VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

DETEKCE OBJEKTŮ
OBJECT DETECTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Filip Baáš**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Miloslav Richter, Ph.D.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: Filip Baáš ***ID:*** 173555
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Detekce objektů

POKYNY PRO VYPRACOVÁNÍ:

Vytvořte prostředí pro detekci kontrastních objektů ve snímcích a pro stanovení jejich polohy a odlišnosti od vzoru. Úkolem je na kontrastním obraze určit pohybující se objekt a porovnat ho se vzorem a povolenými stavy.
1) Proveďte rozbor úlohy a vytvořte sady snímků (videí) pro testování algoritmů.
2) Navrhněte algoritmus pro vyhledání objektů a stanovení jejich stavů (poloha, orientace). Navrhněte postup pro zjištění odchylky aktuálně snímaných objektů od předdefinovaných stavů a pro stanovení odchylky od vzoru. Navrhněte vhodný způsob vizualizace odchylek podle zadaných parametrů nebo vzorových stavů.
3) Realizujte a otestujte navržené algoritmy. Zhodnoťte jejich kvalitu a přesnost.

DOPORUČENÁ LITERATURA:

Žára J., Beneš B., Sochor J., Felkel P.: Moderní počítačová grafika, Computer Press, 1998, ISBN 80-251-0454-0

Hlaváč V., Šonka M.: Počítačové vidění,Grada, Praha 1992, ISBN 80-85424-67-3

Termín zadání: 6.2.2017 ***Termín odevzdání:*** 29.5.2017

Vedoucí práce: Ing. Miloslav Richter, Ph.D.
Konzultant:

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

ABSTRAKT

Táto bakalárska práca sa zaoberá detekciou tvarovo nemenných objektov na snímkach. Pre detekciu je využitý algoritmus hranového vzdialenostného párovania, ktorý je na tieto účely stavaný. Prvá časť tejto práce je určená teoretickému vysvetleniu princípu tohto algoritmu. Sú tu vysvetlené najpoužívanejšie spôsoby prevedenia vzdialenostnej transformácie, potrebnej pre tento algoritmus. Ďalej je tu vysvetlený výpočet hranovej vzdialenosti a pyramídová reprezentácia informácií. Druhá časť je venovaná vývojovým nástrojom použitým v tejto práci, ktorými sú vývojové prostredie Visual Studio a knižnice OpenCV pre spracovanie obrazu a Qt pre tvorbu grafického používateľského rozhrania. V poslednej časti tejto práce je popísaná praktická realizácia detekcie objektov. Je tu popísaný spôsob akým sú objekty renderované, postup vytvorenia vzoru z renderovaného obrazu, spôsob vytvorenia sady vzorov, porovnanie rýchlostí vzdialenostných transformácií počítaných v rôznych metrikách, porovnanie obyčajnej a pyramídovej detekcie a spôsob vyhodnotenia detekcie. V závere práce sú zhrnuté dosiahnuté výsledky práce.

KĽÚČOVÉ SLOVÁ

spracovanie obrazu, počítačové videnie, detekcia objektov, vzdialenostná transformácia, vzor, hranové párovanie, pyramídová detekcia, renderovanie modelov, grafické používateľské rozhranie, OpenCV, Qt, Visual Studio, C++

ABSTRACT

This bachelor thesis deals with detection of rigid objects in images. Chamfer matching algorithm, which is built for this kind of tasks is used as detection algorithm. First part of this work is dedicated to theoretical explanation of the algorithm. Most commonly used metrics of distance transform are explained, which is needed for the algorithm. Also explanation of chamfer distance calculation and pyramid representation of information is here. Next part is dedicated to development tools used in this work, which is integrated development environment Visual Studio and libraries OpenCV for image processing and Qt for graphical user interface creation. In last part of this work, practical implementation of object detection is described. This part explains the way objects are rendered, steps for creating a template from rendered image, method to create set of templates, comparison of speed of distance transformation calculation in different metrics, comparison of speed of common and pyramid detection and method of score calculation. The conclusion summarizes reached goals of this work.

KEYWORDS

image processing, computer vision, object detection, distance transform, template, chamfer matching, pyramid detection, model rendering, graphical user interface, OpenCV, Qt, Visual Studio, C++

BAÁŠ, Filip. *Detekce objektů*. Brno, 2017, 52 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedúci práce: Ing. Miloslav Richter, Ph.D.

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „Detekce objektů“ vypracoval(a) samostatne pod vedením vedúceho bakalárskej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor(ka) uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil(a) autorské práva tretích osôb, najmä som nezasiahol(-la) nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý(-á) následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

POĎAKOVANIE

Týmto spôsobom by som sa chcel poďakoval vedúcemu tejto bakalárskej práce, pánovi Ing. Miloslavovi Richterovi, Ph.D. za odborné vedenie, pohotovú komunikáciu, rady pri konzultáciách, ochotu poradiť a podnetné návrhy k práci.

Brno ....

podpis autora(-ky)

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM VÝPISOV

ÚVOD

Ľudia si už odpradávna snažili uľahčiť svoju prácu, a preto vymýšľali nástroje, ktoré im dopomáhali k tomu, aby za kratší čas vykonali viac práce a vydali pri tom menej energie. Medzi tieto nástroje môžeme zaradiť oštep určený na lov divej zveri, bicykel na rýchlejšiu prepravu, či elektrickú energiu, ktorá nám v súčasnej dobe pomáha vo viacerých smeroch.

Jedným z takýchto nástrojov na uľahčenie práce, ktorý neustále mení náš pohľad na svet, je aj výpočtová technika a samotné počítače. Počítač je diskrétny systém, ktorý nám dennodenne pomáha v najrôznejších situáciách. Povie nám čo sa deje vo svete, nájde nám odpovede na naše otázky a sprostredkuje nám spojenie s inými ľuďmi na diaľku.

Najdôležitejším ľudským zmyslom je zrak, ktorým človek prijíma až 80% informácií zo svojho okolia. Čo sa však stane, ak pridáme tento zmysel práve počítaču? Touto otázkou sa zaoberá odvetvie výpočtovej techniky s názvom počítačové videnie. [8]

Pod pojmom počítačové videnie si je možné predstaviť odvetvie zaoberajúce sa učením počítača získavať informácie zo zachyteného obrazu. Tento obraz môže byť uložených v pamäti počítača, alebo získaný z pripojenej kamery. Pod získaním informácií je myslené správne rozoznanie skupiny javov či objektov a nasledovné využitie týchto informácií pre ďalšie spracovanie. Počítačové videnie má vysoké uplatnenie v medicíne, výrobnom priemysle, vojenskom priemysle, astronómií alebo dopravnom priemysle.[24]

Počítačovým videním sa zaoberá aj táto práca venovaná detekcií objektov na snímke.

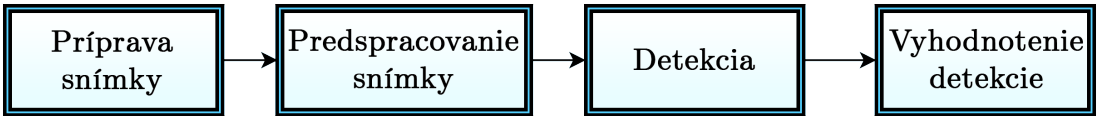
Objekt: Pod pojmom objekt sú v tejto práci myslené dva typy objektov. Prvým typom je plošný objekt, ktorý si je možné predstaviť ako obraz na stene, fotografiu, značkovací kód a podobne. Takýto objekt je reprezentovaný ako 2D obraz, ideálne s vysokým kontrastom.

Druhým typom objektu v tejto práci je zostrojiteľné, pevné a tvarovo nemenné priestorové teleso tvoriace jeden celok s viditeľnými hranami. Objekt je jednej farby, prípadne s jemným farebným prechodom. Medzi takéto objekty patria okrem iného aj telesá, ktoré sú symetrické podľa jednej alebo viacerých svojich osí (kocka), alebo rotačné telesá (kužeľ). Detekcia takýchto telies však často vedie ku nejednoznačnému, singulárnemu riešeniu.

Dôvodom výberu týchto typov objektov je fakt, že tvarovo nemenné objekty je ľahšie detekovať, vzhľadom na to že nie je potrebné brať v úvahu aj možné zmeny ich tvaru.

Z toho dôvodu sa za objekty v tejto práci nepovažujú pevné telesá s pohyblivými časťami a ani telesá, ktoré akýmkoľvek spôsobom menia svoj tvar.

Detekcia: Pod pojmom detekcia je myslené správne rozoznanie objektu a určenie jeho polohy a orientácie za použitia vhodných algoritmov. Detekciu si je možné predstaviť ako reťazec zobrazený na nasledujúcom obrázku:



V tomto reťazci je vytvorená alebo načítaná snímka filtrami predspracovaná a následne je na nej vykonaná detekcia. Po dokončení detekcie je detekcia vyhodnotená napr. percentuálnou zhodou.[4]

Metód detekcie objektov existuje niekoľko. Metóda detekcie na základe výrazných znakov objektu, detekcia na základe tvaru, detekcia na základe farebnej informácie, detekcia na základe porovnávania so vzorom či detekcia pohybu. Nakoľko sa táto práca venuje objektom ktoré sú tvarovo nemenné, ako detekčnou metódou pre túto prácu bola zvolená metóda detekcie na základe porovnávania so vzorom.[7]

Ako detekčný algoritmus je v tejto práci použitý hranový párovací algoritmus (Chamfer Matching) v spojení so vzdialenostnou transformáciou (Distance Transform). Ich vysvetleniu je venovaná kapitola 1. V tejto kapitole je ďalej vysvetlená pyramídová detekcia, ktorá zabezpečuje zníženie času detekcie objektov. Praktickej realizácii detekcie a vyhodnoteniu detekcie sú venované podkapitoly 3.2 a 3.3.

Grafické používateľské rozhranie: Pre uľahčenie a urýchlenie práce s vytváraním vzoru a detekčným algoritmom bolo vytvorené GUI, ktorého popisu je venovaná podkapitola 3.4.

Vývojové nástroje použité pri tvorbe tejto práce, ich popis a dôvody výberu je možné nájsť v kapitole 2

1 TEÓRIA DETEKCIE OBJEKTOV

Prvá kapitola tejto práce je venovaná teórii detekcie objektov. Ako už bolo v úvode spomenuté, táto práca sa zaoberá iba detekciou pevných, tvarovo nemenných telies a preto využíva algoritmus hranového vzdialenostného párovania. Ten je vysvetlený v podkapitole 1.1.

Podkapitola 1.2 je venovaná pyramídovej reprezentácii dát. Je tu vysvetlený princíp a dôvody použitia tejto reprezentácie.

1.1 Hranové vzdialenostné párovanie

Hlavnou úlohou tejto kapitoly je ozrejmiť princíp algoritmu pre hranové vzdialenostné párovanie, na ktorom je založená detekcia objektov v tejto práci.

Tento algoritmus sa skladá z dvoch krokov. Prvým krokom je aplikovanie vzdialenostnej transformácie na hranový (binárny) obraz scény, na ktorej chceme detekovať objekt. Touto transformáciou vzniká vzdialenostná mapa. Algoritmy tejto transformácie a ich kvalita je popísaná v odseku 1.1.1.

Druhým krokom je samotné hranové párovanie vzoru so vzdialenostnou mapou. Vzor si je možné predstaviť ako množinu kľúčových bodov opisujúcich hrany vzorového objektu.[16] Ako výstup získavame hodnotu hranovej vzdialenosti vzoru. Tento proces je popísaný v odseku 1.1.2.

1.1.1 Vzdialenostná transformácia

Všetky obrazy v počítačovom videní sú reprezentované ako matice zložené z elementov nazývaných obrázkové body, alebo pixely. Každý pixel môže niesť jednu, alebo viac informácií o odtieni farby v závislosti od typu obrazu. V čiernobielom obraze nesie pixel iba jednu informáciu o úrovni šedej. Vo farebnom obraze pixel nesie tri informácie o odtieni každej z RGB zložiek. Spôsoby indexovania pixelov sú znázornené na obrázku 1.1.

