována, je možné zavolat metodu *compute* a mapu disparit vypočítat. Po výpočtu je mapa pomocí metody *download* z globální paměti zkopírována do hlavní paměti počítače a stejným způsobem jako u předchozího modulu je předána k dalšímu zpracování.

3.2.5 Modul *cuda_asw*

Dalším implementovaným libyuri modulem je modul nazvaný *asw_cuda*, který implementuje třída *CudaASW*. Jak název napovídá v tomto modulu je implementován algoritmus popsaný v části 2.5. Vzhledem k tomu, že je algoritmus v modulu implementován od základu celý, budou podrobněji popsány jeho jednotlivé části, zejména GPU kernely. Každému kernelu pak bude pro přehlednost věnována samostatná podsekce.

Nastavení modulu je jednoduché a probíhá pomocí následujících dvou parametrů:

num_disparities Počet hodnot disparit, které budou testovány

iterations Počet iterací zjemňovací části algoritmu

fill_iterations Počet aplikací filtru pro vyplnění na výstupní mapu

left Logický příznak zda bude na výstup odeslána levá či pravá mapa disparit

Jako u předchozích modulů jsou jednotlivé snímky zpracovávány v metodě *do_special_single_step*. Algoritmus samotný je implementován, zejména kvůli rychlosti zpracování, pouze pro černobílé snímky. Proto je třeba před zpracováním snímky do černobílého formátu konvertovat. K tomu slouží třída *yuri::core::Convert*, což je součást jádra libyuri.

Po konverzi levého i pravého snímku jsou ještě oba snímky doplněny na všech jejich okrajích o padding. Padding má pevnou šířku 16 pixelů a pomáhá s jednoduchým načítáním podpůrných regionů při agregaci cen. Je totiž důležité aby se kód kernelů spouštěných na GPU co nejméně větvil a došlo tak k co největšímu využití grafického procesoru. Díky tomu, že obrazová data jsou po řádcích uložena v jednorozměrném poli, pomáhá nám padding i s posouváním okének při testování jednotlivých hodnot disparit. Na okrajích snímku se

okénko prostě posune do oblasti vyplněné paddingem a nedojde tak k přístupu do nealokované paměti.

Po doplnění paddingu je již volána funkce *disparity*, ve které je obsažen samotný algoritmus. Jako parametry jí jsou předány jednak ukazatele na oba snímky doplněné paddingem a dále číselné hodnoty udávající počet hodnot disparit k otestování, výšku a šířku snímku (bez paddingu) a počet iterací pro zjemňování. Návratovou hodnotou této funkce je pak ukazatel na pole obsahující vypočítané hodnoty disparit pro všechny pixely levého snímku. Všechny hodnoty v mapě disparit jsou posléze vynásobeny vypočítanou konstantou tak, aby bílá barva odpovídala maximální testované hodnotě disparity a černá hodnotě nulové. Ostatní hodnoty disparity pak odpovídají příslušným stupňům šedi.

Struktura samotné funkce *disparity* v zásadě odpovídá struktuře algoritmu, která byla uvedena v části 2.5 a probíhá zde rovněž alokování potřebné grafické paměti a kopírování dat. Samotné výpočty pak probíhají v následujících GPU kernelech:

verticalCostAggregation Agregace cen ve vertikálním směru

horizontalCostAggregation Agregace cen v horizontálním směru

WTA1 Nalezení nejvhodnějších disparit a výpočet míry důvěryhodnosti po agregaci cen

consistencyCheck Kontrola konzistence levého a pravého snímku

verticalRefinement Zjemňování ve vertikálním směru

horozontalRefinement Zjemňování v horizontálním směru

WTA2 Nalezení nejvhodnějších disparit a výpočet penalizace po provedení jedné iterace zjemňování

disparityMedianFilter Aplikace mediánového filtru na mapu disparit

disparityFill Zaplnění neplatných hodnot disparit pomocí hodnot z okolí

Všechny kernely jsou spouštěny s bloky o velikosti 16×16 vláken, tak jak bylo doporučeno v [2]. Počet samotných bloků je vypočítán jednoduše pomocí následujících vztahů. Pro okraj, na který nevychází celý blok, není výpočet prováděn. Důvodem je opět požadavek na co nejmenší větvení v rámci kernelů.

$$\begin{aligned} blocks_{ver} &= \left\lfloor \frac{height}{16} \right\rfloor \\ blocks_{hor} &= \left\lfloor \frac{width}{16} \right\rfloor \end{aligned} \tag{3.1}$$

Postupně alokována jsou pak následující pole, a to opět jak pro levý tak pro pravý snímek. Tato pole již nejsou opatřena paddingem a jejich velikost odpovídá skutečným rozměrům snímků. Na alokovaná pole a jejich význam se pak budeme dále odvolávat v popisu jednotlivých kernelů. Hvězdička v názvu proměnné indikuje příznak zda pole náleží levému či pravému snímku.

***_image_dev** Obsahuje obrazová data snímku včetně paddingu

V_* Obsahuje ceny párování pixelů pro všechny hodnoty disparit

F_* Obsahuje míru důvěryhodnosti párování

d_* Obsahuje mapu disparit pro snímek

N_v_* Obsahuje první koeficient pro výpočet penalizace z vertikální fáze zjemňování

D_v_* Obsahuje druhý koeficient pro výpočet penalizace z vertikální fáze zjemňování

N_h_* Obsahuje první koeficient pro výpočet penalizace z horizontální fáze zjemňování

D_h_* Obsahuje druhý koeficient pro výpočet penalizace z horizontální fáze zjemňování

eps_v_* Obsahuje třetí koeficient pro výpočet penalizace z vertikální fáze zjemňování

eps_h_* Obsahuje třetí koeficient pro výpočet penalizace z horizontální fáze verze zjemňování

V tuto chvíli jsme již připraveni blíže popsat jednotlivé kernely.

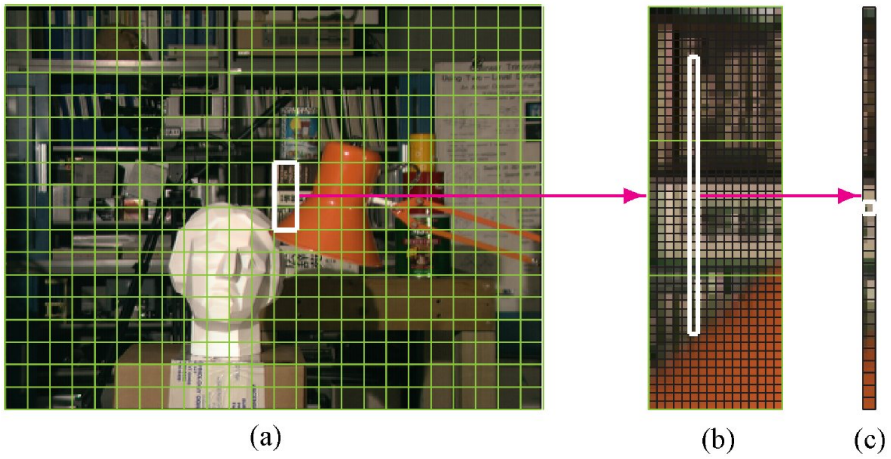
Kernel *verticalCostAggregation*

Jako první bude popsán kernel *verticalCostAggregation*, který provádí výpočet a agregaci cen v podpůrném regionu, a to ve vertikálním směru a pro zadanou hodnotu disparity. Pro otestování všech požadovaných hodnot disparity je tedy nutné tento kernel spustit vícekrát. Vzhledem k tomu, že je tento kernel popisován jako první v pořadí, budou zde blíže vysvětleny i části společné více kernelům.

Prvním krokem vlákna provádějícího kernel je vypočítání jeho souřadnic v rámci snímku. To se děje stejným způsobem, jako bylo uvedeno v příkladu v části 3.1.2. V tomto případě ovšem potřebujeme souřadnice dvojího druhu. První jsou souřadnice, které neberou v úvahu padding. Ty zde budeme potřebovat pro přístup do pole *V_**, kam se ukládají vypočítané ceny. Tyto souřadnice představují proměnné x a y. Souřadnice *img_x* a *img_y* pak představují

souřadnice upravené offsetem o velikosti paddingu. Pomocí nich budeme přistupovat do polí **_image_dev*.

Pro urychlení načítání dat z globální paměti, zde podle doporučení autorů algoritmu využijeme rychlé sdílené paměti grafické karty, která je společná pro všechna vlákna v rámci bloku. Každé vlákno potřebuje pro výpočet ceny svého pixelu ještě 16 pixelů nad a pod daným pixelem. Celý blok tedy potřebuje okénko ze snímku o velikosti 48×16, a to z levého i pravého snímku. V kernelu jsou tedy alokována dvě pole o uvedených rozměrech s názvy *block_region_source* a *block_region_target*. Jako *source* je zde označen snímek, vzhledem ke kterému je cena disparity vypočítávána.



Obrázek 3.5: Ilustrace rozdělení snímků do jednotlivých bloků (b) a vláken (c) Zdroj:[2]

Pro načtení pixelů snímku z globální paměti do sdílené je použit jednoduchý postup. Aby došlo k maximálnímu využití všech vláken bloku při kopírování stačí, aby každé vlákno zkopírovalo mezi oběma pamětmi 3 pixely. Kopírování tedy probíhá ve třech fázích. V první fázi všechna vlákna zkopírují jeden pixel z horní třetiny okénka, ve druhé pixel z prostřední části a ve třetí fázi pixel z třetiny dolní. Tento postup je proveden jak pro *source* tak i pro *target* okénko. Po této operaci již nebude do polí **_image_dev* v rámci jednoho spuštění kernelu dále přistupováno. Po zkopírování obou okének je rovněž nutné provést bariérovou synchronizaci vláken bloku pomocí volání funkce *___syncthreads*.

Poté co je provedena synchronizace, je možné již ze sdílené paměti načíst příslušné pixely a provést výpočet ceny. Cena je vypočítána pomocí rovnic 2.3 a 2.4, kde konstanty Δ_c a Δ_g jsou podle doporučení autorů nastaveny na hodnoty $30, 91$ respektive $28, 21$. Sumy v čitateli a jmenovateli výrazu pro výpočet ceny jsou při průchodu cyklem počítány zvlášť, jejich podíl a tedy celková cena je do pole *V_** uložena až na samotném konci kernelu.