$\mathbf{P}_{(0,0)}$	$\mathbf{P}_{(1,0)}$	$\mathbf{P}_{(2,0)}$	$\mathbf{P}_{(3,0)}$	$\mathbf{P}_{(4,0)}$
$\mathbf{P}_{(0,1)}$	$\mathbf{P}_{(1,1)}$	$\mathbf{P}_{(2,1)}$	$\mathbf{P}_{(3,1)}$	$\mathbf{P}_{(4,1)}$
$\mathbf{P}_{(0,2)}$	$\mathbf{P}_{(1,2)}$	$\mathbf{P}_{(2,2)}$	$\mathbf{P}_{(3,2)}$	$\mathbf{P}_{(4,2)}$
$\mathbf{P}_{(0,3)}$	$\mathbf{P}_{(1,3)}$	$\mathbf{P}_{(2,3)}$	$\mathbf{P}_{(3,3)}$	$\mathbf{P}_{(4,3)}$
$\mathbf{P}_{(0,4)}$	$\mathbf{P}_{(1,4)}$	$\mathbf{P}_{(2,4)}$	$\mathbf{P}_{(3,4)}$	$\mathbf{P}_{(4,4)}$

(a) Štandardné indexovanie

$\mathbf{P}_{(0)}$	$\mathbf{P}_{(1)}$	$\mathbf{P}_{(2)}$	$\mathbf{P}_{(3)}$	$\mathbf{P}_{(4)}$
$\mathbf{P}_{(5)}$	$\mathbf{P}_{(6)}$	$\mathbf{P}_{(7)}$	$\mathbf{P}_{(8)}$	$\mathbf{P}_{(9)}$
$\mathbf{P}_{(10)}$	$\mathbf{P}_{(11)}$	$\mathbf{P}_{(12)}$	$\mathbf{P}_{(13)}$	$\mathbf{P}_{(14)}$
$\mathbf{P}_{(15)}$	$\mathbf{P}_{(16)}$	$\mathbf{P}_{(17)}$	$\mathbf{P}_{(18)}$	$\mathbf{P}_{(19)}$
$\mathbf{P}_{(20)}$	$\mathbf{P}_{(21)}$	$\mathbf{P}_{(22)}$	$\mathbf{P}_{(23)}$	$\mathbf{P}_{(24)}$

(b) Lineárne indexovanie

Obr. 1.1: Spôsoby indexovania pixelov v obraze.

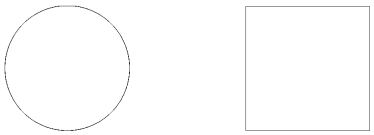
Pristupovať ku pixelu je možné pomocou štandardného indexovania a lineárneho indexovania.[11] Ak pixel $P_{1\,(\mathrm{i,j})}$ štandardne indexuje pixel P a pixel $P_{2\,(\mathrm{k})}$ indexuje ten istý pixel lineárne, potom prepočet medzi týmito indexmi, kde $max(i)$ je šírka obrazu, je nasledovný:

$$k \ = j \cdot max(i) + i \tag{1.1}$$

$$i \ = \left\lfloor \frac{k}{max(i)} \right\rfloor \qquad j = k \mod max(i) \tag{1.2}$$

Vzdialenostná transformácia (vzdialenostná funkcia) je transformácia, ktorá binárny obraz mení na vzdialenostnú mapu (transformácia do úrovní šedej). Hodnota každého nenulového pixelu je nahradená hodnotou vzdialenosti ku najbližšiemu nulového pixelu pôvodného obrazu.[3, 5]

Vypočítaná vzdialenosť medzi jednotlivými pixelmi je závislá od metriky, ktorá je pri výpočte použitá. Najznámejšie metriky ktoré sa v spracovaní obrazu používajú sú Euklidovská metrika, metrika mestských blokov, šachovnicová metrika a chamfer metrika.[9, 2]



(a)

1	1	1	1	1
1	1	1	1	1
0	1	1	1	1
1	1	1	1	1
1	1	1	1	1

(b)

Obr. 1.2: Vzorový binárny hranový obraz.

Vzdialenostné transformácie budú aplikované na vzorový obraz 1.2. Získané vzdialenostné mapy boli získané na školskej verzii programového prostredia Matlab 2015. Príkazy pre získanie vzdialenostných máp sú vo výpise 1.1.

Výpis 1.1: Príklad vytvorenia vzdialenostných obrazov v prostredí Matlab.[9]

```
%% Priklad vytvorenia vzdialenostných obrazov

% načítanie vzorového obrazu a konverzia
% do binárnej podoby
binaryImage = im2bw(imread('vzor.png'),0.5);

%vytvorenie jednotlivých vzdialenostných obrazov
Euclidean = uint8(bwdist(~binaryImage,'euclidean'));
CityBlock = uint8(bwdist(~binaryImage,'cityblock'));
Chessboard = uint8(bwdist(~binaryImage,'chessboard'));
Chamfer = uint8(bwdist(~binaryImage,'quasi-euclidean'));
```

Euklidovská metrika (Euclidean): Táto metrika je základom pre všetky ostatné metriky. Presne počíta vzdialenosť medzi dvoma pixelmi $P_{(\mathrm{i,j})}$ a $P_{(\mathrm{k,l})}$ za použitia vzťahu [10]:

$$D_E = \sqrt{(i-k)^2 + (j-l)^2} \tag{1.3}$$

Je však výpočtovo veľmi náročná, lebo pri výpočte používa umocňovanie a odmocňovanie. Takisto je pri tejto metrike potrebné počítať vzdialenosť pre všetky kombinácie pixelov na snímke. Na druhej strane metriky ako metrika mestských blokov, šachovnicová a chamfer metrika pracujú iba s okolitými pixelmi. Preto sú tieto metriky menej výpočtovo náročné a viac využívané.

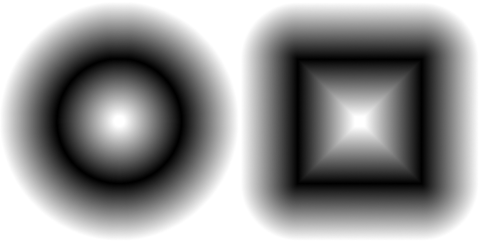
Nasledujúce metriky používajú pre vytvorenie vzdialenostnej mapy masku. Masky sa delia podľa okolia s ktorým pracujú na masky so 4-susednosťou a 8-susednosťou znázornené na obrázku1.4.

Zvolená maska prechádza binárny obraz dvakrát. Smer prvého prechodu je zľava doprava a zhora nadol. Druhý prechod sa vykonáva sprava doľava a zdola nahor. Spôsob prechodu masky s 8-susednosťou je zobrazený na obrázku 1.5.

Pri prechode sa zisťuje hodnota ktorá bude priradená pixelu $P_{(\mathrm{i,j})}$, ktorý je na maskách znázornený čiernou farbou.

Keďže pri jednotlivých prechodoch nie je pre získanie správnej vzdialenostnej mapy potrebná celá maska, v tejto práci sú uvedené pre každú metriku dve masky.

Prvá maska je použitá pri prvom prechode binárnym obrazom a druhá pri druhom. Použitím dvoch masiek sa zaručí maximálny výkon vzdialenostnej transformácie.[4]



(a)

2	2,2	2,8	3,6	4,5
1	1,4	2,2	3,2	4,1
0	1	2	3	4
1	1,4	2,2	3,2	4,1
2	2,2	2,8	3,6	4,5

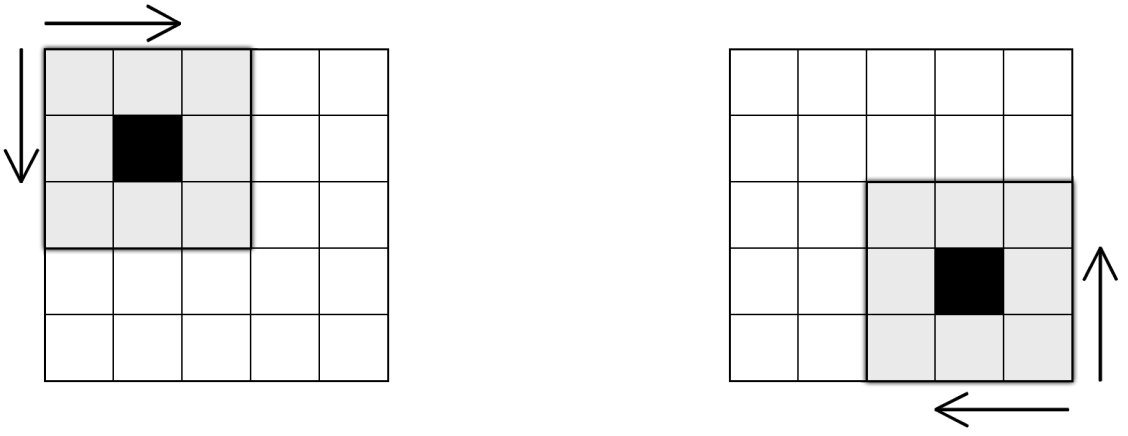
(b)

Obr. 1.3: Vzdialenostná transformácia v Euklidovskej metrike aplikovaná na vzorový obraz 1.2.

(a) 4-susedná maska

(b) 8-susedná maska

Obr. 1.4: Ukážka tvaru masiek podľa druhu susednosti.



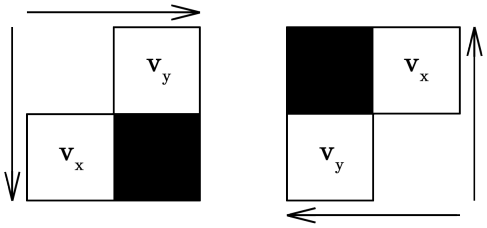
(a) Prvý prechod binárnym obrazom

(b) Druhý prechod binárnym obrazom

Obr. 1.5: Ukážka prechodu 8-susednej masky binárnym obrazom.

Metrika mestských blokov (City block, Manhattan, Taxicab geometry): Táto metrika je najjednoduchšou zo všetkých metrík v tejto práci. Vzdialenosť dvoch pixelov $P_{(\mathrm{i,j})}$ a $P_{(\mathrm{k,l})}$ je v tejto metrike daná vzťahom [28]:

$$D_{City\,block} = |i-k| + |j-l| \tag{1.4}$$



Obr. 1.6: Masky a spôsob prechodu pre metriku mestských blokov.

Táto metrika používa masky so 4-susednosťou znázornené na obrázku 1.6. Hodnoty v_{x} a v_{y} sú váhové hodnoty, ktoré sú vo väčšine prípadov nastavené na hodnotu 1. Pri prvom prechode sa hodnota pixelu $P_{(\mathrm{i,j})}$ určí nasledovne:

$$P_{(i,j)} = min(P_{(i,j)}, P_{(i,j-1)} + v_y, P_{(i-1,j)} + v_x) \tag{1.5}$$

Pri druhom prechode sa hodnota pixelu $P_{(\mathrm{i,j})}$ určí nasledovne:

$$P_{(i,j)} = min(P_{(i,j)}, P_{(i,j+1)} + v_y, P_{(i+1,j)} + v_x) \tag{1.6}$$

(a)

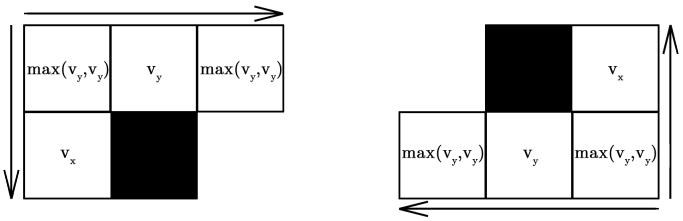
2	3	4	5	6
1	2	3	4	5
0	1	2	3	4
1	2	3	4	5
2	3	4	5	6

(b)

Obr. 1.7: Vzdialenostná transformácia v metrike mestských blokov aplikovaná na vzorový obraz 1.2.

Šachovnicová metrika (Chessboard, Chebyshev distance): Túto metriku si je možné predstaviť ako najmenší počet krokov, ktoré musí kráľ na šachovej hracej ploche vykonať aby sa dostal z jednej pozície na druhú. Vzdialenosť dvoch pixelov $P_{(\mathrm{i,j})}$ a $P_{(\mathrm{k,l})}$ sa vypočíta vzťahom [27]:

$$D_{Chessboard} = max(|i-k|, |j-l|) \tag{1.7}$$



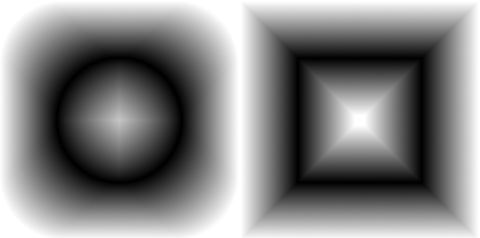
Obr. 1.8: Masky a spôsob prechodu pre šachovnicovú metriku.