Důležité je ještě zmínit, že pro výpočetně náročnější operace, tedy v tomto případě exponenciální funkci a výpočet podílu jsou použity speciální CUDA matematické funkce *___expf* a *___fdividef*. Tyto funkce jsou speciálně implementované matematické operace pro čísla s plovoucí desetinnou čárkou, které za cenu mírného snížení přesnosti operací nabízí poměrně velké zrychlení výpočtů. Tato výhoda je pak velmi patrná hlavně pro jinak časově náročné dělení, které se v kernelu často vyskytuje.

Kernel *horizontalCostAggregation*

Tento kernel je velmi podobný kernelu předchozímu. Jediným rozdílem je že okénko ze kterého se vypočítávají ceny je orientované horizontálně a má tedy velikost 16×48. Data se do sdílené paměti načítají obdobným způsobem a rovněž cyklus pro výpočet sum v ceně je patřičně upraven. Výsledná cena pro daný pixel a disparitu je pak přičtena k hodnotě již uložené na patřičné pozici v poli *V_**.

Kernel *WTA1*

Tento kernel je prvním ze dvou kernelů, které provádějí vlastní výpočet disparity. Kernel *WTA1* je pak použit po prvotním otestování všech disparit a vyplnění polí *V_**. Každému vláknu je opět přidělen jeden pixel snímku a vlákno projde všechny ceny k danému pixelu vypočítaných disparit. Z nich určí ty dvě nejnižší. Disparita s nejnižší cenou je použita jako výchozí disparita pro zjemňování a druhá nejnižší hodnota je použita pro výpočet důvěryhodnosti nejlepšího výsledku. Čím je odstup nejlepšího výsledku od druhého nejlepšího větší, tím je hodnota důvěryhodnější. Důvěryhodnost je vypočítána podle následujícího vztahu.

$$F^i = \frac{C^i_{min2} - C^i_{min}}{C^i_{min2}} \tag{3.2}$$

Kde C^i_{min} je nejlepší výsledek a C^i_{min2} druhý nejlepší výsledek. F^i je pak míra důvěryhodnosti pro daný pixel i. I v tomto kernelu je využito rychlých aritmetických funkcí GPU. Vzhledem k tomu, že zde nedochází k opakovaným přístupům ke stejným adresám v paměti, není v tomto kernelu využito sdílené paměti.

Kernel *consistencyCheck*

Další poměrně jednoduchý kernel, jež má za úkol kontrolovat konzistenci nalezených disparit. Jak bylo uvedeno v části 2.5, každý pixel by měl mít stejnou hodnotu disparity jako pixel s ním spárovaný, pokud bychom provedli výpočet disparit v obráceném gardu, což činíme. Pokud srovnávané hodnoty disparit nesplňují nerovnost $|D^R_p - D^L_{p'}| \leq 1$, je míra důvěryhodnosti jejich pixelů *F_**

okamžitě nastavena na nulu. Při poslední kontrole konzistence v rámci iterativní části algoritmu jsou pak na nulu nastaveny i samotné hodnoty disparit, které byly shledány nekonzistentními.

Kernel *verticalRefinement*

Stejně jako je do dvou fází rozdělena agregace cen, je na vertikální a horizontální rozděleno i iterativní zjemňování. Tento kernel pak podle svého názvu odpovídá fázi vertikální. Stejně jako při agregaci je zde využito okénko velikosti 48×16, pro jehož zkopírování do sdílené paměti je použit stejný postup. Zde nám ovšem stačí okénko pouze z jednoho snímku, neboť zjemňování se pro pravý i levý snímek provádí zvlášť.

S použitím pixelů z podpůrného regionu, se pak vypočítávají koeficienty *N_v_** a *D_v_**. Ty jsou vypočítány podle následujících vztahů.

$$\mathcal{N}_v(p) = \sum_{q \in \Omega_p} w(p,q) F^{i-1}(q) D^{i-1}(q) \tag{3.3}$$

$$\mathcal{D}_v(p) = \sum_{q \in \Omega_p} w(p,q) F^{i-1}(q) \tag{3.4}$$

Kde F_q^{i-1} a D_q^{i-1} jsou hodnoty důvěryhodnosti a disparity z předchozí iterace zjemňování, v případě první iterace se jedná o hodnoty vypočítané kernelem *WTA1*. Konstanty Δ_c a Δ_g pro výpočet váhy $w(p,q)$ jsou zde rovněž podle doporučení autorů nastaveny na hodnoty $10,94$ a $118,78$. Sumy představující hodnoty $\mathcal{N}_v(p)$ a $\mathcal{N}_v(p)$ jsou pak v průběhu cyklu počítány lokálně, do příslušného pole v globální paměti je uložen až finální výsledek. Do pole *eps_v_** je pak uložen podíl těchto dvou koeficientů.

Kernel *horizontalRefinement*

Tento kernel má k vertikální fázi zjemňování podobný vztah jako *horizontalCostAggregation* k *verticalCostAggregation*. Opět se zde objevuje horizontálně orientované okénko podpůrného regionu, které je kopírováno do sdílené paměti. Pomocí něj se zde vypočítávají koeficienty *N_h_** a *D_h_**. Ty jsou v případě horizontální fáze definovány následovně.

$$\mathcal{N}_h(p) = \sum_{q \in \Omega_p} w(p,q) F^{i-1}(q) \mathcal{D}_v(q) \varepsilon_v(q) \tag{3.5}$$

$$\mathcal{D}_h(p) = \sum_{q \in \Omega_p} w(p,q) F_q^{i-1} \mathcal{D}_v(q) \tag{3.6}$$

Hodnoty v *eps_h_** pak opět odpovídají podílům *N_h_** a *D_h_**.

Kernel *WTA2*

Kernel *WTA2* plní obdobnou funkci jako kernel *WTA1*, ovšem s tím rozdílem, že se zde k k hodnotám cen z pole *V_** připočítává penalizace Λ. Ta je

vypočítána dle následujícího vztahu.

$$\Lambda = \alpha \mathcal{D}_h(p)|\varepsilon_h(p) - d| \tag{3.7}$$

Kde α je podle doporučení nastavena na hodnotu 0,81 a hodnota d odpovídá disparitě hodnoty, pro kterou je penalizace vypočítávána. Tato penalizace se pak při porovnávání přičítá k hodnotám cen párování z pole *V_**, jinak je princip včetně výpočtu míry důvěryhodnosti totožný s kernelem *WTA1*. Ani zde pak není využito žádné sdílené paměti.

Kernel *disparityMedianFilter*

Předposlední zmiňovaný kernel je již součástí fáze post-processingu a jeho úkolem je na získanou mapu disparity aplikovat mediánový filtr o velikosti 3×3. Princip mediánového filtru je jednoduchý. Okolo pixelu, na který je filtr právě aplikován, je načteno okénko o zadané velikosti (v tomto případě 3×3), hodnoty z okénka jsou seřazeny a hodnota pixelu, která se nachází uprostřed seřazeného pole je zapsána na patřičnou pozici do výstupní matice.

V případě tohoto kernelu, je opět využito sdílené paměti. Tentokrát je ze snímku pro každý blok načteno okénko o velikosti 18×18 pixelů tak, aby každý pixel bloku měl k dispozici okolí požadované velikosti. Princip načítání je v tomto případě odlišný od načítání v předchozích kernelech a probíhá následujícím způsobem.

1. Každé vlákno zkopíruje pixel jemu odpovídající do prostřední části sdílené matice
2. Vlákna odpovídající rohům výřezu zkopírují pixely do rohů sdílené matice
3. Vlákna se souřadnicemi na pravém okraji bloku zkopírují pixely do prázdné části pravého okraje sdílené matice
4. Vlákna se souřadnicemi na levém okraji bloku zkopírují pixely do prázdné části levého okraje sdílené matice
5. Vlákna se souřadnicemi na horním okraji bloku zkopírují pixely do prázdné části horního okraje sdílené matice
6. Vlákna se souřadnicemi na dolním okraji bloku zkopírují pixely do prázdné části dolního okraje sdílené matice

Po kopírování do sdílené paměti opět musí být provedena bariérová synchronizace všech vláken bloku. Po synchronizaci si každé vlákno ze sdílené paměti načte jemu odpovídající okénko velikosti 3×3 do lokálního jednorozměrného pole a řazení je pak provedeno pomocí jednoduchého bubble-sortu. Prostřední prvek pole je pak zapsán do výstupního pole nacházejícího se v globální paměti.

Kernel *disparityFill*

Druhým a posledním kernelem, který se věnuje dodatečnému zpracování mapy disparit, je kernel *disparityFill*. Jeho účelem je vyplnění oblastí mapy, které byly označeny při posledním běhu kernelu *WTA2* jako neplatné. K tomu opět slouží podpůrný region, tentokrát horizontálně orientovaný, který je známým způsobem opět načten do sdílené paměti. Pro neplatné pixely mapy je jejich hodnota vypočítána následujícím způsobem.

V cyklu je v obou směrech postupováno směrem od pixelu, který je právě vyplňován. Ve chvíli kdy se na obou stranách narazí na platnou hodnotu, je do výstupního pole na patřičnou pozici zapsána hodnota, která se nachází blíže. Pokud je hodnota pixelu již na začátku vyplňování platná, je automaticky zkopírována do výstupního pole.

3.2.6 Modul *cuda_sncc*

Posledním modulem, který implementuje algoritmus provádějící stereo-matching a vytvářející mapu disparit je modul *cuda_sncc*, jenž je implementován třídou CudaSNCC. Implementovaným algoritmem je pak dvouúrovňová korelační metoda představená v části 2.4. Vzhledem k tomu, že pro velikosti okének jsou použity hodnoty doporučené autory algoritmu, má modul pouze jeden parametr k nastavení, kterým je parametr **num_disparities**.

Modul sám je pak velmi podobný předchozímu modulu *cuda_asw*. V metodě *do_special_single_step* se vstupní snímky opět konvertují na stupně šedi a přidá se k nim stejně veliký padding. Poté je opět volána funkce *disparity*, ve které je implementováno tělo celého algoritmu. Ten je implementován podle popisu v [9]. I v případě tohoto modulu opět vytváříme mapu disparit vzhledem k oběma snímkům, abychom mohli stejně jako v předchozím případě provést kontrolu konzistence nalezených hodnot. Prováděný postup můžeme popsat následujícím způsobem.