Táto metrika používa masku s 8-susednosťou, čo zvyšuje jej výpočtovú náročnosť. Masky pre výpočet vzdialenostnej transformácie pomocou šachovnicovej metriky sú zobrazené na obrázku 1.8. Hodnota pixelu $P_{(\mathrm{i,j})}$ sa pri prvom prechode vypočíta následovne:

$$P_{(i,j)} = min(P_{(i,j)}, P_{(i,j-1)}+v_y, P_{(i-1,j)}+v_x, P_{(i-1,j-1)}+max(v_x,v_y), P_{(i+1,j-1)}+max(v_x,v_y)) \tag{1.8}$$

Hodnota pixelu $P_{(\mathrm{i,j})}$ sa pri druhom prechode vypočíta následovne:

$$P_{(i,j)} = min(P_{(i,j)}, P_{(i,j+1)}+v_y, P_{(i+1,j)}+v_x, P_{(i+1,j+1)}+max(v_x,v_y), P_{(i-1,j+1)}+max(v_x,v_y)) \tag{1.9}$$



(a)

2	**2**	**2**	**3**	**4**
1	**1**	**2**	**3**	**4**
0	**1**	**2**	**3**	**4**
1	**1**	**2**	**3**	**4**
2	**2**	**2**	**3**	**4**

(b)

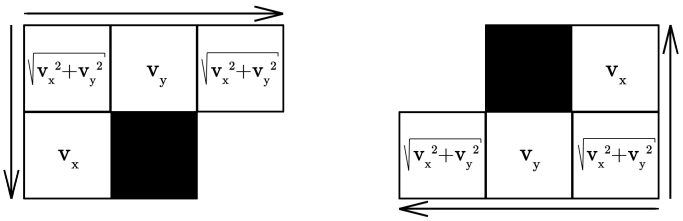
Obr. 1.9: Vzdialenostná transformácia v šachovnicovej metrike aplikovaná na vzorový obraz 1.2.

Chamfer metrika (Quasi-Euclidean): Táto metrika má zo všetkých spomenutých metrík výsledky najpodobnejšie Euklidovskej metrike, preto je taktiež označovaná ako kvázi-Euklidovská metrika. Využíva masku s rovnakým tvarom, ako šachovnicová metrika a tým pádom je aj rovnako výpočtovo náročná.

Vzdialenosť dvoch pixelov $P_{(\mathrm{i,j})}$ a $P_{(\mathrm{k,l})}$ sa vypočíta vzťahom [10]:

$$Ak\,|i-k| > |j-l| \qquad D_{Chamfer} = |i-k| + (\sqrt{2}-1)|j-l| \tag{1.10}$$

$$Ak\,|i-k| < |j-l| \qquad D_{Chamfer} = |j-l| + (\sqrt{2}-1)|i-k| \tag{1.11}$$



Obr. 1.10: Masky a spôsob prechodu pre Chamfer metriku.

Hodnota pixelu $P_{(\mathrm{i,j})}$ sa pri prvom prechode vypočíta nasledovne:

$$P_{(i,j)} = min(P_{(i,j)}, P_{(i,j-1)}+v_y, P_{(i-1,j)}+v_x, P_{(i-1,j-1)}+\sqrt{v_x^2+v_y^2}, P_{(i+1,j-1)}+\sqrt{v_x^2+v_y^2}) \tag{1.12}$$

Hodnota pixelu $P_{(\mathrm{i,j})}$ sa pri druhom prechode vypočíta nasledovne:

$$P_{(i,j)} = min(P_{(i,j)}, P_{(i,j+1)}+v_y, P_{(i+1,j)}+v_x, P_{(i+1,j+1)}+\sqrt{v_x^2+v_y^2}, P_{(i-1,j+1)}+\sqrt{v_x^2+v_y^2}) \tag{1.13}$$

(a)

2	**2,4**	**2,8**	**3,8**	**4,8**
1	**1,4**	**2,4**	**3,4**	**4,4**
0	**1**	**2**	**3**	**4**
1	**1,4**	**2,4**	**3,4**	**4,4**
2	**2,4**	**2,8**	**3,8**	**4,8**

(b)

Obr. 1.11: Vzdialenostná transformácia v chamfer metrike aplikovaná na vzorový obraz 1.2.

Ako už bolo spomenuté, zo všetkých typov metrík dosahuje výsledky najpodobnejšie Euklidovskej metrike práve chamfer metrika. Najmenej výpočtovo náročná je metrika mestských blokov, ktorá je vďaka svojej rýchlosti použitá v tejto práci ako metrika na výpočet vzdialenostnej transformácie.[2]

1.1.2 Hranové párovanie

Hranové párovanie je druhý krok hranového vzdialenostného párovania. Pre tento krok je nutné mať vzdialenostnú mapu, ktorej vytvorenie je vysvetlené v predchádzajúcom odseku a vzor, za pomoci ktorej sa počíta hranová vzdialenosť. Vďaka svojej presnosti sa často používa vzdialenostná mapa vytvorená v chamfer metrike, (z toho aj názov Chamfer Matching) ale nie je to pravidlo.

Ako vzor si je možné predstaviť množinu kľúčových bodov, ktoré spolu tvoria hranový obraz objektu. Každý z týchto kľúčových bodov nesie informáciu o pozícii na vzdialenostnej mape. Inak povedané, každý kľúčový bod ukazuje na pozíciu na vzdialenostnej mape tak, ako je to znázornené na obrázku 1.12. Ak teda vďaka kľúčovým bodom vieme ktoré hodnoty na vzdialenostnej mape sú pre nás dôležité, môžeme za použitia nasledovného vzťahu zistiť hodnotu hranovej vzdialenosti M_{RMS}.

$$M_{RMS} = \frac{1}{3}\sqrt{\frac{1}{n}\sum_{i=1}^{n} v_i^2} \tag{1.14}$$

Vo vzorci 1.14 je n počet kľúčových bodov vzoru a v_{i} je hodnota súradnice vzdialenostnej mapy, na ktorú ukazuje i-ty kľúčový bod vzoru.[13]

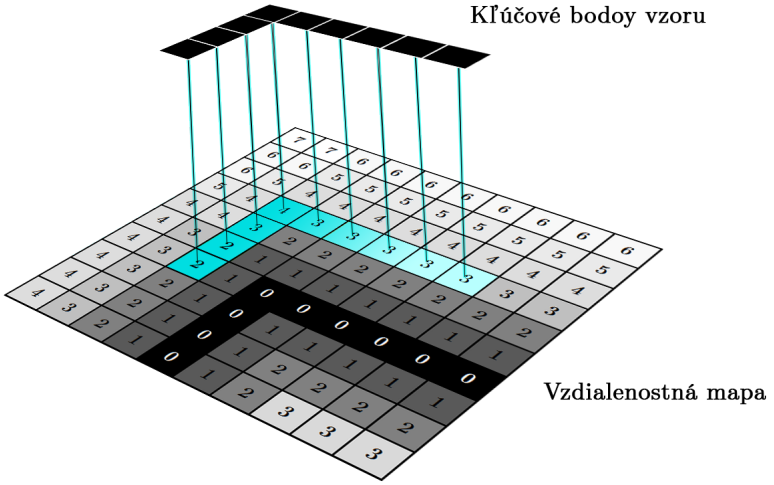
Posunom vzoru po vzdialenostnej mape hľadáme práve pozíciu vzoru, pri ktorej je hranová vzdialenosť M_{RMS} minimálna. Taká pozícia je považovaná za pozíciu objektu na snímke.

Výhodou tohto algoritmu je, že je výpočtovo nenáročný, je odolný voči šumu a viacerým objektom na snímke.

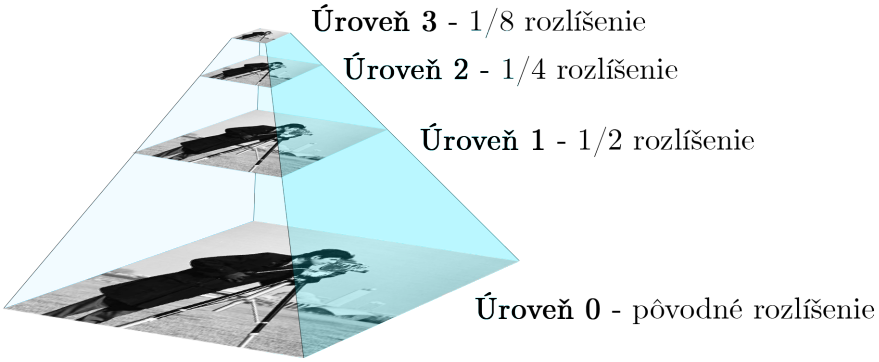
Nevýhodou je, že kvalita detekcie závisí od detailov a presnosti vzoru. Algoritmus taktiež nepočíta s rotáciou alebo zmenou veľkosti vzoru a preto je nutné vytvoriť viacero vzorov. To môže byť pamäťovo veľmi náročné a z toho dôvodu je možné zredukovať počet kľúčových bodov výberom vhodných bodov v rámci vzoru. [7, 13, 15]

1.2 Pyramídová reprezentácia

Pre zrýchlenie výpočtov sa v počítačovom videní využíva pyramídová reprezentácia informácií. Táto reprezentácia spočíta v redukcii informácií s cieľom zvýšenia výpočtovej rýchlosti. Obraz, na ktorom sa vykonávajú výpočty, je postupne zmenšený 2^n krát, kde n označuje úroveň pyramídy. Takýto postup sa využíva hlavne pri detekcii objektov. Detekcia začína na najvyššej úrovni pyramídy, kde sa nájde najvyššia zhoda so vzorom. Pokračuje sa na nižšej úrovni v okolí predošlej najvyššej zhody. Takýmto spôsobom sa pokračuje cez všetky úrovne pyramídy. Po vykonaní detekcie na nultej úrovni sa detekcia ukončí.[29]



Obr. 1.12: Názorná ukážka hranového párovania. Čierna farba na vzdialenostnej mape vyznačuje pozície hrán objektu. Modrá farba vyznačuje pixely vzdialenostnej mapy, na ktoré ukazujú kľúčové body vzoru.



Obr. 1.13: Názorná ukážka obrazovej pyramídy so 4 úrovňami na snímke s fotografom. [33]

2 POUŽITÉ VÝVOJOVÉ NÁSTROJE

Táto kapitola popisuje vývojové nástroje použité v tejto práci, ich výhody a dôvody použitia.

2.1 Visual Studio 2015

Ako IDE bolo použité Visual Studio 2015 s aktivovanou študentskou licenciou. Visual Studio je vývojový nástroj určený pre tvorbu počítačových programov, webových stránok, webových aplikácií či mobilných aplikácií. Je vyvíjané spoločnosťou Microsoft. Programovací jazyk použitý pri vytváraní programu bol C++.[25]

Výhodou VS je možnosť využitia študentskej licencie, široké spektrum možností, veľká užívateľská komunita a množstvo doplnkov.

2.2 OpenCV 3.2

OpenCV verzia 3.2 je najnovšia verzia voľne dostupnej knižnice určenej na spracovanie obrazu a počítačové videnie. Bola vytvorená ruskou divíziou firmy Intel, aby poskytovala bežnú podporu pre počítačové videnie a pre zrýchlenie vnímania zariadení v komerčných produktoch.

Obsahuje viac ako 2500 optimalizovaných algoritmov. Tieto algoritmy môžu byť použité na detekciu tvárí, identifikáciu objektov, rozoznanie pohybov osôb, zisťovanie pohybu kamery, zistenie pohybu objektov, vytvorenie 3D modelov, získanie 3D mračien bodov z dvoch kamier, spájanie obrazov pre získanie obrazov s vysokým rozlíšením, hľadanie rovnakých obrazov z databázy, rozoznanie pohybu očí. Túto knižnicu využívajú hlavne firmy, vývojové skupiny a taktiež vládne kruhy. OpenCV má široké využitie od spájanie obrázkov pre Streetview, detekciu osôb na bezpečnostných kamerách, pomáhanie robotom pri zdvíhaní vecí, detekcia pohybov topiacich sa ľudí, kontrola odtokových ciest, inšpekcia značenia na produktoch atď.

Podporuje programovacie jazyky C++, C, Python, Java a MATLAB a podporuje operačné systémy Windows, Linux, Android and Mac OS.[12]

Dôvod výberu tejto knižnice bola jej jednoduchosť, efektívnosť, voľná licencia a obrovské množstvo používateľov.

OpenCV 3.2 bolo získané z oficiálnej stránky, vygenerované za pomoci programu CMake a importované do VS.

Z tejto knižnice boli v tejto práci použité nasledovné triedy: `LineIterator`, `Mat`, `Point`, `Point2f`, `Point2i`, `Size`, `Scalar`, `Vec2w`, `Vec3b`, `Vec3f`, `Vec3w`, `VideoCapture`.

Z tejto knižnice boli v tejto práci použité nasledovné funkcie a metódy: `arrowedLine`, `bitwise_not`, `Canny`, `cvtColor`, `distanceTransform`, `drawContours`, `findNonZero`, `GaussianBlur`, `imread`, `line`, `putText`, `threshold`, `Mat::convertTo`, `Mat::zeros`.