1. Inicializace hodnot
   a) Aplikace box-filteru o velikosti 3×3 na oba snímky
   b) Výpočet směrodatných odchylek všech okének z předchozího kroku
   c) Nastavit nejlepší korelaci pro všechny pixely: $\rho_{max} = -\infty$
2. Pro každou hodnotu disparity d provést:
   a) Výpočet korelace pro všechny pixely za pomoci předpočítaných hodnot
   b) Aplikace box-filteru o velikosti 5×9 na hodnoty korelací
   c) Pokud je hodnota korelace pro daný pixel větší než ρ_{max} pak $\rho_{max} = \rho_p$ a $D_p = d$

3. Kontrola konzistence
4. Aplikace mediánového filtru
5. Vyplnění neplatných hodnot, za pomoci hodnot z okolí

Veškeré aplikace filtrů a výpočty korelací jsou i v tomto modulu realizovány pomocí CUDA kernelů. V případě tohoto modulu je jich 8 a mají následující názvy a účely.

boxFilter3x3 Aplikuje na snímek box-filter o velikosti 3×3 a počítá tak aritmetické průměry z pixelů z okolí pixelu

sdFilter3x3 Počítá směrodatné odchylky pixelů za použití vypočítaných středních hodnot

setCorrelation Nastavuje hodnoty korelací pro všechny pixely na hodnotu $-\infty$

correlationFilter3x3 Počítá korelace pixelů za pomoci jejich okolí dané velikosti, maximální velikost je 33×33

boxFilterMxN Aplikuje box-filter dané velikosti, v tomto případě na vypočítané hodnoty korelací pixelů

WTA Porovnává novou hodnotu korelace s nejlepší dosud nalezenou

consistencyCheck Kontroluje konzistenci nalezených hodnot disparit

disparityMedianFilter Stejný mediánový filtr jako u modulu *cuda_asw*

disparityFill Vyplňuje neplatné hodnoty

Spouštěny jsou pak opět v blocích o velikosti 16×16 jako v předchozím případě. Kromě obrazových dat pak funkce ke své činnosti potřebuje následující informace, které jsou uloženy v následujících polích. Hvězdička v názvu je opět použita pro příznak levého či pravého snímku.

means_* Obsahuje výsledek aplikace box-filteru na vstupní snímky

sds_* Obsahuje směrodatné odchylky okének vstupních snímků

bestCorrelation_* Obsahuje nejlepší zatím nalezenou hodnotu korelace

disparityMap_* Obsahuje nalezené hodnoty disparit

cor_* Obsahuje korelace pro danou disparitu vypočítané kernelem *correlationFilter3x3*

cor2_* Obsahuje korelace po aplikaci box-filteru $M \times N$

Kernely, které se liší od těch použitých v předchozím modulu budou nyní opět detailněji popsány. Vzhledem k tomu, že všechny kernely jsou si poměrně podobné, nebudou popisovány zvlášť jak o předchozího modulu.

Jako první bude přiblížen kernel realizující box-filter. Prvním krokem po spuštění kernelu je načtení potřebných obrazových dat do sdílené paměti. K tomu je použit stejný způsob jako u kernelu určeného pro výpočet mediánového filtru v sekci 3.2.5. Poté každé vlákno agreguje hodnoty z jejího okolí v lokální proměnné *sum* a po vypočítání aritmetického průměru hodnotu uloží do výstupního pole.

Stejný postup používá i kernel *boxFilterMxN*. Pole alokované ve sdílené paměti má ovšem velikost $(16 + M - 1) \times (16 + N - 1)$. Data jsou do něj kopírována obdobným způsobem, ovšem do kopírování okrajů regionu je zapojeno více vláken, tak aby jedno vlákno kopírovalo vždy jednu hodnotu. Do oblasti $\left\lfloor \frac{N}{2} \right\rfloor$ sloupců na levém okraji tak kopírují vlákna, která mají souřadnici v ose x menší než $\left\lfloor \frac{N}{2} \right\rfloor$. Obdobně jsou zkopírovány i další oblasti.

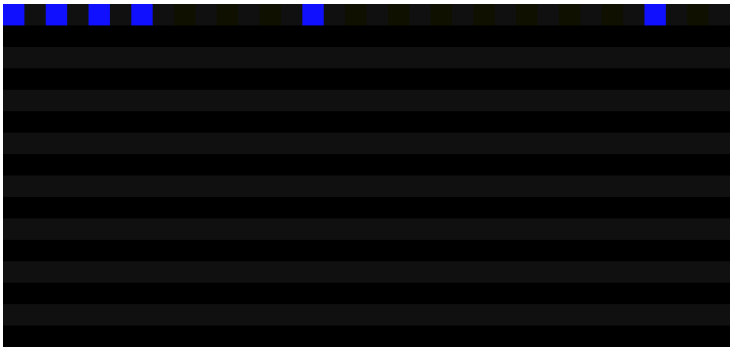
Podobný postup je pak aplikován i u kernelu počítajícího směrodatnou odchylku. Načítání do sdílené paměti je stejné jako v případě box-filteru, liší se pouze část agregace sumy.

Kernel *WTA* je v tomto případě mnohem jednodušší, než v případě algoritmu předchozího. Zde stačí pouze porovnat hodnoty v polích *bestCorrelation_** a *cor2_** a případně zapíše do mapy disparit. Stejně tak je zjednodušen i kernel *consistencyCheck*. Již zde není nutné nastavovat hodnotu důvěryhodnosti, kterou zde nepotřebujeme, a tak pouze provede porovnání disparit z obou snímků a v případě rozdílu nastaví hodnoty disparit na 0. Kernel *disparityFill* je pak použit beze změny.

3.2.7 Modul *dimenco_out*

Tento modul je určen pro přípravu výstupu dříve představených modulů k zobrazení na auto-stereo monitoru značky Dimenco. Tento monitor umožňuje zobrazit obraz obohacený o hloubkovou informaci bez použití speciálních brýlí. K tomu potřebuje zdroj obrazu ve formátu „side-by-side“, kde se na levé straně nachází barevný obraz a na pravé černobílá hloubková mapa. Ve spolupráci s libyuri modulem *combine* jsou moduly, implementované v rámci této práce, schopny takovýto obraz poskytnout.

Samotný side-by-side obraz ovšem ke 3D zobrazení nestačí. Monitor potřebuje informace, že se skutečně o takový obraz jedná a pravou polovinu obrazu použít pro vytvoření 3D efektu. Pokud ovšem pomocí našich modulů vytvořený videozáznam spustíme pomocí softwarového přehrávače dodávaného k monitoru, k zapnutí 3D efektu dojde. Pokud při přehrávání tímto přehrávačem provedeme print-screen obrazovky, zjistíme že jisté pixely v prvním řádku jsou nastaveny na modrou barvu. Konkrétně se jedná o některé pixely



Obrázek 3.6: Prvních 6 pixelů nastavených Dimenco přehrávačem

v rozmezí 0 – 508 a jeho barevné kanály ve formátu *RGB24* jsou nastaveny na následující hodnoty.

$$R = 16 \tag{3.8}$$
$$G = 16 \tag{3.9}$$
$$B = 255 \tag{3.10}$$

Na obrázku je rovněž vidět, že řádky ve snímku z přehrávače jsou prokládané. Toto prokládání je také nutnou podmínkou k tomu, aby se monitor přepnul do 3D režimu. Prokládání je pak vytvořeno jednoduchým způsobem, kdy jsou do snímku odcházejícího na výstup modulu zkopírovány pouze sudé řádky. Všechny pixely v prvním řádku snímku jsou pak nastaveny identicky s prvním řádkem snímku získaného print-screenem z přehrávače. Pokud byly změněny pouze modré pixely a ostatní byly ponechány tak jak jsou, k přepnutí do 3D módu nedojde. V případě kdy byl použit celý první řádek snímku z přehrávače k přepnutí okamžitě došlo.

Samotný modul *dimenco_out* je pak opět podtřídou třídy *SpecializedIOFilter* a má tedy jeden vstup a jeden výstup. Příslušné pixely jsou pak na uvedenou barvu nastaveny pouze v případě, kdy vstupující snímek má rozlišení 3840×1080 nebo 3840×2160 pixelů. V těchto rozlišeních totiž Dimenco monitor dokáže pracovat a vytvořit požadovaný 3D efekt. Pokud je rozlišení snímku jiné, modul žádné úpravy neprovede a snímek beze změn pošle na svůj výstup. Pokud se tak stane, uživatel je o tomto stavu srozuměn ve formě hlášení v logu aplikace. Pro správnou činnost modulu je tedy nutné aby mu předcházel modul *scale*, který snímek na potřebnou velikost zvětší.

Kapitola 4

Testování

V této kapitole bude popsáno testování modulů pro výpočet mapy disparit, které byly uvedeny v předchozí kapitole. První část testování bude provedena na statickém, již rektifikovaném snímku, a druhá část zaměřená na výkon modulů bude provedena z živého vstupu dvou webkamer. Ty budou zkalibrované a jimi poskytnuté snímky rektifikované rovněž pomocí implementovaných modulů. U obou způsobů testování pak bude rovněž uvedeno, jaké další libyuri moduly byly pro testování použity a jakým způsobem byly zapojeny.

4.1 Testování statickými snímky

Testování statickými snímky bude provedeno na dvou scénách, které pochází ze sad vytvořených skupinou počítačového vidění z univerzity Middlebury[13]. Jedná se o scény vytvořené roku 2003 nazvané *cones* a *teddy*, pro které byla přesným způsobem vytvořena hloubková mapa a již jsou patřičně rektifikovány. Pro testování byly použity snímky o rozlišení 900 × 750 s maximální hodnotou disparity 128.

Obrázek 4.1: Scéna *cones*

Pro otestování byl použit následující postup. Nejprve jsou ze souborů ve formátu PNG načteny jak levý, tak pravý snímek. Tyto snímky jsou dekódovány a odeslány na vstupy modulu pro výpočet mapy disparit. Výstup tohoto

Obrázek 4.2: Scéna *teddy*

modulu, tedy mapa disparit a v tomto případě levý snímek, jsou dále spojeny do jednoho snímků a tento snímek je posléze opět uložen na disk ve formátu PNG. K tomuto postupu jsou použity následující libyuri moduly.

file_picker Slouží pro načtení souboru z filesystému

png_decoder Dekomprimuje snímek zkomprimovaný pomocí algoritmu PNG

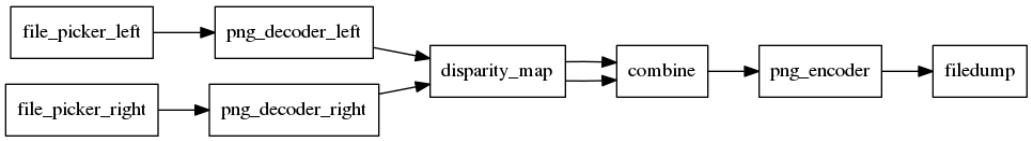
disparity_module Vytvoří mapu disparit, jedná se o jeden z modulů *opencv_sgbm*, *opencv_cudabm*, *cuda_asw* nebo *cuda_sncc*

combine Spojuje vstupní snímky do jednoho

png_encoder Komprimuje snímek na vstupu do formátu PNG

filedump Uloží snímek do požadované cesty ve filesystému

Moduly *file_picker* a *png_decoder* se v systému vyskytují dvakrát, neboť je nutné nezávisle načíst dva různé snímky. U modulu *file_picker* je nutné nastavit parametr *format* na hodnotu PNG. Výchozí hodnota tohoto parametru je nastavena na hodnotu JPEG a tento parametr musí odpovídat formátu ve kterém jsou snímky uloženy, jinak je modul při načítání přeskočí. Moduly v systému jsou pak zapojeny následujícím způsobem.