Pre ľahšiu prácu s obrazom bol do VS nainštalovaný doplnok Image Watch, za pomoci ktorého bolo možné v móde Debug prehliadať obrazy v pamäti programu vytvorené knižnicou OpenCV.

2.3 Qt 5.8.0

Qt 5.8.0 je najnovšia verzia multiplatformovej knižnice, určenej na programovanie a tvorbu GUI. Bola vyvíjaná od roku 1991 spoločnosťou Trolltech. V roku 2008 bola predaná a ďalej vyvíjaná firmou Nokia. Od roku 2014 patrí spoločnosti The Qt Company.

Qt podporuje programovacie jazyky C++, Phyton, Ruby, C, Pascal, C# a Java. Túto knižnicu využívajú programy ako Skype, Google Earth, či Opera.[31]

Pri tvorbe licencovaných programov je nutné pre knižnicu Qt zakúpiť licenciu. Qt však ponúka voľnú licenciu pri tvorbe neziskových programov, ktorá bola jedným z dôvodov využitia tejto knižnice v tejto práci. Ďalším z dôvodov využitia tejto knižnice bola veľká užívateľská komunita a multiplatformovosť knižnice.

Pre prácu s Qt bolo do VS nainštalované rozšírenie Qt5Package a pre návrh GUI bol použitý program Qt Designer.

3 PRAKTICKÁ REALIZÁCIA DETEKCIE OBJEKTOV

V tejto kapitole sú popísané postupy pri praktickej realizácii práce. Prvá časť tejto kapitoly je venovaná vzoru. Je tu vysvetlené, z akých informácií pozostáva vzor a ako je vytvorená sada vzorov. Ďalej je tu vysvetlený princíp vytvorenia vzoru z plošného obrazu, z priestorového modelu a spôsoby ukladania a načítavania vzorov z pamäte.

Druhá časť tejto kapitoly pozostáva zo samotnej praktickej realizácie detekcie. Sú v nej porovnané rýchlosti algoritmov vzdialenostnej transformácie pri použití rôznych metrík, vývojové diagramy programov vytvorených pre túto prácu a vysvetlenie pseudo-pyramídovej detekcie vytvorenej pre túto prácu. Na záver je tu vysvetlený spôsob vyhodnotenia detekcie.

V poslednej časti tejto kapitoly je popísaná trieda DetectionWindow, ktorá zabezpečuje GUI celého detekčného programu. Sú tu popísané vnútorné premenné a metódy tejto triedy, a ich úlohy. Ďalej je tu vizuálna ukážka GUI s popismi jednotlivých častí.

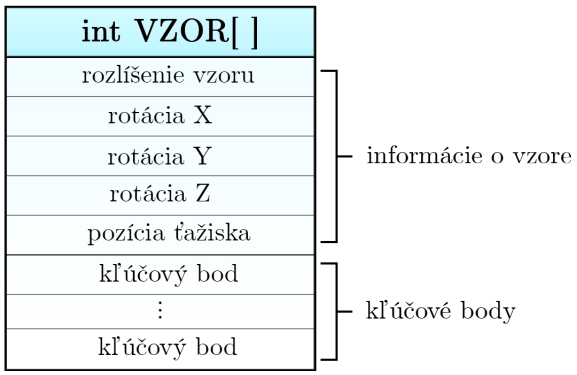
3.1 Vzor

V tejto podkapitole je vysvetlené z akých informácií vzor a sada vzorov pozostávajú, ako je vzor vytvorený z plošného obrazu a z priestorového modelu a akým spôsobom sú sady vzorov uložené a načítané z pamäte.

V odseku 1.1.2 bolo vysvetlené, že vzor si je možné predstaviť ako množinu kľúčových bodov nesúcich informáciu o pozícií hrany daného objektu. Avšak na to, aby bolo vzor možné použiť pre hranové párovanie, je nutné poskytnúť informáciu o rozlíšení daného vzoru. Táto informácia zabezpečí umožnenie pohybu vzoru po celej ploche vzdialenostnej mapy. Za pomoci takéhoto vzoru je možné detekovať objekt konkrétnej rotácie a priblíženie.

Pre detekciu viacerých rotácií a priblížení objektu, sa musí vytvoriť takzvaná sada vzorov a do vzoru sa musia pridať ďalšie informácie. Pre každé natočenie vzoru sa musí pridať informácia o rotačných uhloch objektu, pri ktorých bol vzor vytvorený. Poslednou informáciou o vzore je pozícia jeho ťažiska. Táto informácia sa využíva pri vykreslení výsledku detekcie. Výsledný vzor obsahuje informácie znázornené na obrázku 3.1.

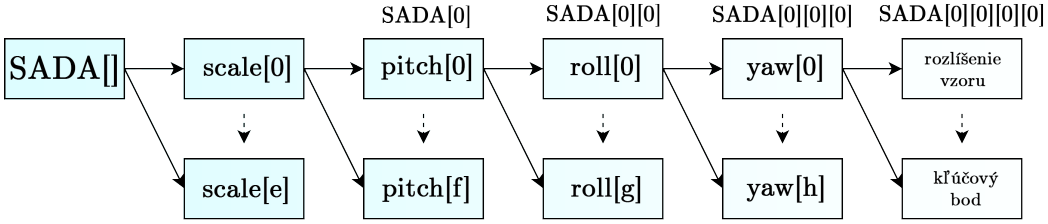
Dátový typ v ktorom sú zapuzdrené informácie o vzore je `int`. Tento dátový typ je zvolený z dôvodu nutnosti znamienkového dátového typu pre označenie záporných uhlov rotácie. Ďalším dôvodom je dostatočná presnosť detekcie pri celočíselnom kroku rotačných uhlov.



Obr. 3.1: Grafická interpretácia informácií obsiahnutých vo vzore.

Z dôvodu použitia tohto dátového typu je nutné kľúčové body definovať pomocou lineárnej indexácie znázornenej na obrázku 1.1b. Maximálna hodnota tohto dátového typu je 2 147 483 647.[23] Ak by mal vzor tvar štvorca, jeho rozlíšenie by muselo presahovať hodnotu $\sqrt{2\,147\,483\,647} \times \sqrt{2\,147\,483\,647}$, čo odpovedá rozlíšeniu $46\,340 \times 46\,340$, aby tento dátový typ pretiekol. Tento stav sa však pri súčasne používaných rozlíšeniach snímok, ktoré sú podstatne menšie nepredpokladá.

Sada vzorov je v tejto práci 5-rozmerné pole, ktorého elementárnym prvkom je dátový typ `int`. Pre prácu s 5-rozmerným poľom je použitá trieda `std::vector`, ktorá dokáže so všetkými svojimi prvkami pracovať dynamicky. Jednotlivým rozmerom sú priradené nasledovné vlastnosti: priblíženie (scale), rotácia v osi Z (pitch), rotácia v osi Y (roll) a rotácia v osi X (yaw).



Obr. 3.2: Grafická interpretácia jednotlivých rozmerov sady vzorov.

Premenné *e*, *f*, *g* a *h* na obrázku 3.2 predstavujú maximálny index prvku v danom rozmere. Všetky prvky *yaw* obsahujú základné informácie vzoru podľa obrázku 3.1. V závislosti od zvoleného indexu sú v prvkoch *yaw* striedavo obsiahnuté všetky kľúčové body, alebo zredukované množstvo kľúčových bodov.

Pod párnymi indexmi je zapuzdrené zredukované množstvo kľúčových body vzoru. Tieto indexy sa využívajú pri prvom kroku hranového párovania pre zrýchlenie detekcie. Nepárne indexy zapuzdrujú všetky kľúčové body toho istého vzoru a vy-

užívajú sa pri druhom kroku hranového párovania. Kroky hranového párovania sú znázornené na vývojovom diagrame 3.9. Celkový počet vzorov v sade vzorov je tým pádom daný:

$$N = \frac{(e+1)(f+1)(g+1)(h+1)}{2} \tag{3.1}$$

Dôvod podielu je ten, že každý vzor je v sade obsiahnutý dvakrát. Raz v zredukovanej a raz v plnej podobe.

Vytvorenie sady vzorov si je teda možné predstaviť ako napĺňanie prvkov *VZOR* vhodnými údajmi. V prvom kroku tvorby vzoru je potrebné vytvoriť hranový (binárny) obraz objektu ktorý chceme detekovať. Pre vytvorenie takéhoto obrazu je potrebné definovať presnú pozíciu a rotáciu objektu, z ktorého chceme vytvoriť vzor. Taktiež je nutné definovať ohniskovú vzdialenosť kamery pre perspektívne zobrazenie. Po tomto kroku sa môže zo vstupného objektu vyrenderovať[1] hranový (binárny) obraz.

Na takomto obraze sa pomocou OpenCV funkcie `findNonZero` nájdu nenulové (biele) body, ktoré reprezentujú hrany objektu a uložia sa ako *kľúčové body*. Takisto sa získajú informácie o *rozlíšení vzoru* a *pozícii ťažiska*. Hodnoty rotácií okolo osí X, Y a Z sú priamo hodnoty rotácií, podľa ktorých je model renderovaný. Spôsoby akými sa renderuje obraz z rôznych vstupov sú vysvetlené v nasledovných odsekoch.

3.1.1 Renderovanie plošného objektu

V tomto odseku bude vysvetlený spôsob transformácie plošného objektu do priestoru.

Pre vyrenderovanie binárneho obrazu z ľubovoľného vstupného obrazu je nutné zo vstupného obrazu získať kľúčové body. Tieto body budú nasledovne reprezentovať priestorové body pre renderovanie objektu.

Kľúčové body sú zo vstupného obrazu získané pomocou OpenCV funkcie `Canny`, ktorá vykonáva algoritmus Cannyho hranového detektoru, čím sa obraz v úrovniach šedej premieňa na binárny obraz. Keďže vstupom môže byť obrázok s ľubovoľným kontrastom, je vhodné vyhnúť sa statickému nastaveniu prahovania v tejto funkcii.

Použitím Otsuovej metódy prahovania je možné získať optimálnu hodnotu prahu pre daný obraz. Táto metóda vytvorí z obrazu v úrovniach šedej odtieňový histogram, z ktorého určí optimálnu hodnotu prahu.[30] Praktická ukážka použitia tejto metódy je znázornená vo výpise 3.1.

[1] renderovanie - vytvorenie obrazu z 2D alebo 3D modelov za pomoci počítačového softvéru.

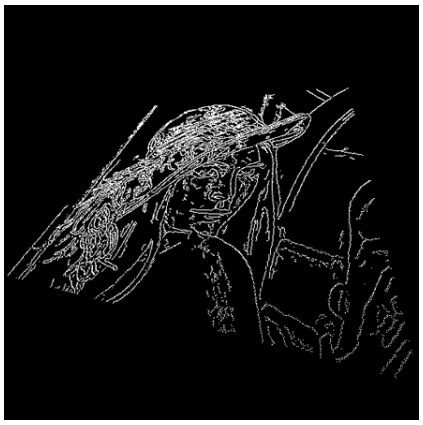
Výpis 3.1: Príklad použitia dynamického prahovania za pomoci knižnice OpenCV v jazyku C++.

```
// získanie optimálneho prahu pomocou Otsuovej metódy
double dynamicThreshold = threshold(
    gray                                ,
    threshold                           ,
    0                                   ,
    UCHAR_MAX                           ,
    CV_THRESH_BINARY | CV_THRESH_OTSU );

//aplikovanie optimálneho prahu na funkciu Canny
Canny(
    gray                    ,
    canny                   ,
    0                       ,
    dynamicThreshold        );
```

Na takto pripravený binárny obraz aplikujeme OpenCV funkciu `findNonZero` a nájdené nenulové (biele) body uložíme ako priestorové body renderovaného objektu. Polohy takto získaných vrcholov sa za pomoci rotačných, translačných, zväčšovacích a projekčných matíc prepočítajú a vykreslí sa z nich obraz.[6, 26] Takýto binárny obraz je následne možné použiť pre vytvorenie vzoru.

(a) Pôvodná pozícia vrcholov.



(b) Transformovaná pozícia vrcholov.

Obr. 3.3: Ukážka transformačných matíc použitých na vrcholoch získaných zo skúšobnej snímky. [32]

3.1.2 Renderovanie priestorového objektu

V tejto kapitole bude vysvetlený spôsob zobrazenia priestorového objektu uloženého v súboroch typu OBJ.

Súbory OBJ sú súbory, ktoré popisujú všetky svoje telesá za pomoci konvexných mnohouholníkov, najčastejšie trojuholníkov. Dôvod používania trojuholníka ako elementárneho prvku je ten, že trojuholník je najjednoduchší mnohouholníkový útvar a ako jediný zo všetkých mnohouholníkov je vždy konvexný. Pre konvexný objekt platí, že spojením ľubovoľných dvoch bodov patriacich tomuto objektu vznikne úsečka, ktorá všetkými svojimi bodmi patrí danému objektu. Konvexnosť sa využíva hlavne pri počítaní normály plochy a pri textúrovaní plochy. Pre prácu so súbormi typu OBJ je nevyhnutné pochopiť spôsob, akým sú v nich dáta uchované. Tento spôsob bude vysvetlený na najjednoduchšom priestorovom telese, ktorým je štvorsten.[6, 18, 26]

Štvorsten je teleso pozostávajúce zo štyroch trojuholníkových stien, štyroch vrcholov a šiestich hrán. Jeho reprezentácia v súbore OBJ je zobrazená vo výpise 3.2.

Výpis 3.2: Textová podoba súboru OBJ obsahujúca model štvorstenu. [39]

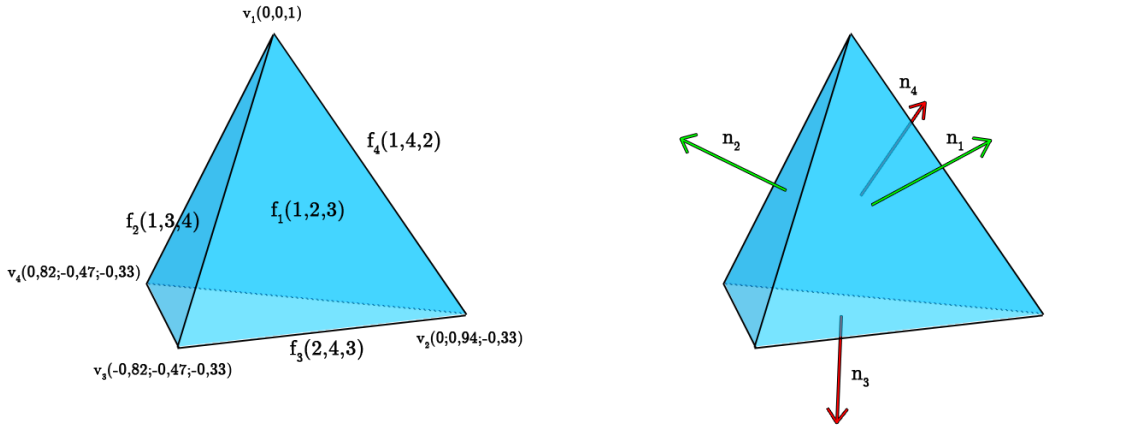
```
1 v 0            0            1
2 v 0            0.942809    -0.33333
3 v -0.816497    -0.471405    -0.33333
4 v 0.816497     -0.471405    -0.33333
5
6 f  1  2  3
7 f  1  3  4
8 f  2  4  3
9 f  1  4  2
```

Riadky vo výpise 3.2, ktoré začínajú písmenom 'v' (riadky 1-4) sú riadky obsahujúce informácie o priestorovej polohe jednotlivých vrcholov (vertices) štvorstenu. Riadky začínajúce písmenom 'f' (riadky 6-9) nesú informáciu o indexoch troch vrcholov, ktoré navzájom tvoria stenu (face) objektu a tým pádom aj informácie o troch hranách, ktorými sú vrcholy navzájom prepojené. Priestorovo si je možné model predstaviť tak, ako je to znázornené na obrázku 3.4a.

Ak by sa tieto dáta využili na vykreslenie hrán objektu, výsledok vykreslenia by bol taký, že by sa vykreslili aj hrany ktoré majú byť neviditeľné. Vykreslil by sa tzv. drôtený model objektu.[17] Preto je nutné za pomoci vektorového súčinu dvoch po sebe nasledujúcich hrán vypočítať normálu každej zo štyroch plôch štvorstenu. Postupnosť hrán je daná poradím vrcholov v súbore OBJ na riadkoch začínajúcich

písmenom 'f'. Obecný vzorec pre výpočet normály plochy f_i je nasledovný:[19]

$$n_i = (f_i[1] - f_i[0]) \times (f_i[2] - f_i[1]) \tag{3.2}$$



(a) Štvorsten s vyznačenými vrcholmi a stenami.

(b) Štvorsten s vyznačenými normálami plôch. Zelené normály sú priklonené ku pozorovateľovi, červené sú odvrátené.

Obr. 3.4: Grafická interpretácia štvorstenu.

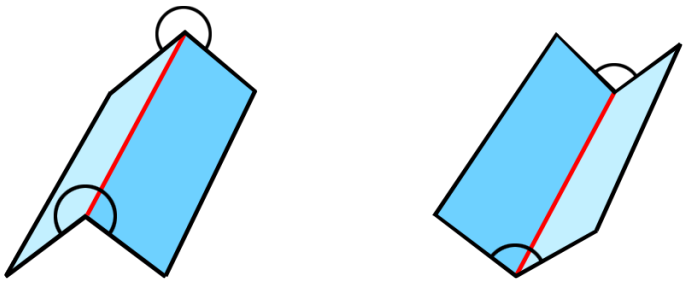
Ak zistená normála smeruje ku kamere, potom sa hrany danej plochy vykreslia. Hrany patriace ploche, ktorej normála smeruje od kamery sa nevykreslia. Ak je pozícia kamery definovaná v počiatku zobrazovacieho priestoru, stačí zisťovať aké znamienko má Z súradnica normály danej plochy.

Takéto riešenie už bude štvorsten zobrazovať správne. Ďalšie opatrenia, ktoré je nutné pridať je zistenie, či uhol medzi dvoma plochami tvoriacimi hranu nie je približne 180° a takéto hrany nevykresľovať. Program s takýmto riešením už bude správne zobrazovať všetky konvexné modely.

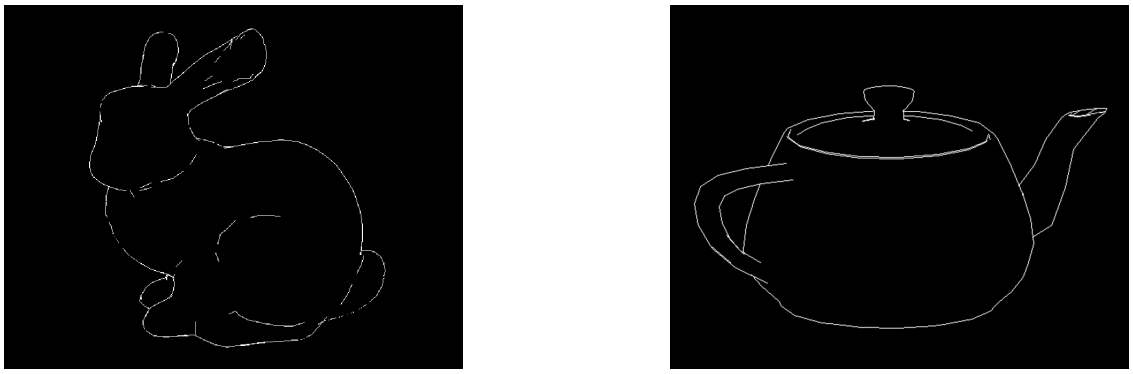
Pre správne zobrazenie konkávnych modelov je nutné zistiť, či hrany tvorené dvoma stenami majú konvexný, alebo konkávny charakter. Konvexný charakter hrany je vtedy, ak je uhol medzi plochami ktoré tvoria hranu väčší ako 180°. V opačnom prípade je charakter hrany konkávny. Ak majú hrany konvexný charakter, tak budú vykresľované aj vtedy, ak jedna normála plochy smeruje ku kamere a druhá nie. Tento prípad nastane vtedy, ak je hrana súčasne vonkajším obrysom telesa. V opačnom prípade, kedy má hrana konkávny charakter sa táto hrana nevykreslí. Ukážku konvexnej a konkávnej hrany je možné vidieť na obrázku 3.5.

Posledným ošetrením pre správne zobrazenie konkávnych modelov je zoradenie trojuholníkov modelu podľa vzdialenosti od kamery, tzv. vzdialenostný Z-buffer. Takto zoradené trojuholníky sa následne vyplnia čiernou farbou v poradí od najvzdialenejšieho ku najbližšiemu. Týmto krokom sa odstránia akékoľvek obrysové

hrany ktoré boli vykreslené, no nachádzajú sa za inou časťou objektu. Na obrázku 3.6 je možné vidieť výsledný render skúšobných OBJ modelov.[2]



Obr. 3.5: Názorná ukážka konvexnej hrany (vľavo) a konkávnej hrany (vpravo).



(a) Stanfordský králik [37]

(b) Utahský čajník [38].

Obr. 3.6: Ukážka renderovania známych skúšobných modelov.

3.1.3 Ukladanie a načítavanie vzorov

Ku vytvoreniu vzoru sa viažu aj spôsoby jeho uloženia do pamäte a následné načítavanie z nej. V tomto odseku sú tieto spôsoby vysvetlené.

Pre ukladanie vzoru je nutné určiť, ktoré dáta sú nevyhnutné pre jeho opätovné použitie. Prvým dôležitým údajom je samotný vzor obsahujúci informácie o jeho rozmeroch, rotáciách, pozícii ťažiska a lineárne indexovaných kľúčových bodov. Práve kvôli použitiu lineárnej indexácie je ďalším dôležitým údajom rozlíšenie, z ktorého boli lineárne indexy vzoru určené. Posledným potrebným údajom je maximálne rozlíšenie, ktoré sada vzorov dosahuje. Toto rozlíšenie sa využíva pri určovaní povolenia, použiť danú sadu na detekciu v konkrétnej scéne. Ak je jeden z rozmerov tohto rozlíšenia väčší, ako rozmer danej scény, detekcia nie je povolená.

Tieto údaje sú následne vo forme textu uložené do súboru typu CMT (Chamfer Template). Skrátená textová podoba takéhoto súboru vyzerá nasledovne:

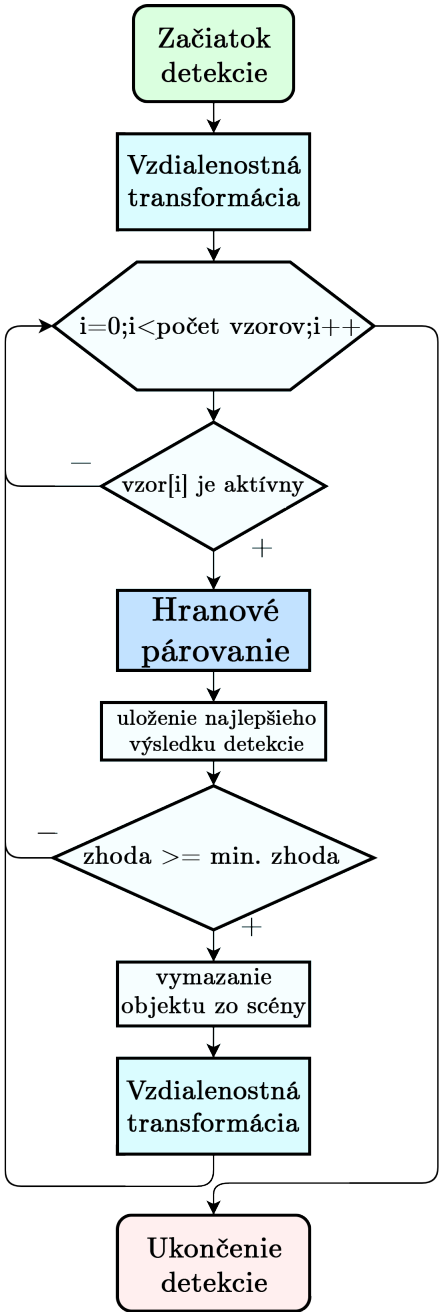
Výpis 3.3: Textová podoba súboru CMT opisujúceho jeden vzor.

```
1 800x600
2 550x400
3
4 ;350000,0,0,0,176000,4583,17003, ... ,338775,345803!
5
6 ;350000,0,0,0,176000,174,1054,1055,1056,1057, ...
7 ... ,352888,352889,352890,352891,352892!!!!!
```

Prvý riadok výpisu 3.3 obsahuje rozlíšenie, pri ktorom boli určené lineárne indexy kľúčových bodov. Druhý riadok obsahuje maximálne rozlíšenie, ktoré sada vzorov dosahuje. Znak ';' na začiatku riadkov 4 a 6 znamená začiatok informácií pre konkrétny vzor. Ukončenie týchto dát je sprevádzané znakom '!'. Samotný znak '!', nasledovaný znakom ';' označuje ukončenie informácií jedného vzoru a pokračovanie načítavania nového vzoru do inkrementovaného indexu vektoru *yaw* znázornenom na obrázku 3.2 . Dva a viac po sebe nasledujúce znaky '!' určujú nutnosť inkrementácie indexov vektorov *roll*, *pitch*, alebo *scale*, v závislosti od ich počtu.