Obrázek 4.3: Zapojení modulů

Pro objektivní zhodnocení kvality vytvořených hloubkových map, byly vytvořené hloubkové mapy pomocí metriky PSNR (peak signal to noise ratio), která je implementována v nástroji *compare*, což je součást známé sady pro

zpracování obrázků *ImageMagick*[14]. Hodnota PSNR je udávána ve decibelech a čím je tato hodnota vyšší, tím si jsou oba porovnávané snímky podobnější. Při porovnání zcela totožných snímků pak hodnota PSNR odpovídá nekonečnu.

Samotná hodnota PSNR pro dva různé snímky I a K o N pixelech se počítá pomocí následujících vztahů.[15].

$$PSNR = 10 \log \left(\frac{MAX^2}{\varepsilon^2(i)} \right) \tag{4.1}$$

$$\varepsilon^2(i) = \frac{1}{N} \sum_{i=1}^{N} (I(i) - K(i))^2 \tag{4.2}$$

Hodnoty $I(i)$ a $K(i)$ odpovídají intenzitám jednotlivých pixelů. Hodnota MAX je pak maximální hodnota, které jeden pixel může nabývat. V případě 8 bitové barevné hloubky by se jednalo o hodnotu 255.

4.1.1 Modul *opencv_sgbm*

Jako první byl otestován modul *opencv_sgbm*. Parametr určující počet disparit byl nastaven na hodnotu 128, tak jak bylo uvedeno u zdroje snímku.[13] Výsledek pro velikost okénka 3 je možné vidět na obrázcích níže.

Obrázek 4.4: Scéna *cones* zpracovaná modulem *opencv_sgbm*

Na výsledných obrázcích můžeme vidět že neplatné (černé) pixely se nachází hlavně v oblastech kde dochází k překryvu objektů ve scéně. Tento je v velmi dobře vidět u domečku ve druhé scéně. Pixely, které se nepodařilo úspěšně spárovat, se vyskytují rovněž v oblasti napravo od medvídka. Je pravděpodobné, že jejich špatné párování bylo způsobeno právě jednolitou světlou plochou, která se zde nachází, a blízkostí okraje snímku.

Pokud budeme velikost okénka zvyšovat, můžeme si povšimnout následujícího efektu. Se zvětšujícím se okénkem se postupně vyhlazují hrany objektů ve scéně a mizí malé oblasti neplatných pixelů. U okénka o velikosti 11 se

Obrázek 4.5: Scéna *teddy* zpracovaná modulem *opencv_sgbm*

ovšem kolem hran objeví poměrně velké neplatné oblasti, které se dříve neobjevovaly a nejsou způsobeny překryvem. Pokud velikost nastavíme na hodnotu 13, algoritmus přestává správně pracovat. V dokumentaci OpenCV je potom u parametru nastavujícího velikost okénka uvedeno, že „by se měla nacházet v intervalu 3 – 11“.[12]

Další možnost, kterou máme, je zapnout přesnější režim algoritmu pomocí parametru *hh_mode*. Na zpracování scény *teddy*, na které byla tato možnost otestována ovšem není velké zlepšení patrné. Neplatná oblast vpravo od medvídka se objevuje i v tomto případě a neplatné oblasti kolem hran se objeví už u okénka o velikosti 9. U velikosti 11 je pak degradace ještě výraznější.

Hodnoty PSNR pro obě scény, otestované velikosti okének a nastavení parametru *hh_mode* jsou uvedeny v následujících tabulkách. Naměřené hodnoty pak potvrzují subjektivní pozorování provedená na výsledcích. Ze zvětšováním okénka kvalita výsledné mapy klesá a při zapnutí parametru *hh_mode* se výraznější degradace objevuje už při použití okénka o velikosti 9.

	3	5	7	9	11
hhmode=0	12,3217	12,2876	12,2618	12,1877	12,5638
hhmode=1	12,3557	12,3273	12,2813	11,3961	10,4112

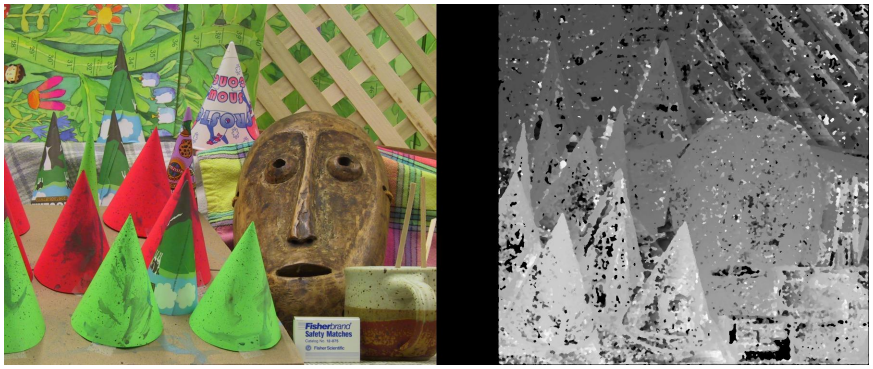
Tabulka 4.1: Hodnoty metriky PSNR pro scénu *cones* zpracovanou modulem *opencv_sgbm*

	3	5	7	9	11
hhmode=0	12,4338	12,2139	12,1172	12,1052	11,5194
hhmode=1	12,0063	11,8774	11,8411	11,1303	10,3024

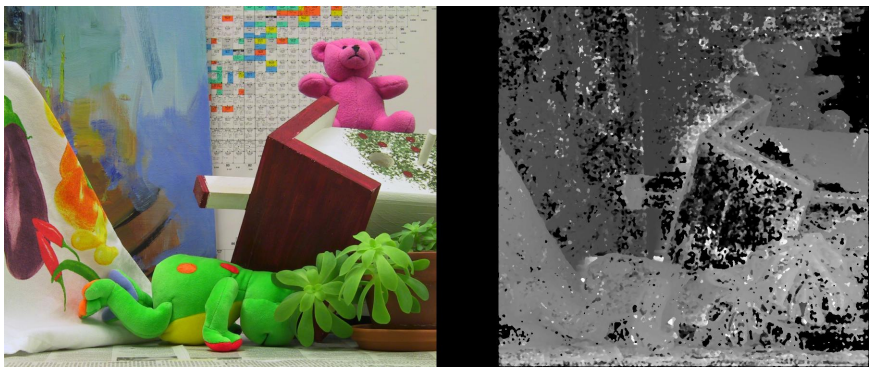
Tabulka 4.2: Hodnoty metriky PSNR pro scénu *teddy* zpracovanou modulem *opencv_sgbm*

4.1.2 Modul *opencv_cudabm*

Druhým testovaným modulem je modul *opencv_cudabm*. Jeho nastavení je jednodušší než u předchozího modulu, nastavujeme pouze velikost okénka a počet disparit o otestování. Tyto hodnoty byly opět nastaveny na hodnoty 3 (velikost okénka) a 128 (počet disparit). Výsledky tohoto nastavení pro obě testovací scény jsou prezentovány na obrázcích 4.6 a 4.7.



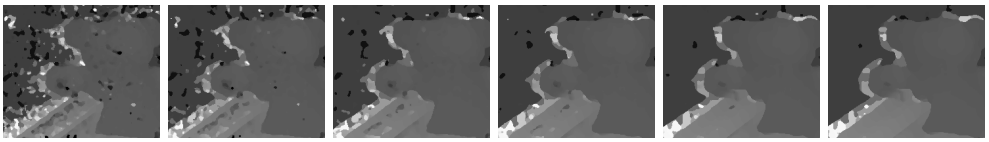
Obrázek 4.6: Scéna *cones* zpracovaná modulem *opencv_cudabm*



Obrázek 4.7: Scéna *teddy* zpracovaná modulem *opencv_cudabm*

Na výsledcích jasně vidíme, že výsledek je výrazně horší než v předchozím případě. V oblastech překryvu se tu neobjevují pouze neplatné pixely ale oblasti které svou udávanou hodnotou disparity neodpovídají skutečnosti. Dochází tak jakémusi „rozpití“ hran, což je dobře vidět pokud porovnáme oblast kolem střechy domečku s výsledky předchozího modulu. U výsledků modulu *opencv_sgbm* bylo díky neplatným pixelům jasně patrné, kde se hrany nacházejí. U tohoto modulu nejsou hrany díky vyskytujícím se artefaktům tak dobře viditelné. Rovněž oblasti s jednolitější barvou jsou v tomto případě problematické

Nyní otestujeme jaký má na výsledek vliv zvětšení okénka. Stejně jako v předchozím případě vyzkoušíme vliv tohoto parametru na scéně *teddy*, tentokrát s hodnotami o velikosti 5, 7, 9, 11, 13 a 15.



Obrázek 4.8: Výřez scény *teddy* zpracované s velikostí okénka 5, 7, 9, 11, 13 a 15 pixelů

Z výsledků na obrázku 4.8 můžeme vidět, že se zvětšováním okénka se celkový dojem z výsledku zlepšuje. Jednolité plochy se spojují a artefakty s neplatnými nebo neodpovídajícími pixely z nich mizí. Objevuje se však jiný jev, a to rozšiřování oblastí kolem hran. Pokud porovnáme první a poslední obrázek, tak zejména u hran střechy domečku k tomuto jevu dochází. Hranice oblasti v mapě disparit tu výrazně překračují hranice skutečného objektu ve zdrojovém snímku. Tato pozorování pak opět potvrzují naměřené hodnoty PSNR. Nejlepší hodnoty bylo dosaženo při použití okénka s velikostí 9. U větších okének je již naměřená hodnota nižší, což odpovídá dojmu z výsledné hloubkové mapy.

vel. ok. / p	5	7	9	11	13	15
PSNR / dB	11,8453	11,9552	11,9814	11,9556	11,8912	11,8174

Tabulka 4.3: Hodnoty metriky PSNR pro scénu *cones* zpracovanou modulem *opencv_cudabm*

vel. ok. / p	5	7	9	11	13	15
PSNR / dB	12,0212	12,0854	12,1025	12,0889	12,0679	12,0428

Tabulka 4.4: Hodnoty metriky PSNR pro scénu *teddy* zpracovanou modulem *opencv_cudabm*

4.1.3 Modul *cuda_asw*

V této sekci bude provedeno otestování modulu *CudaASW*. Zdrojové snímky a počet testovaných disparit zůstávají opět stejné. Parametr *iterations* určující počet iterací zjemňovací části algoritmu je nastaven na hodnotu 6, jak doporučují autoři algoritmu, kteří s touto hodnotou dosahovali nejlepšího výsledku.[2] Jeho hodnota pak bude postupně snižována, aby bylo možné ilustrovat vliv tohoto parametru na kvalitu výsledné mapy disparit.