3.2 Hranové vzdialenostné párovanie

V tejto podkapitole je vysvetlená praktická realizácia algoritmu pre vzdialenostnú transformáciu a hranové párovanie. Nasledovne je tu popísaný princíp použitej pyramídovej detekcie a spôsob detekcie viacerých objektov v snímke. Na obrázku 3.7 je zobrazený vývojový diagram praktickej realizácie detekcie v tejto práci.



Obr. 3.7: Vývojový diagram detekcie.

3.2.1 Vzdialenostná transformácia

Pre vzdialenostnú transformáciu je v tejto práci vytvorená funkcia pracujúca v metrike mestských blokov. Dôvodom použitia tejto metriky je jej vysoká rýchlosť, viď. tabuľka 3.1 a dostačujúce výsledky.

Metóda použitá v tejto práci ďalej používa hodnotu váh v_x a v_y použitých v maske o veľkosti *32*. Tým je dosiahnuté, že v obraze s 8-bitovými farbami zasahujú úrovne šedej menšiu plochu, ako pri použití váhy o hodnote 1, čo zabezpečí vyššiu presnosť pri detekcii a prísnejšie hodnotenie percentuálnej zhody.

Tab. 3.1: Porovnanie použitia rôznych metód vzdialenostnej transformácie na snímke zobrazenej na obrázku 1.2a s rozlíšením 1200x600.

Použitá metóda	priemerný čas výpočtu 100 transformácií [s]
metrika mestských blokov (OpenCV)	0,506
šachovnicová metrika (OpenCV)	0,426
chamfer metrika (OpenCV)	0,444
metrika mestských blokov (vlastná)	0,283
šachovnicová metrika (vlastná)	0,457

64	**96**	**128**	**160**	**192**
32	**64**	**96**	**128**	**160**
0	**32**	**64**	**96**	**128**
32	**64**	**96**	**128**	**160**
64	**96**	**128**	**160**	**192**

Obr. 3.8: Zobrazenie výsledku vzdialenostnej transformácie použitej v tejto práci.

3.2.2 Hranové párovanie

Hranové párovanie pracuje na podobnom princípe ako je vysvetlené v odseku 1.1.2, ale namiesto rovnice 1.14 je použitý jej zjednodušený tvar:

$$M = \frac{1}{n} \sum_{i=1}^{n} v_i \tag{3.3}$$

Tento tvar počíta hranovú vzdialenosť M ako priemer všetkých hodnôt vzdialenostnej mapy, na ktoré ukazujú kľúčové body. Výhodou takéhoto výpočtu je, odstránenie výpočtu mocnín a odmocnín, čo značne urýchli dobu detekcie. Praktickú realizáciu algoritmu popisuje vývojový diagram na obrázku 3.9.

Hranové párovanie prebieha v dvoch krokoch. V prvom kroku sa za využitia vzoru s zredukovaným množstvom kľúčových bodov zistí hranová vzdialenosť sum_{R}. Ak je táto vzdialenosť menšia, ako najmenšia dovtedy zistená hranová vzdialenosť M_{R}, algoritmus pokračuje do druhého kroku. V opačnom prípade hranové párovanie pokračuje nasledujúcim vzorom.

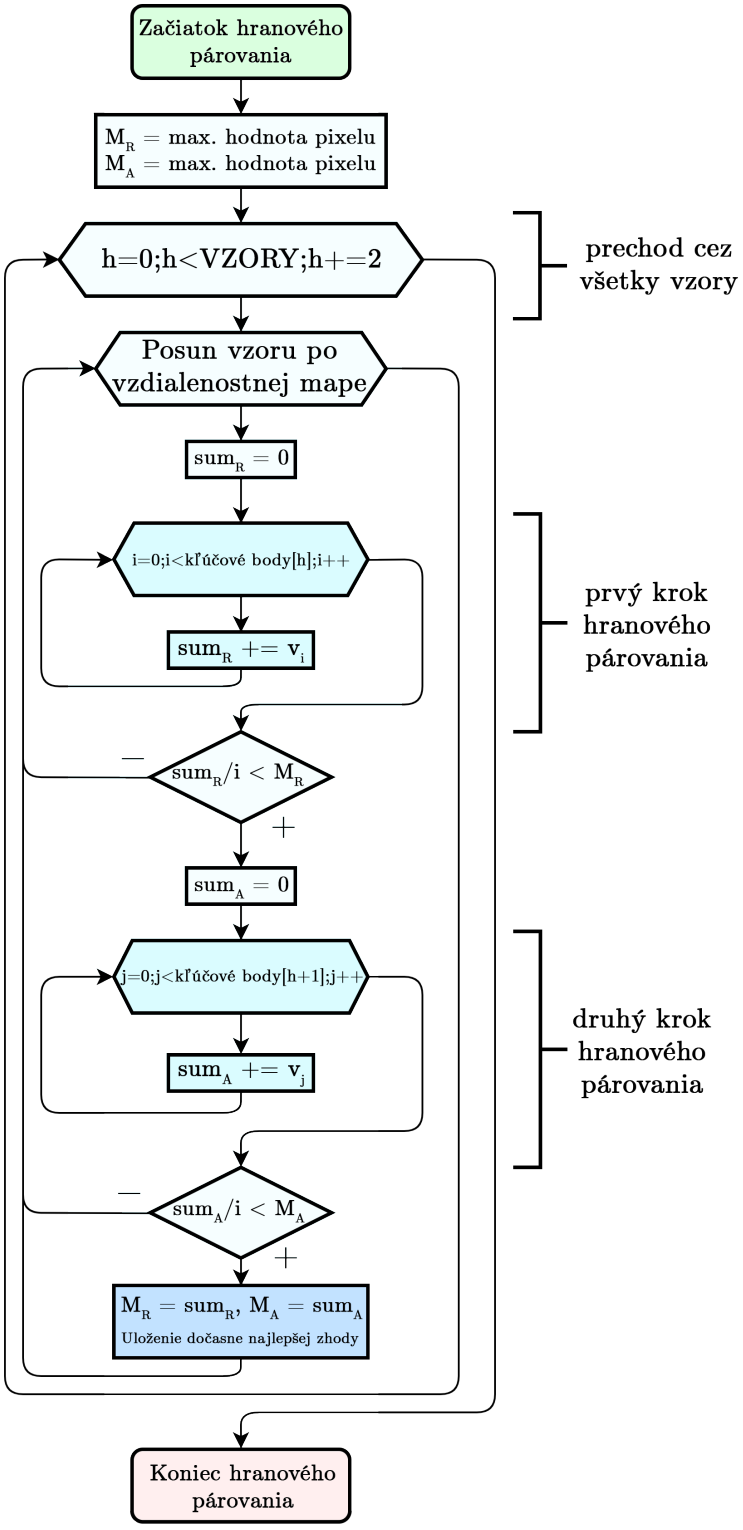
V druhom krokom hranového párovania sa zistí hranová vzdialenosť sum_{A} pre vzor obsahujúci všetky kľúčové body. Ak je táto vzdialenosť menšia, ako najmenšia dovtedy zistená hranová vzdialenosť M_{A}, nastavia sa nové, najmenšie hodnoty M_{R} a M_{A} a daný vzor a jeho pozícia sa uložia ako nové najlepšie pozície objektu.

Posun vzoru po vzdialenostnej mape

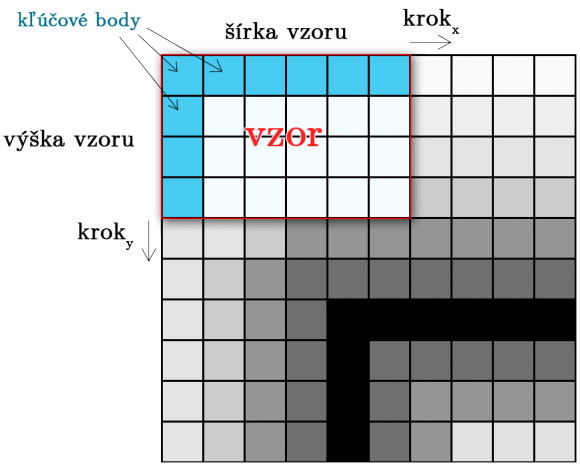
Keďže vzor je vo väčšine prípadov menší ako rozlíšenie vzdialenostnej mapy, musí byť zabezpečený jeho posun po celej jej ploche. Tým sú získané všetky možné hodnoty hranovej vzdialenosti M, z ktorých sa detekcia vyhodnotí.

Z pravidla sa začína v ľavom hornom rohu vzdialenostnej mapy a postupuje sa zľava doprava s krokom $krok_{\mathrm{x}}$ a zvrchu nadol s krokom $krok_{\mathrm{y}}$, tak ako je to znázornené na obrázku 3.10. Použitím hodnoty 1 pre kroky $krok_{\mathrm{x}}$ a $krok_{\mathrm{y}}$ sa zabezpečí vykonanie hranového párovania pre každú možnú polohu vzoru na vzdialenostnej mape.

V tejto práci je veľkosť premenných $krok_{\mathrm{x}}$ a $krok_{\mathrm{y}}$ pri hranovom párovaní nastavená na hodnotu 5. Použitím tejto hodnoty boli dosiahnuté dostačujúce výsledky a zníženie času detekcie.



Obr. 3.9: Vývojový diagram hranového párovania.



Obr. 3.10: Zobrazenie posunu vzoru po vzdialenostnej mape.

3.2.3 Pseudo-pyramídové hranové párovanie

Pojem pseudo-pyramídového hranového párovania v tejto práci označuje detekciu, ktorá za účelom zmenšenia detekčného času pyramídovo redukuje množstvo dát, na ktorých je vykonané hranové párovanie.

Každý vzor má kľúčové body vytvorené v jednom rozlíšení. Z toho dôvodu by sa pri pyramídovom zmenšovaní rozlíšenia detekčnej scény, ako je to popísané v odseku 1.2, musel pre každú úroveň pyramídy vytvoriť nový vzor, ktorý by mal kľúčové body vytvorené pri danom rozlíšení. To by však bolo pamäťovo náročné a čas vytvorenia sady vzorov by sa značne predĺžil.

V tejto práci je tento problém riešený zachovaním rozlíšenia detekčnej scény a zmenou kroku, ktorým vzory sady prechádzajú.

Každá sada vzorov pozostáva zo 4 rozmerov ktoré určujú polohu a orientáciu objektu (zväčšenie a 3 rotácie) a 1 rozmeru nesúceho základné informácie o vzore. Počet vzorov, z ktorých sada pozostáva je daný vzťahom 3.1. Pre sadu vzorov, ktorá obsahuje všetky možné rotácie modelu pri konštantnom zväčšení sa počet vzorov v tejto sade určí nasledovne: $360^3 = 46\,656\,000$. To je však príliš veľké množstvo dát na detekciu. Preto pyramídová detekcia redukuje maximálny počet kontrolovaných vzorov na jednej úrovni na $90^3 = 729\,000$, bez ohľadu na to aký je celkový počet vzorov v sade. Hodnota 90 bola testovaním určená ako optimálna. Pri tejto hodnote detekcia dosahuje vysokého výkonu a presnosti. Maximálna úroveň pyramídy v tejto práci je 5.

V prvom kroku pseudo-pyramídového hranového párovania sa od najvyššej úrovne prechádzajú všetky úrovne pyramídy. Veľkosť kroku cyklov, ktoré prechádzajú sadu vzorov, sa pre každý rozmer sady určuje osobitne pred spustením prvého kroku. Krok

cyklu môže nadobúdať iba hodnoty 2^n a je vždy menší alebo rovný ako $2^{úroveň\ pyramídy}$. Premenná n sa vypočíta ako:

$$n = \log\left(\frac{\text{veľkosť rozmeru}}{90}\right) / \log(2) \tag{3.4}$$

Kde veľkosť rozmeru označuje počet prvkov v danom rozmere a výsledné n je zaokrúhlené smerom nahor a vždy v rozsahu <0;maximálna úroveň pyramídy>.

Súčasne sa mení aj krok, ktorým je vzor posúvaný po vzdialenostnej mape. Namiesto konštantného kroku o veľkosti 5 sa veľkosti $krok_{\mathrm{x}}$ a $krok_{\mathrm{y}}$ určia na každej úrovni pyramídy podľa vzorca:

$$krok_{x,y} = \frac{min(\text{šírka vzoru}, \text{výška vzoru}) \cdot 2^{\text{úroveň pyramídy}}}{100} \tag{3.5}$$

Hodnota 100 bola testovaná na viacerých rozmeroch vzorov a určená ako optimálna hodnota.