Obrázek 4.9: Scéna *cones* zpracovaná modulem *cuda_asw*

Obrázek 4.10: Scéna *teddy* zpracovaná modulem *cuda_asw*

Jak vidíme, na výsledných snímcích se opět objevují oblasti s neplatnými pixely, a to na podobných místech jako u předchozích modulů. Stejně jako u modulu *opencv_cudabm* se zde pak objevují neplatné pixely v oblasti střechy domečku ve scéně *teddy*, zatímco modul *opencv_sgbm* si s touto oblastí poradil výrazně lépe. I tomuto modulu pak problémy způsobují zejména větší oblasti jednolité barvy.

Pro ilustraci vlivu parametru *iterations* nyní opět použijeme stejný výřez ze scény *teddy*.

Obrázek 4.11: Výřez scény *teddy* zpracované s počtem zjemňovacích iterací 1, 2, 3, 4, 5 a 6

Jediný alespoň trochu výrazný rozdíl je vidět mezi počtem iterací 1 a

2. Při bližším pohledu je vidět, že došlo k mírnému vyhlazení hran střechy a medvídka. Další zvyšování počtu iterací se tak alespoň v tomto případě jeví jako zbytečné. Vliv počtu iterací na výpočetní náročnost bude změřen a popsán v následující části, kde bude provedeno testování s živým video vstupem.

iterace	1	2	3	4	5	6
PSNR / dB	12,9371	13,2862	13,2066	13,0862	12,8713	12,7930

Tabulka 4.5: Hodnoty metriky PSNR pro scénu *cones* zpracovanou modulem *cuda_asw*

iterace	1	2	3	4	5	6
PSNR / dB	12,4999	12,7848	12,7463	12,7393	12,7180	12,7267

Tabulka 4.6: Hodnoty metriky PSNR pro scénu *cones* zpracovanou modulem *cuda_asw*

V tabulkách jsou opět uvedeny hodnoty PSNR porovnávající referenční mapu disparity a mapu vytvořenou modulem. I v tomto případě subjektivní pozorování odpovídá naměřeným hodnotám a zdá se, že 2 zjemňující iterace budou při použití tohoto modulu naprosto dostačující.

4.1.4 Modul *cuda_sncc*

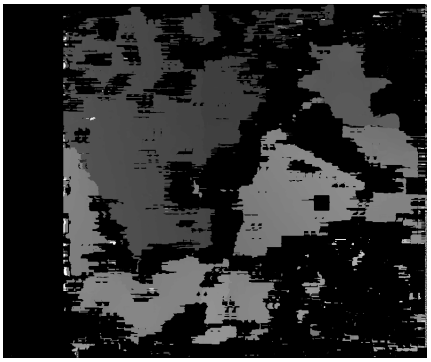
Poslední modul který zbývá k otestování je modul využívající vzájemné korelace *cuda_sncc*. Podle doporučení autorů je velikost druhého box-filtru nastavena na hodnotu 5×9. Výsledné mapy disparit pro obě testovací scény můžeme vidět níže.

Obrázek 4.12: Scéna *cones* zpracovaná modulem *cuda_sncc*

Obrázek 4.13: Scéna *teddy* zpracovaná modulem *cuda_sncc*

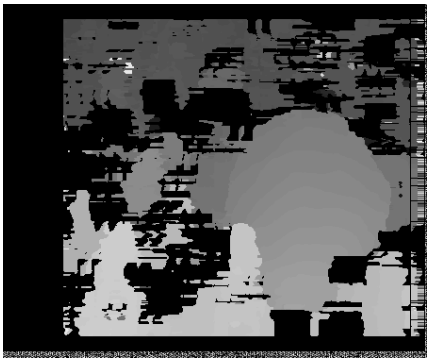
Výsledek je bohužel velmi špatný. Mapy disparit obsahují pouze malé oblasti platných hodnot a dodatečné zpracování v podobě mediánového filtru a vyplňování děr nemá šanci situaci zlepšit. Vyzkoušíme tedy zvětšit velikost filtru na největší možnou hodnotu (33×33). Výsledek pro toto nastavení je pak předveden na obrázku 4.14. Výsledek samotný je o mnoho lepší než při předchozím nastavení a již zde jsou patrné objekty, které se ve scéně nachází. Stále ovšem ani zdaleka nestačí na výsledky předchozích tří modulů. Rovněž se zdá že na kvalitu výsledku má vliv rovněž poměr velikosti zdrojového snímku a aplikovaného filtru.

Vyzkoušíme tedy algoritmus aplikovat na snímky s menší velikostí. V tomto případě použijeme scénu *cones* a snímky o poloviční rozměrech (450×375).



Obrázek 4.14: Mapa disparit pro scénu *teddy* zpracovaná s filtrem o velikosti 33×33

Na polovinu rovněž snížíme i počet testovaných disparit, velikost filtru pak ponecháme na nejvyšší možné hodnotě. Na výsledku, který lze vidět na obrázku 4.15, je opět vidět značné zlepšení, ale kvality předchozích opět nedosahuje. V článku, který algoritmus popisuje a v hodnocení University of Middlebury, jsou uvedené výsledky rovněž mnohem kvalitnější. Je ovšem možné, že na tyto výsledky bylo ještě aplikováno dodatečné zpracování, které ovšem není v článku popsáno.



Obrázek 4.15: Mapa disparit pro scénu *cones* o poloviční velikosti zpracovaná s filtrem o velikosti 33×33

4.2 Testování video vstupem

V této části budou všechny čtyři moduly otestovány živým video vstupem ze dvou webkamer Logitech HD Pro Webcam C920. Tyto kamery byly zkalibrovány pomocí modulu *opencv_stereocalib* a snímky z nich získané následně rektifikovány modulem *opencv_rectify*. Testování proběhlo na počítači vyba-

veném čtyřjádrovým procesorem Intel Core i5-4690K o frekvenci 3,5GHz a GPU GeForce GTX 960 s 4GB grafické RAM.

Všechny moduly byly otestovány stejnou vstupní sekvencí snímků o rozlišení 1280×480. Jedná se tedy o snímky z obou kamer uložené ve formátu „side-by-side“. Tyto snímky byly pořízeny pomocí následující konfigurace modulů. Moduly, které byly představeny v předchozí části, nebudou znovu popisovány, pouze se objeví ve schématu zapojení.

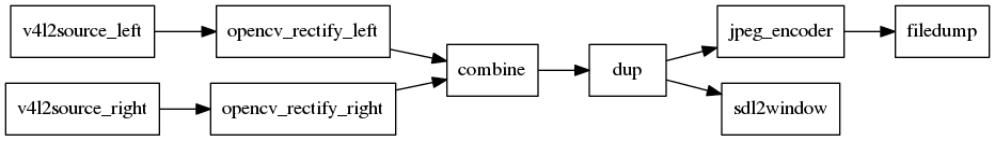
v4l2source Představuje V4L zařízení, které produkuje snímky ve formátu YUYV

opencv_rectify Rektifikuje levý nebo pravý snímek pomocí transformační matice ze souboru *undistort.yml*

jpeg_encoder Komprimuje snímek pomocí algoritmu JPEG, zde je použit z důvodu vyšší rychlosti oproti modulu *png_encoder*

dup Duplikuje snímek na neomezené množství výstupů

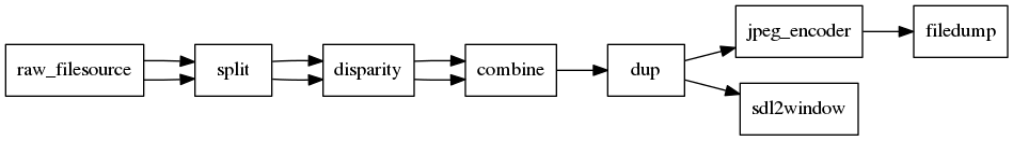
sdl2window Slouží k zobrazení zachytávané sekvence



Obrázek 4.16: Zapojení modulů pro uložení sekvence snímků

Výsledný testovací videozáznam o frekvenci 30 snímků za sekundu je k dispozici na přiloženém DVD (soubor *bear_test.mp4*) nebo na serveru YouTube[1].

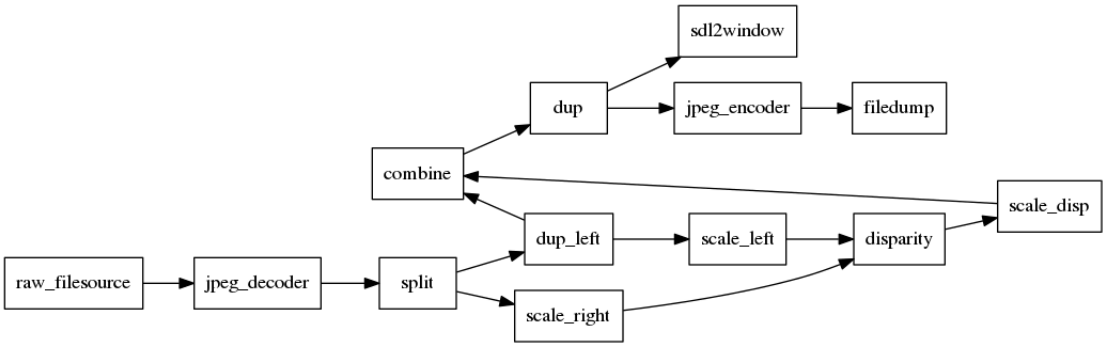
Pro otestování samotných modulů pak byla použita dvě různá zapojení. První zapojení jednoduše rozdělí vstupní snímek na levou a pravou část, které jsou dále předány jednomu ze čtyř testovaných modulů. Jeho výstup je pak zpracován stejným způsobem jako v zapojení výše (zobrazení v okně a uložení snímku do souboru). Počet testovaných disparit je v tomto případě 64.



Obrázek 4.17: Zapojení modulů pro zpracování testovací sekvence

[1] `https://youtu.be/wnmbtW4MeBU`

Druhý způsob zapojení má za úkol zjistit, jaký vliv na kvalitu mapy disparit a rychlost zpracování by měl postup, kdy jsou oba zdrojové snímky zmenšeny. Při zmenšení snímků by pak stačilo pro obsáhnutí stejné vzdálenosti otestovat menší množství disparit (poloviční oproti předchozímu zapojení, tedy 32), ovšem za cenu mírného snížení přesnosti. Mapa vypočítaná modulem je pak znovu zvětšena a modulem *combine* znovu spojena s původním nezmenšeným snímkem.



Obrázek 4.18: Zapojení modulů pro zpracování testovací sekvence se zmenšením vstupních snímků

Vedle testování výkonu, byl pak proveden ještě jeden test, který měl za úkol prověřit stabilitu vytvářené mapy. K tomuto testu byl použit odlišný videozáznam[2], který obsahuje pouze statickou scénu. Způsob provedení testu byl pak následující. První zpracovaný snímek byl považován za referenční a zbývající snímky s ním byly opět porovnány prostřednictvím metriky PSNR. Ze získaných hodnot byl posléze vypočítán aritmetický průměr a směrodatná odchylka. Tyto hodnoty jsou opět uvedeny v tabulkách u jednotlivých modulů.

4.2.1 Modul *opencv_sgbm*

Stejně jako při testování statickými snímky, byl jako první otestován modul *opencv_sgbm*. Postupně byla vyzkoušena různá nastavení parametrů a jejich vliv na rychlost zpracování. Rychlost zpracování je u tohoto i dalších modulů měřena ve snímcích za sekundu. Jako první bylo provedeno měření na zapojení bez zmenšování vstupních snímků a to pro velikosti okénka 3 – 11. Pro každou hodnotu velikosti byl rovněž otestován vliv parametru *hh_mode*.

Naměřené hodnoty jsou uvedeny v následujících tabulkách. Pro měření výkonu modulu při zpracovávání zmenšených snímků byla rychlost zdroje zvýšena na hodnotu 120 snímků za sekundu, aby byl rozdíl mezi zdrojem a zpracováním dobře patrný. Původní rychlost 30 snímků za sekundu modul při tomto nastavení zvládal beze ztrát.

[2] `https://www.youtube.com/watch?v=R4rdI-IOPZA` nebo soubor *stability_test.mp4*

	3	5	7	9	11
hhmode=0	12,27	12,44	12,44	12,45	12,86
hhmode=1	8,23	8,49	8,53	8,85	9,06

Tabulka 4.7: Počty snímků za sekundu pro různá nastavení parametrů při zpracování snímků o původní velikosti 640 × 480 pixelů a 64 hodnot disparit modulem *opencv_sgbm*

	3	5	7	9	11
hhmode=0	69,1	71	69,61	70,51	71,37
hhmode=1	51,12	51,21	51,34	52,01	53,06

Tabulka 4.8: Počty snímků za sekundu pro různá nastavení parametrů při zpracování snímků zmenšených na velikost 320 × 240 pixelů a 32 hodnot disparit modulem *opencv_sgbm*

Z obou tabulek je patrné, že zvětšování okénka má za následek velmi mírný nárůst výkonu a použití přesnějšího režimu algoritmu pak výkon snižuje, a to zhruba o jednu třetinu. Kvalitu zpracování a porovnání mezi oběma způsoby zapojení modulů budeme ilustrovat následujícími dvěma snímky, které byly získány s velikostí okénka 7 a zapnutým parametrem *hh_mode*.

Obrázek 4.19: Snímky z testovacího videozáznamu zpracované modulem *opencv_sgbm* bez a za použití zmenšených vstupních snímků

Na horním snímku, který byl pořízen ze zdrojových snímků o původní ve-

likosti můžeme vidět problematické oblasti v pozadí scény. Na videozáznamu[3] je pak vidět, že se tyto oblasti poměrně rychle mění, což nepřispívá ke stabilitě obrazu. Zajímavá je pak oblast zavřeného víka piana v pozadí, kterou algoritmus identifikoval jako blízko se nacházející objekt.

Při pohledu na dolní snímek, kde byla mapa disparit získána ze zmenšených snímků, působí pozadí o něco lepším dojmem, protože se zde již nenachází tolik neplatných oblastí a pozadí tak působí jednolitějším dojmem. Na výsledném videozáznamu[4] se pak jako lepší jeví i stabilita celé scény. I zde se ovšem nachází chybná oblast v prostoru piana a hrany medvěda nejsou kvůli dodatečnému zvětšení mapy tak ostré jako v předchozím případě.

Všechny výstupní soubory pro testovaná nastavení parametrů jsou též k dispozici na přiloženém DVD nebo na YouTube[5].

V následujících tabulkách jsou uvedeny průměrné hodnoty PSNR získané při porovnání prvního snímku se snímky ostatními. Z naměřených hodnot lze vidět, že zvětšování okénka má za následek zlepšení stability celé sekvence, ovšem i zde je vidět vliv degradace kvality při nastavení velikosti okénka na hodnotu 11. Použití zmenšeného snímku má na stabilitu sekvence rovněž kladný vliv.

	3	5	7	9	11
prům. PSNR	19,1170	19,7091	20,5774	21,0978	19,6429
směr. odch.	0,1155	0,1546	0,1602	0,1996	0,2354

Tabulka 4.9: Průměrné hodnoty PSNR a směrodatné odchylky při zpracování snímků o původní velikosti modulem *opencv_sgbm* se zapnutým parametrem *hh_mode*

	3	5	7	9	11
prům. PSNR	25,8054	25,7021	26,0151	26,6714	25,6320
směr. odch.	0,3486	0,3777	0,3921	0,5441	0,3538

Tabulka 4.10: Průměrné hodnoty PSNR a směrodatné odchylky při zpracování snímků o poloviční velikosti modulem *opencv_sgbm* se zapnutým parametrem *hh_mode*

4.2.2 Modul *opencv_cudabm*

Jako druhý přichází na řadu opět modul *opencv_cudabm*. Postup testování byl opět stejný a v případě tohoto modulu byly otestovány velikosti okénka 5 až

[3] soubor *sgbm07hh.mp4* nebo `https://youtu.be/qpP6laMxQv8`

[4] soubor *scaled_sgbm07hh.mp4* nebo `https://youtu.be/0OkfnkwHODI`

[5] `https://www.youtube.com/playlist?list=PLjmqw4nZO6mgYiMfVBNRVCeuSo_NK-1oO`

15. Vzhledem k tomu, že tento modul bez problému zvládal i bez zmenšování zpracovávat 120 snímků za sekundu, a to pro všechny velikosti okénka, není zde možné ukázat vliv tohoto parametru jako v předchozím případě. Místo toho byl ovšem použit nástroj NVIDIA Visual Profiler, který je součástí vývojové sady CUDA. S jeho pomocí jsme pak zjistili jaké kernely tento modul volá a jak dlouho jednotlivá volání trvají. V následujících tabulkách tedy nebudou uvedeny počty snímků za sekundu, ale průměrná doba běhu kernelu *stereoKernel*, což je součást OpenCV implementace tohoto algoritmu, která provádí hlavní část párování. Hodnoty uvedené v tabulkách jsou pak v milisekundách.

vel. ok. / p	5	7	9	11	13	15
čas běhu / ms	1,24	1,67	1,81	1,91	2,01	2,11

Tabulka 4.11: Časy trvání hlavního kernelu pro různá nastavení velikosti okénka při zpracování snímků o původní velikosti 640×480 pixelů a 64 hodnot disparit modulem *opencv_cudabm*

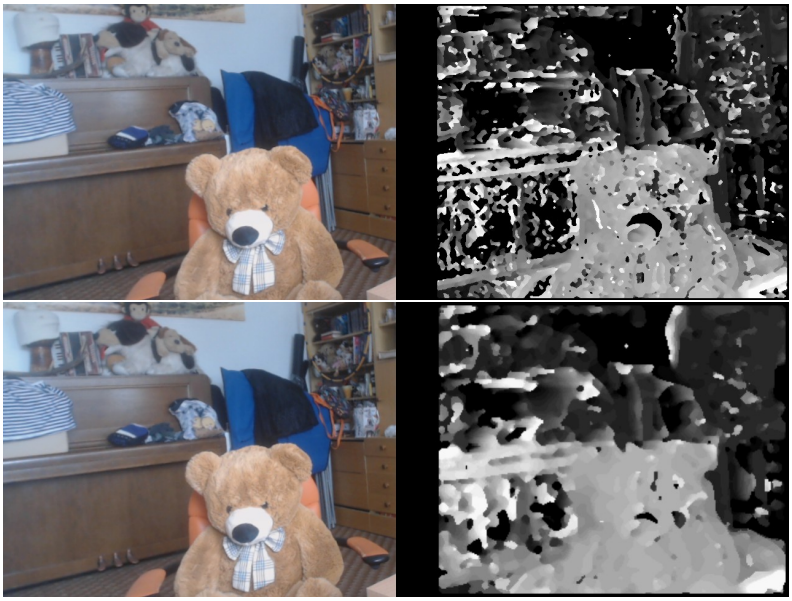
vel. ok. / p	5	7	9	11	13	15
čas běhu / ms	0,25	0,64	0,72	0,74	0,77	0,80

Tabulka 4.12: Časy trvání hlavního kernelu pro různá nastavení velikosti okénka při zpracování snímků zmenšených na velikost 320×240 pixelů a 32 hodnot disparit modulem *opencv_cudabm*

Z tabulek výše můžeme vidět, že v případě tohoto modulu má zvětšování okénka na rychlost zpracovaní opačný efekt než u modulu *opencv_sgbm*. To je očekávaný stav, neboť u tohoto block-matchingového algoritmu znamená větší okénko více hodnot k agregaci. Pro demonstraci kvality zpracování zde opět předvedeme dva snímky. Rovněž v tomto případě byla velikost okénka na hodnotu 7.

Jak vidíme ze snímků, stejně jako při testování statickými snímky, je výsledek ve srovnání s předchozím modulem poněkud horší. Je zde větší množství neplatných pixelů a rovněž se zvýšil počet oblastí, jejichž nalezená disparita zjevně neodpovídá realitě. Z hlediska stability zde opět dochází k fluktuacím v chybně spárovaných oblastech. Stejně jako v předchozím případě jsou pak tyto nestabilní oblasti o něco méně patrné ve videozáznamu[6] kde byly použity zmenšené snímky. Výstup[7] ze snímků o původní velikosti se opět jeví jako o něco méně stabilní. Oblast kolem víka piana, která byla problematická u předchozího modulu byla pak chybně spárována i zde. Co se týče rozšíření hran, které bylo popsáno u testování pomocí statického snímku, to se vy-

[6] soubor *scaled_cbm07.mp4* nebo `https://youtu.be/JwSDg0Krzwk`
[7] soubor *cbm07.mp4* nebo `https://youtu.be/8CijBn_5cMg`



Obrázek 4.20: Snímky z testovacího videozáznamu zpracované modulem *opencv_cudabm* bez a za použití zmenšených vstupních snímků

skytuje i zde a v případě zmenšených vstupních snímků je tento efekt ještě markantnější.