Po dosiahnutí 0. úrovne pyramídy sa ukončí prvý krok pseudo-pyramídovej detekcie. V druhom, finálnom kroku sa vzor s najlepším výsledkom posúva v okolí pozície s najlepším výsledkom s krokom 1. Maximálna vzdialenosť v ktorej sa vzor pohybuje je daná zo vzťahu 3.5 ako veľkosť kroku na 0. úrovni pyramídy. Táto vzdialenosť je udaná v šachovnicovej metrike.

Výsledky porovnania detekcie objektov za pomoci algoritmu hranového vzdialenostného párovania a pseudo-pyramídového hranového párovania sú znázornené v tabuľke 3.2

3.2.4 Detekcia viacerých objektov

Pri nájdení objektu s vyššou zhodou, ako je minimálna požadovaná zhoda je nutné tento objekt zo scény vymazať. Mazanie sa vykonáva na binárnom (hranovom) obraze scény, na ktorom sa mažú hrany, ktoré sú v okolí kľúčových bodov nájdeného objektu. Okolie je nastavené na vzdialenosť 5 pixelov od kľúčového bodu v šachovnicovej metrike. Takýmto spôsobom sa zabráni duplicitnej detekcii toho istého objektu.

3.3 Vyhodnotenie detekcie

V tejto podkapitole je vysvetlené, akým spôsobom sa vypočíta percentuálna zhoda nájdeného objektu s objektom v scéne a akým spôsobom sa detekcia zobrazí v GUI.

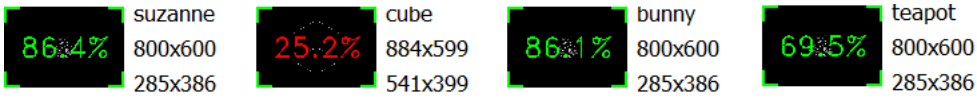
3.3.1 Výpočet percentuálnej zhody

Vzorec na výpočet percentuálnej zhody je odvodený z hodnoty hranovej vzdialenosti M zistenej pri detekcii. Keďže táto hodnota môže byť iba v rozsahu od 0 do

max. hodnota pixelu, kde hodnota 0 značí najvyššiu zhodu, prevod na percentá je nasledovný:

$$S = \left(1 - \frac{M}{max.\ hodnota\ pixelu}\right) \cdot 100\ [\%] \tag{3.6}$$

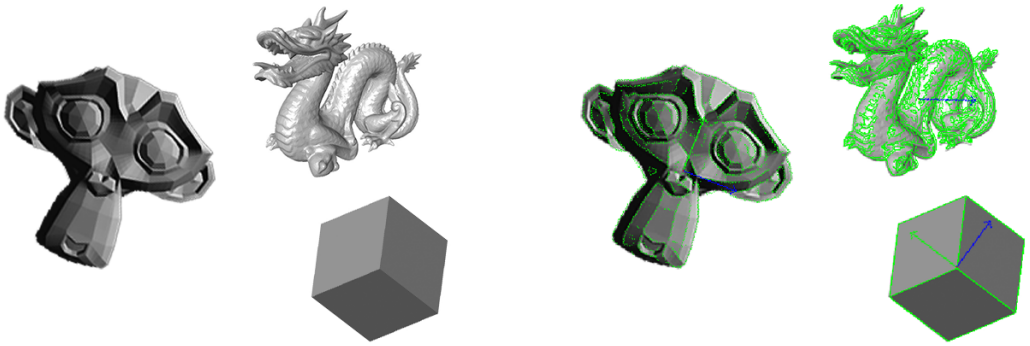
V GUI sa percentuálna zhoda vypíše v miniatúre konkrétneho vzoru. Ak je percentuálna zhoda vyššia ako minimálna nastavená percentuálna zhoda, vypíše sa zeleným písmom, inak červeným.



Obr. 3.11: Ukážka výpisu percentuálnej zhody v GUI.

3.3.2 Zobrazenie nájdeného objektu

Vzor, ktorý pri detekcii dosiahne vyššiu percentuálnu zhodu ako je minimálna nastavená percentuálna zhoda, sa vykreslí do pôvodnej scény zelenou farbou. Užívateľ si takisto môže zapnúť zobrazenie vlastných súradnicových osí vzoru.



(a) Scéna pred detekciou

(b) Scéna po detekcií

Obr. 3.12: Ukážka zobrazenia detekovaných objektov [35, 36, 34] a ich súradnicových osí v scéne.

Tab. 3.2: Porovnanie algoritmov obyčajného hranového párovania (HP) a pseudo-pyramídového hranového párovania (P-PHP).

Rozlíšenie snímky	**Objekty**	HP		P-PHP	
		Zhoda [%]	Čas [s]	Zhoda [%]	Čas [s]
640 x 480	1. objekt	77,1	0,763	85,2	0,023
	2. objekt	89,8		73,6	
	3. objekt	77,8		63,5	
800 x 600	1. objekt	82,8	2,276	86,4	0,069
	2. objekt	82,5		86,1	
	3. objekt	80,5		69,5	
	4. objekt	72,5		85,6	
1280 x 720	1. objekt	82,8	7,684	65,9	0,152
	2. objekt	79,6		78,9	
	3. objekt	75,8		73,6	
	4. objekt	90,0		94,2	
	5. objekt	78,9		85,6	
	6. objekt	83,7		94,3	
	7. objekt	82,6		94,2	
2000 x 1000	1. objekt	99,7	12,709	91,3	0,202
	2. objekt	90,6		80,7	
	3. objekt	91,9		96,1	
	4. objekt	91,9		92,0	
	5. objekt	86,2		89,8	
	6. objekt	79,6		90,4	
4096 x 2160	1. objekt	82,8	103,905	78,9	1,439
	2. objekt	73,2		71,1	
	3. objekt	69,8		75,2	
	4. objekt	79,7		72,7	
	5. objekt	89,0		94,2	
	6. objekt	86,0		89,6	
	7. objekt	84,3		84,8	

3.4 Trieda DetectionWindow

Táto podkapitola patrí popisu triedy `DetectionWindow`, ktorá zabezpečuje tvorbu sád vzorov, detekciu a GUI.

Trieda `DetectionWindow` bola vytvorená za použitia knižnice Qt pre ľahšiu a rýchlejšiu prácu pri vytváraní sád vzorov a pri detekcii. Zapuzdruje všetky na to potrebné premenné a metódy. Pre spracovanie obrazu používa triedy a funkcie knižnice OpenCV.

Referenčné a detekčné snímky: Táto trieda dokáže pracovať so všetkými formátmi snímok a videí, ktoré podporuje knižnica OpenCV. Je v nej implementovaná funkcia manuálneho pretáčania videa. Súčasne dokáže pracovať s kamerami pripojenými ku počítaču a zobrazovať ich obraz.

Model: Trieda za pomoci vytvorených funkcií dokáže načítať snímky a súbory OBJ, vytvoriť z nich model a prepojením knižníc OpenCV a Qt model zobraziť.[20] Táto funkcia sa využíva pri vytváraní sád vzorov. Model je možné rotovať a zväčšovať/zmenšovať za použitia myši, alebo zmeny hodnôt v skupine Render Settings.

Rotácie sú počítané pomocou Eulerových uhlov a rotačných matíc. Takéto riešenie však kvôli javu zvanému *gimbal lock* nedovoľuje plynulo spätne získať hodnoty uhlov rotácií. Tento jav je možné obísť použitím komplexných čísel reprezentujúcich uhly s názvom Kvaternióny. Ich implementácia sa v tejto práci napriek snahe nepodarila.[2, 6, 21]

Rotácie a priblíženie modelu je možné obmedziť nastavením maximálnych a minimálnych hodnôt rotácií a priblíženia.

Vzor: Trieda dokáže vytvoriť sadu vzorov z načítaného modelu v pamäti programu. Z nastaveného rozlíšenia referenčnej snímky, kroku tvorby vzoru a nastavených minimálnych a maximálnych hodnôt rotácií okolo osí a priblíženia sú vyrenderované binárne obrazy, z ktorých sú získané lineárne indexy. Z tých je následne vytvorená sada vzorov, ktorú je možné uložiť ako súbor CMT, alebo použiť pre detekciu. Uložené sady vzorov je možné spätne načítať a použiť pre detekciu.

Detekcia: V tejto triede sa pre detekciu využívajú dva algoritmy. Prvý je hranový párovací algoritmus. Druhým algoritmom je pseudo-pyramídový hranový algoritmus. Súčasne je možné detekovať viacero objektov v snímke a vidieť výsledok detekcie.

Trieda využíva vytvorenú triedu `MyThread`, ktorá sprostredkúva prácu s vláknami. Vlákna sú využité pri zobrazovaní obrazu z kamier a pri vytváraní sady vzorov.

3.4.1 Premenné a metódy

V tomto odseku sú spomenuté skupiny privátnych premenných a metódy triedy `DetectionWindow`, a vysvetlené ich funkcie.

Premenné sa členia do nasledovných skupín, z ktorých každá skupina má svoju špecifickú funkciu.

Global: premenné z tejto skupiny majú globálne použitie. Používajú sa vo viacerých miestach programu.
Thread: tieto premenné slúžia na prácu programu vo vláknach.
Model variables: premenné obsahujúce informácie o modeli.
Template variables: premenné obsahujúce informácie o vytvorených, alebo načítaných sadách vzorov.
Creating template variables: tieto premenné sa využívajú pri vytváraní sady vzorov.
Messagebox: premenné vytvárajúce informačné okná.
Rotations: premenné nesúce informácie o rotačných maticiach.
Template: premenné zabezpečujúce prácu na karte Template.
Detection: premenné zabezpečujúce prácu na karte Detection.
ScrollArea: skupina premenných nesúca informácie o sadách vzorov zobrazených v skupine Templates na karte Detection.

Metódy triedy `DetectionWindow` a ich funkcie:

`void my_showModel(void)` - zabezpečuje zobrazenie vyrenderovaného modelu na karte Template.
`void my_create2DModelRender(void)` - vytvára render z plošného obrazu.
`void my_create3DModelRender(void)` - vytvára render z priestorového modelu.
`void my_create2DModelTemplate(void)` - vytvára binárny obraz z plošného obrazu, použité pri vytváraní vzoru.
`void my_create3DModelTemplate(void)` - vytvára binárny obraz z priestorového obrazu, použité pri vytváraní vzoru.
`void my_load2DObject(const string filePath)` - načíta snímku a vytvorí z nej model.
`void my_load3DObject(const string filePath)` - načíta súbor OBJ a vytvorí z neho model.
`void my_loadReferenceImage(void)` - otvorí okno pre výber referenčnej snímky a zvolenú snímku načíta.
`void my_loadSceneImage(void)` - otvorí okno pre výber snímky detekčenj scény a zvolenú snímku načíta.

`void my_loadReferenceVideo(void)` - otvorí okno pre výber referenčného videa a zvolené video načíta.
`void my_loadSceneVideo(void)` - otvorí okno pre výber videa detekčenj scény a zvolené video načíta.
`void my_refreshCameraList(void)` - obnoví zoznam pripojených kamier.
`void my_checkAngles(void)` - skontroluje, či sú všetky rotačné uhly v rozmedzí minimálnych a maximálnych nastavených hodnôt.
`void my_setDefaultValues(void)` - obnoví všetky nastavenia na predvolené hodnoty.
`void my_onImageResize(void)` - zabezpečuje konštantný pomer okna, v ktorom sa zobrazuje vzor a detekcia.
`void my_rotationMatrix(void)` - pripraví hodnoty rotačných matíc podľa nastavených rotačných uhlov.
`void my_createTemplateCore(void)` - je volaná metódou `my_createTemplate()`, vytvára jeden vzor ktorý je následne vložený do sady vzorov.
`void my_createTemplate(void)` - volá metódu `my_createTemplateCore()` a zabezpečuje ukladanie dát vzoru do sady vzorov.
`void my_showScene(void)` - zabezpečuje zobrazenie detekčnej scény a výsledku detekcie na karte Detection.
`void my_onTemplateChange(void)` - zabezpečuje obnovenie zoznamu načítaných sád vzorov na karte Detection.
`void my_chamferMatchingDetection(void)` - použitím hranového párovacieho algoritmu hľadá na vzdialenostnej mape najvyššiu zhodu zo všetkých sád vzorov načítaných v pamäti.
`void my_chamferMatchingDetectionPyramid(void)` - použitím pseudo-pyramídového hranového párovacieho algoritmu hľadá na vzdialenostnej mape najvyššiu zhodu zo všetkých sád vzorov načítaných v pamäti.
`void my_distanceTransform(const Mat* input, Mat* output)` - zo vstupného binárneho obrazu vytvára vzdialenostnú mapu použitím metriky mestských blokov a váhových hodnôt o hodnote 32.
`void my_checkTemplatesLimits(void)` - kontroluje maximálne rozlíšenia sady vzorov. Ak je toto rozlíšenie vyššie ako rozlíšenie detekčnej scény, sada vzorov sa nastaví ako neaktívna.
`bool my_checkCursorPosition(void)` - kontroľuje, či kurzor nie je na pozícií jednej z vertikálnych hrán obrazovky. Využitie pri skokovej zmene pozície kurzoru z jednej strany obrazovky na druhú.
`ushort my_findOptimalStep(const size_t sizeOfVector)` - hľadá optimálny krok pre pyramídovú detekciu.