Na samotném konci výstupního videozáznamu se zde oproti předchozímu modulu objevuje další negativní efekt. Z nějakého důvodu zřejmě u jedné z kamer došlo ke změně parametru expozice či vyvážení bílé a změnil se tak i jas celého snímku. To negativně ovlivnilo párování pixelů a kvalita výsledné mapy se zhoršila. Výstupy pro ostatní otestovaná nastavení jsou samozřejmě opět k dispozici[8].

V tabulkách níže jsou opět uvedeny hodnoty PSNR indikující stabilitu sekvence. V tomto případě jsou hodnoty nižší než v případě předchozího modulu, což je zřejmé i z výsledků zpracování pohyblivé scény. Zvětšování okénka má i zde na stabilitu pozitivní vliv, ovšem zlepšení při použití zmenšených snímků již není tak výrazné jako v předchozím případě.

	5	7	9	11	13	15
prům. PSNR	14,9022	15,2260	15,6510	15,9856	16,2940	16,6031
směr. odch.	0,0695	0,0809	0,0915	0,1051	0,1131	0,1261

Tabulka 4.13: Průměrné hodnoty PSNR a směrodatné odchylky při zpracování snímků o původní velikosti modulem *opencv_cudabm*

[8] `https://www.youtube.com/playlist?list=PLjmqw4nZ06mhKQVReoE_kW2DgaiCdV03I`

	5	7	9	11	13	15
prům. PSNR	16,9751	17,6418	18,7825	19,5817	20,3913	20,9922
směr. odch.	0,1387	0,1425	0,1357	0,1344	0,1365	0,1199

Tabulka 4.14: Průměrné hodnoty PSNR a směrodatné odchylky při zpracování snímků o poloviční velikosti modulem *opencv_cudabm*

4.2.3 Modul *cuda_asw*

Jako další v pořadí byl stejným způsobem otestován modul *cuda_asw*. Parametrem jehož vliv na rychlost zpracování byl v tomto případě otestován, je parametr určující počet zjemňovacích iterací. Otestován byl počet iterací 1 až 6, a to opět při zpracování zmenšených snímků, tak snímků o původní velikosti. Pro ilustraci běhu jednotlivých kernelů byl opět použit nástroj NVIDIA Visual Profiler. Pomocí něho byly získány jednak o délce běhu jednotlivých kernelů a také o jejich procentuálním zastoupení v průběhu celého zpracování. Výsledky měření pro jednotlivé počty iterací jsou opět uvedeny v následujících tabulkách.

iterace	1	2	3	4	5	6
snímky za s.	13,63	12,50	11,50	10,72	10,00	9,34

Tabulka 4.15: Počet snímků zpracovaných za sekundu pro různé počty iterací při zpracování snímků o původní velikosti 640×480 pixelů a 64 hodnot disparit modulem *asw_cuda*

iterace	1	2	3	4	5	6
snímky za s.	88,28	80,58	73,72	68,40	63,63	59,39

Tabulka 4.16: Počet snímků zpracovaných za sekundu pro různé počty iterací při zpracování snímků zmenšených na velikost 320 × 240 pixelů a 32 hodnot disparit modulem *asw_cuda*

Práci modulu opět předvedeme na výsledcích testovacího záznamu. Pro pořízení následujících snímků byl parametr *iterations* nastaven na hodnotu 3.

Z výsledných snímků vidíme že kvalita zpracování medvěda v popředí je poměrně dobrá, rozhodně lepší než u předchozího modulu. Co se týče pozadí, je jiý situace o něco horší. Nachází se zde poměrně velké neplatné oblasti a ani zde nebyla správně spárována oblast piana. Na rozdíl od modulu *opencv_sgbm* tento modul hůře zvládá oblasti s jednolitou barvou. Objevuje se zde rovněž

[9]Zbývající procenta odpovídají kopírování do paměti GPU a nastavování hodnot v paměti

Kernel	*počet volání*	*doba běhu 1 volání*	*podíl v běhu*[9]
horizontalCostAggregation	128	216 μs	34,8%
verticalCostAggregation	128	206 μs	33,2%
WTA2	6	2,59 ms	19,5%
horizontalRefinement	6	682 μs	5,1%
WTA1	2	970 μs	2,4%
verticalRefinement	6	292 μs	2,2%
disparityMedianFilter	1	308 μs	0,4%
consistencyCheck	4	41 μs	0,2%
disparityFill	1	51 μs	0,1%

Tabulka 4.17: Tabulka hodnot pro jednotlivé kernely modulu *cuda_asw* při zpracování snímků o velikosti 640 × 480 s 64 hodnotami disparity a třemi zjemňovacími iteracemi

Obrázek 4.21: Snímky z testovacího videozáznamu zpracované modulem *cuda_asw* bez a za použití zmenšených vstupních snímků

zajímavý efekt, kdy správně zpracovaný objekt v popředí ovlivňuje neplatné oblasti v pozadí. To je vidět zejména na výsledku získaném ze zmenšených snímků v oblasti nalevo od medvěda, kde dochází k výraznému „rozpíjení“ okrajů medvěda a opěradla židle. Na pravé straně ovšem k tomuto efektu nedochází, protože oblast v pozadí byla díky své barevné různorodosti spárována lépe.

Pokud se zaměříme na výsledné videozáznamy[10] a jejich stabilitu, zjistíme že chybně spárované oblasti jsou poměrně nestabilní. Při použití zmenšených snímků se situace jeví jako o něco lepší, i když nestabilní oblasti se mění poměrně rychle. To je ovšem způsobeno výrazným zrychlením zdrojového záznamu (záznam byl kvůli přesnému určení výkonu modulů zrychlen čtyřnásobně).

Při zvyšování počtu iterací pak stejně jako v případě statických snímků nedochází k téměř žádnému pozorovatelnému efektu. Můžeme tedy říci, že v zájmu dosažení většího výkonu pro zpracování bohatě dostačují 1 či 2 zjemňovací iterace. Výsledné záznamy jsou pro porovnání opět k dispozici na přiloženém DVD a serveru YouTube[11].

Naměřené hodnoty PSNR v následujících tabulkách (4.18 a 4.19) opět potvrzují subjektivní pozorování z videozáznamu s pohyblivými objekty. Zvyšování počtu iterací má na stabilitu jen velmi malý vliv a použitím zmenšených snímků dojde k jejímu mírnému zlepšení. V porovnání s ostatními moduly se tento modul pak podle naměřených hodnot jeví jako nejhorší. Subjektivně ovšem jeho výsledky působí lépe než předchozí modul *opencv_cudabm*, zejména proto, že výsledky obsahují menší množství pixelů, které byly sice vyhodnoceny jako platné ale svou hodnotou neodpovídají skutečnému stavu.

	1	2	3	4	5	6
prům. PSNR	14,5430	14,7189	14,6450	14,4946	14,4663	14,3908
směr. odch.	0,1001	0,0939	0,1072	0,1073	0,1341	0,1071

Tabulka 4.18: Průměrné hodnoty PSNR a směrodatné odchylky při zpracování snímků o původní velikosti modulem *cuda_asw*

	1	2	3	4	5	6
prům. PSNR	18,5263	19,0134	18,8071	18,8240	18,9916	18,9187
směr. odch.	0,1272	0,1343	0,1463	0,1403	0,1903	0,1827

Tabulka 4.19: Průměrné hodnoty PSNR a směrodatné odchylky při zpracování snímků o poloviční velikosti modulem *cuda_asw*

4.2.4 Modul *cuda_sncc*

Tento modul nebyl pro své slabé výsledky s videozáznamy vůbec testován. Výsledky totiž byly subjektivně ještě mnohem horší než při testování statickými

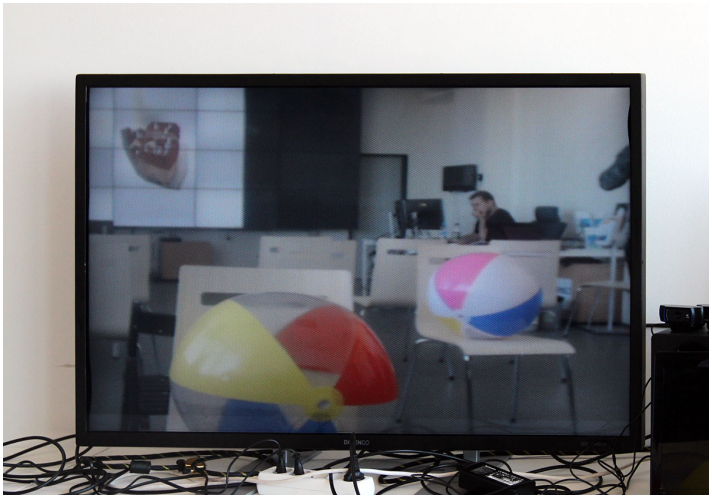
[10] soubory *asw03.mp4* a *scaled_asw03.mp4* nebo `https://youtu.be/w51NNIraSnQ` respektive `https://youtu.be/JTlHdSBIeAM`

[11] `https://www.youtube.com/playlist?list=PLjmqw4nZO6mhOACgYEKiGNlGExTjakMVC`

snímky. Z hlediska stability také nebylo možné modul otestovat, neboť naprostou většinu snímku zabíraly černé neplatné pixely, což by bylo při měření vyhodnoceno jako velmi stabilní snímek.

4.3 Testování výstupu na 3D monitor

Jako poslední byl otestován modul *dimenco_out*, jehož implementace byla popsána v části 3.2.7 a který je určen pro výstup na autostereo monitor značky Dimenco. Ve chvíli kdy byly pixely v prvním řádku snímku nastaveny na patřičné hodnoty, bylo vytvořeno prokládání tak, jak bylo popsáno a výstup byl zobrazen v SDL okně ve full-screen režimu, se monitor okamžitě přepnul do 3D režimu. V tu chvíli byla na obrazovce jasně patrná hloubka, která odpovídala hloubkové mapě vytvořené jedním z modulů.



Obrázek 4.22: Fotografie z testování 3D monitoru

Obraz byl získán ze stejných kamer, které byly použity pro testování zpracování video vstupu a které byly v této práci popsaným způsobem zkalibrovány. Vyzkoušeny pak byly tři moduly, konkrétně *opencv_sgbm*, *opencv_cudabm* a *cuda_asw*. Dojem z 3D efektu vytvořeného monitorem je ovšem čistě subjektivní a není dobře možné ho popsat. Bylo však jasně zřejmé, které objekty ve snímané scéně se nachází v pozadí a které v popředí.

Při použití zmenšených snímků se pak celkový dojem nijak nezhoršil, i když se zmenšením snížilo hloubkové rozlišení. Projevil se naopak pozitivní vliv zmenšení snímků na stabilitu hloubkové mapy v čase. Poměrně rušivě však působily oblasti, u nichž byla určena malá hloubka, i když se evidentně nacházely v pozadí scény. Tyto oblasti se pak opět objevovaly zejména při použití modulu *opencv_cudabm*.

Závěr

Cílem této práce bylo nalezení způsobu pro vytvoření hloubkové mapy ze stereoskopického obrazu. Byly popsány základní principy tvorby hloubkové mapy a rovněž byly popsány čtyři algoritmy k tomuto úkolu určené. Ty pak byly implementovány a úspěšně otestovány, a to i na speciálním zařízení v laboratoři SAGElab. Implementovány byly rovněž dva moduly, které jsou pro správnou činnost algoritmů pro tvorbu mapy nutné. Jedná se konkrétně o moduly provádějící kalibraci dvojice kamer a následnou rektifikaci získávaných snímků.

Moduly byly otestovány a byla tak porovnána jejich výkonnost, přesnost a stabilita získávané mapy v čase. Jako nejpřesnější byl při testování statickými snímky vyhodnocen modul *cuda_asw* implementující algoritmus pro iterativní zjemňování s adaptivními podpůrnými vahami (popsán v části 2.5). Z hlediska výkonu jasně zvítězil modul *opencv_cudabm*, který využívá OpenCV implementaci block-matchingového algoritmu (2.2) s metrikou součtu absolutních hodnot rozdílů. V kritériu stability se jako nejlepší projevil modul *opencv_sgbm* obsahující OpenCV implementaci algoritmu Semi-Global Block Matching (2.3).

Nepříliš úspěšná pak byla implementace modulu *cuda_sncc*, který využívá techniky popsané v části 2.4. Tento algoritmus podával v porovnání s ostatními velmi špatné výsledky i když v částech snímku, kde se správné párování podařilo, byla kvalita velmi dobrá.

Do budoucna je pak již implementované moduly doplňovat o další využívající jiné druhy algoritmů. V této práci byly totiž popsány a implementovány zejména takzvané lokální metody tvorby hloubkové mapy a srovnání s globálními nebo jinými druhy metod by jistě bylo zajímavé.

Literatura

[1] Bradski, G.; Kaehler, A.: *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008, ISBN 9780596554040. Dostupné z: https://books.google.cz/books?id=seAgiOfu2EIC

[2] Kowalczuk, J.; Psota, E.; Perez, L.: Real-Time Stereo Matching on CUDA Using an Iterative Refinement Method for Adaptive Support-Weight Correspondences. *Circuits and Systems for Video Technology, IEEE Transactions on*, ročník 23, č. 1, Jan 2013: s. 94–104, ISSN 1051-8215, doi:10.1109/TCSVT.2012.2203200.

[3] IDS Imaging Development Systems GmbH: Obtaining Depth Information from Stereo Images [online]. 2012, [cit: 2016-02-25]. Dostupné z: http://www.sanxo.eu/content/whitepaper/IDS_Whitepaper_3D_Stereo_Vision.pdf

[4] OpenCV: Camera Calibration [online]. 2016, [cit: 2016-03-15]. Dostupné z: http://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html

[5] Ahmed, M.; Farag, A.: Nonmetric calibration of camera lens distortion: differential methods and robust estimation. *Image Processing, IEEE Transactions on*, ročník 14, č. 8, Aug 2005: s. 1215–1230, ISSN 1057-7149, doi:10.1109/TIP.2005.846025.

[6] Scharstein, D.: Middlebury Stereo Evaluation - Version 3 [online]. Srpen 2015, [cit: 2016-04-03]. Dostupné z: http://vision.middlebury.edu/stereo

[7] Scharstein, D.; Szeliski, R.: A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, ročník 47, č. 1: s. 7–42, ISSN 1573-1405, doi:10.1023/A:1014573219977. Dostupné z: http://dx.doi.org/10.1023/A:1014573219977

[8] Hirschmuller, H.: Stereo Processing by Semiglobal Matching and Mutual Information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, ročník 30, č. 2, Feb 2008: s. 328–341, ISSN 0162-8828, doi: 10.1109/TPAMI.2007.1166.

[9] Einecke, N.; Eggert, J.: A Two-Stage Correlation Method for Stereoscopic Depth Estimation. In *Digital Image Computing: Techniques and Applications (DICTA), 2010 International Conference on*, Dec 2010, s. 227–234, doi:10.1109/DICTA.2010.49.

[10] yuri [projects.iim.cz] [online]. Listopad 2013, [cit: 2016-03-20]. Dostupné z: `https://projects.iim.cz/yuri`

[11] NVIDIA Corporation: Programming Guide :: CUDA Toolkit Documentation [online]. Září 2015, [cit: 2016-03-17]. Dostupné z: `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`

[12] OpenCV: cv::StereoSGBM Class Reference [online]. 2016, [cit: 2016-04-03]. Dostupné z: `http://docs.opencv.org/3.1.0/d2/d85/classcv_1_1StereoSGBM.html`

[13] Scharstein, D.: 2003 Stereo datasets with ground truth [online]. Leden 2015, [cit: 2016-04-03]. Dostupné z: `http://vision.middlebury.edu/stereo/data/scenes2003/`

[14] ImageMagick: Command-line Tools: Compare [online]. Duben 2016, [cit: 2016-04-28]. Dostupné z: `http://www.imagemagick.org/script/compare.php`

[15] Huynh-Thu, Q.; Ghanbari, M.: Scope of validity of PSNR in image/video quality assessment. *Electronics Letters*, ročník 44, č. 13, June 2008: s. 800–801, ISSN 0013-5194, doi:10.1049/el:20080522.

Příloha **A**

Seznam použitých zkratek

ASW Adaptive Support Weights

CUDA Compute Unified Device Architecture

DSI Disparity space image

NCC Normalized cross-corelation

SNCC Summed normalized cross-corelation

PSNR Peak signal to noise ratio

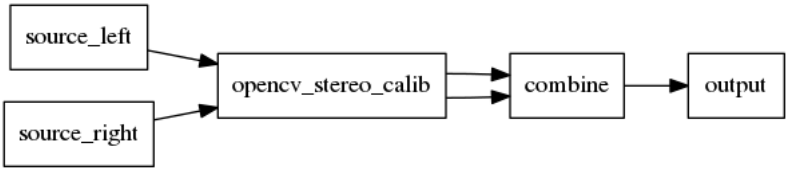
Příloha **B**

Návod pro provedení kalibrace

Pro provedení kalibrace soustavy kamer potřebujeme následující moduly:

- 2 instance modulu pro video vstup, například modul *v4l2source*
- 1 instance modulu *opencv_stereo_calib*
- (dobrovolně) 1 instance modulu *combine*, pro spojení výstupů předchozího modulu
- Modul pro zobrazení výstupu, není bezpodmínečně nutný, ovšem zobrazení živého výstupu kalibraci výrazně usnadňuje. Lze použít například modul *sdl2window*.

Jednotlivé moduly pak pomocí konfiguračního souboru zapojíme následujícím způsobem.



Obrázek B.1: Zapojení modulů pro kalibraci

Dále potřebujeme standardní černobílou šachovnici. Ta by měla být rovná a dostatečně pevná, aby u ní nedocházelo k deformacím, které by kalibraci mohly negativně ovlivnit. Podle použité šachovnice je pak nutné nastavit parametry *chessboard_x* a *chessboard_y* modulu *opencv_stereo_calib*. Tyto parametry nastavíme na hodnoty odpovídající počtu vnitřních průsečíků naší šachovnice. Pokud tedy použijeme běžnou šachovnici s 64 políčky, parametry nastavíme na hodnoty 7 a 7. V případě že použitá šachovnice není čtvercová,

můžeme její osy x a y zvolit libovolně, je ovšem nezbytné aby samotná políčka čtvercová byla.

Ve chvíli kdy máme připravenu šachovnici a konfigurační soubor s moduly můžeme s tímto konfiguračním souborem libyuri spustit. Nyní nastává klíčová část celého procesu, kdy pomocí šachovnice provedeme celou kalibraci. Při této činnosti je důležité, aby se šachovnice v několika různých polohách objevila pokud možno ve všech částech záběru (nikoli například pouze v okolí středu), a to v různých vzdálenostech od kamer. Díky živému výstupu (pokud jsme se ho rozhodli použít) pak máme dobrou zpětnou vazbu o umístění kalibrační šachovnice v rámci scény.

Ve chvíli kdy máme nasbírán dostatečný počet snímků s šachovnicí v různých polohách, proběhne kalibrace a jsou uloženy transformační matice. Kvalitu provedené kalibrace si pak můžeme otestovat například podobným zapojením jaké bylo předvedeno na obrázku 4.2, ze kterého vyřadíme část, která snímky ukládá na disk. Pokud provedená kalibrace nedopadla dobře a jako výsledek vidíme pouze velmi malý výřez obrazu, nebo je obraz viditelně deformován, musíme kalibraci provést a následně otestovat znovu.

Pokud je kalibrace úspěšná, můžeme z ní získané transformační matice použít k rektifikaci snímků a následný výpočet mapy disparit. Existuje rovněž možnost vytvořit si pomocí kalibračního modulu transformačních matic více. Pomocí parametru *path* můžeme určit kam se výsledné soubory uloží a my si můžeme připravit transformační matice pro více rozlišení snímků, neboť tyto transformační matice nejsou univerzální. Správnou transformační matici pro modul *opencv_rectify* pak můžeme určit pomocí parametru *map_file*, který modulu udává cestu přímo k souboru s maticemi.

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Lhotan_Miroslav_2016.pdf` ..... text práce ve formátu PDF
- `videos` ..... výstupy z testování zpracování videozáznamu
  - `asw` ..... výstupy zpracované modulem *cuda_asw*
  - `cudabm` ..... výstupy zpracované modulem *opencv_cudabm*
  - `sgbm` ..... výstupy zpracované modulem *opencv_sgbm*