`void closeEvent(QCloseEvent *event)` - zabezpečuje odpojenie kamier a ukončenie vlákien pri vypnutí okna programu.

Celý kód programu je pre lepšiu prehľadnosť a pochopiteľnosť organizovaný a upravený podľa zaužívaných pravidiel jazyku C++.[14]

3.4.2 Grafické používateľské rozhranie

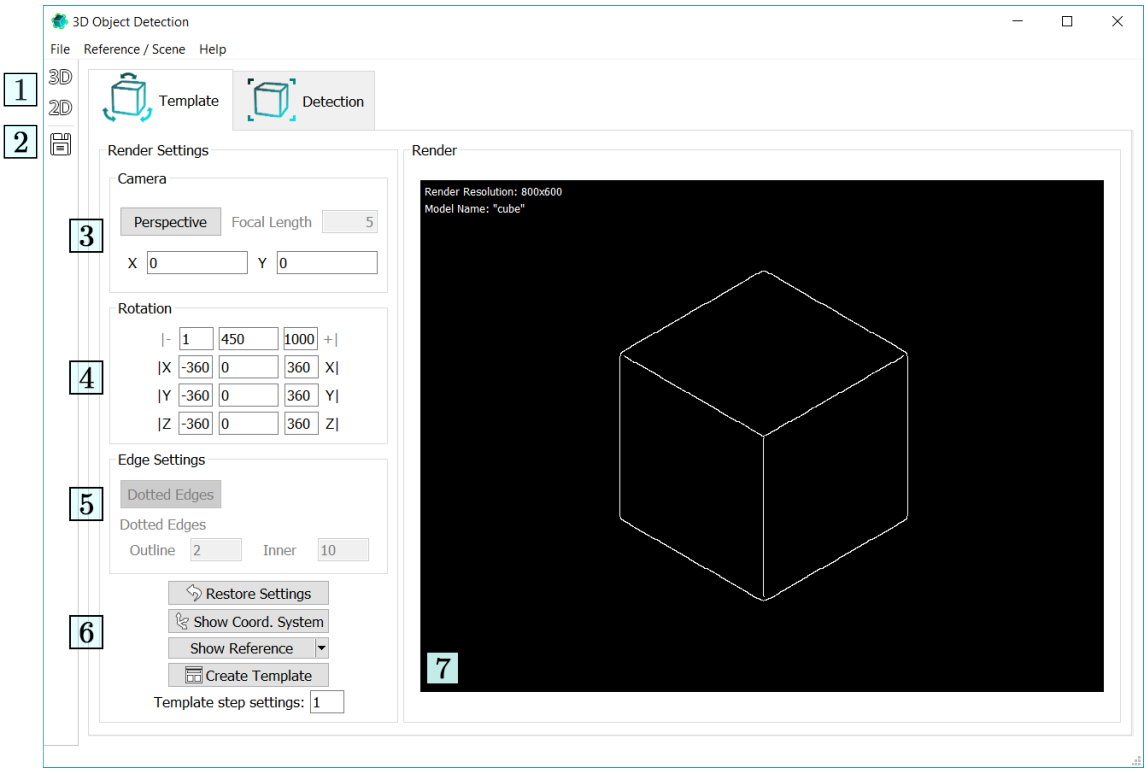
GUI sa delí na dve časti. Časť pre tvorbu vzoru (Template) a detekčnú časť (Detection). Ikony v GUI boli buď vytvorené, alebo sú súčasťou voľne dostupného balíku iconSweets.[22] Grafické rozhranie bude popísané na obrázku 3.13.

Popis karty **Template** z obrázku 3.13a:

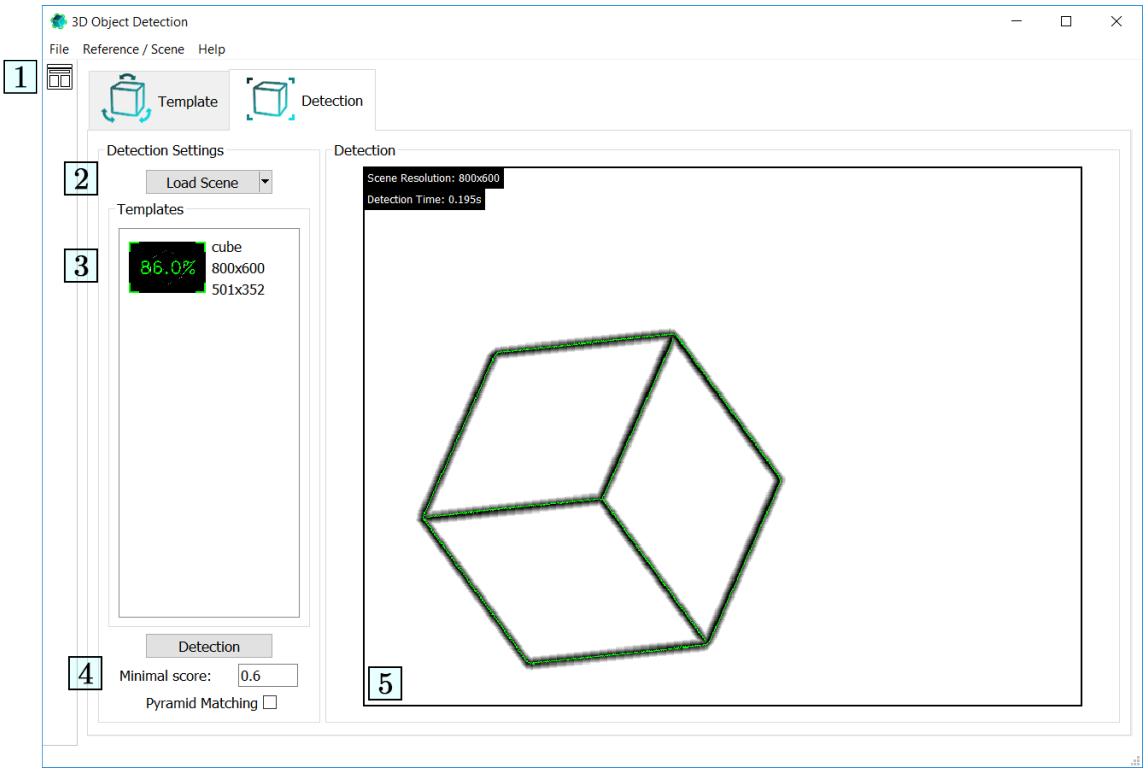
(1) tlačidlá na načítanie plošného objektu, alebo priestorového modelu,
(2) tlačidlo pre uloženie poslednej vytvorenej sady vzorov,
(3) zapnutie/vypnutie perspektívneho zobrazenia a nastavenie ohniskovej vzdialenosti kamery,
(4) nastavenie rotácií a priblíženia vzoru,
(5) zobrazenie hrán bodkovane,
(6) tlačidlá pre obnovenie nastavení, zobrazenie súradnicových osí modelu, zobrazenie referenčného obrazu a vytvorenie sady vzorov s nastavením kroku,
(7) okno určené pre zobrazenie vyrenderovaného vzoru a referenčnej snímky.

Popis karty **Detection** z obrázku 3.13b:

(1) tlačidlo pre načítanie sady vzorov z pamäte,
(2) tlačidlo pre zobrazenie a načítanie detekčnej scény,
(3) okno určené pre zobrazenie všetkých sád vzorov v pamäti programu,
(4) tlačidlo pre spustenie detekcie, nastavenie hodnoty minimálnej zhody a nastavenie metódy detekcie,
(5) okno určené pre zobrazenie detekčnej scény a výsledku detekcie.



(a) Ukážka karty Template pri vytváraní vzoru zo snímky.



(b) Zobrazenie karty Detection pri detekcií objektu.

Obr. 3.13: Ukážka tvorby vzoru a detekcie zo snímky [34].

4 ZÁVER

Cieľom tejto bakalárskej práce bolo vykonať rozbor riešení detekcie objektov, vybrať vhodné detekčné algoritmy, prakticky ich implementovať do programu a porovnať ich medzi sebou. Ďalším cieľom práce bolo vytvorenie prostredia pre detekciu objektov na snímkach a zobrazenie výsledku detekcie.

V prvom kroku riešenia tohto problému bolo potrebné definovať, aké objekty bude program schopný detekovať. Zvolené boli obsahovo nemenné plošné obrazy a tvarovo nemenné priestorové modely. Výhodou takýchto objektov je, že nie je nutné riešiť zmeny tvaru objektov. Všetko potrebné, je mať referenčný objekt za pomoci ktorého bude sprostredkovaná detekcia.

Pre zvolené typy objektov bolo potrebné vybrať vhodný algoritmus, ktorý by bol rýchly, pamäťovo nenáročný a ľahko pochopiteľný. Zvolený algoritmus hranového vzdialenostného párovania (Chamfer distance matching) spĺňal všetky tieto požiadavky. Tento algoritmus vyžaduje vytvorenie vzdialenostnej mapy a vzor objektu, ktorý chceme detekovať. Algoritmus je určený na detekciu objektov s konštantným tvarom. Je výpočtov nenáročný, nezávislý od šumu a odolný voči viacerým objektom na snímke. Medzi nevýhody tohoto algoritmu je, že nepočíta s rotáciou alebo zmenou veľkosti vzoru. Preto je potrebné jeden objekt definovať sadou vzorov, čo môže byť pamäťovo aj výpočtovo náročné.

Vzdialenostná mapa potrebná na detekciu pomocou algoritmu hranového vzdialenostného párovania, je vytváraná použitím vzdialenostnej transformácie na binárny (hranový) obrazu. V tejto práci je použitá vlastná vzdialenostná transformácia v metrike mestských blokov. Tá sa preukázala ako najrýchlejšia zo všetkých testovaných vlastných a OpenCV transformácií s časom 0,283 s na 100 transformácií binárneho obrazu s rozlíšením 1200 x 600. Použitá vzdialenostná transformácia používa pri výpočte váhu v smeroch v_x a v_y o hodnote 32. Tým je dosiahnutá vyššia presnosť detekcie a prísnejšie hodnotenie výslednej percentuálnej zhody vzoru.

Druhým potrebným komponentom pre hranový vzdialenostný algoritmus je vzor objektu, ktorý má byť na snímke detekovaný. Vzor je definovaný pozíciami hrán objektu, ktoré sú nazvané ako kľúčové body. V tejto práci je vzor definovaný ako jednorozmerný `vektor` dátového typu `int`. V tomto dynamickom poli sú zapuzdrené základné informácie o vzore, jeho rozlíšenie, rotačné uhly objektu okolo súradnicových osí, pozícia ťažiska a samotné kľúčové body.

Ak však má byť detekovaných viacero rotácií objektu, je nutné vytvoriť sadu niekoľkých vzorov. Sada vzorov je v tejto práci 5-rozmerné dynamické pole, ktorého 4 rozmery sú určené trom rotáciám okolo osí a zmene veľkosti. Piaty rozmer nesie samotné informácie jedného vzoru. Za pomoci takejto sady vzorov je možné definovať objekt rôznych natočení a priblížení. V sade vzorov je každý vzor vložený dvakrát.

Raz s redukovaným a raz s plným počtom kľúčových vzorov, čo sa využíva pri hranovom párovaní.

Pre vytvorenie vzoru je nutné získať binárny obraz objektu, ktorý chceme detekovať. Získanie takéhoto obrazu je v tejto práci riešené renderovaním plošných obrazov a priestorových modelov. Vstupom môže byť snímka na ktorej je objekt ktorý chceme detekovať alebo súbor OBJ v ktorom je objekt vymodelovaný. Zo snímky sú za pomoci adaptívneho prahovania získané hrany objektu, ktorý je na nej. Zo súboru OBJ sú získané informácie o vrcholoch a hranách modelu, ktorý je za pomoci algoritmov pre zobrazenie priestorového modelu vyrenderovaný do snímky.

Hranový párovací algoritmus v tejto práci pracuje v dvoch krokoch. V prvom kroku sa posunom vzoru po vzdialenostnej mape zisťuje minimálna hodnota hranovej vzdialenosti za použitia vzoru s zredukovaným počtom kľúčových bodov. Pri nájdení minimálnej hodnoty hranovej vzdialenosti sa spustí druhý krok, ktorý za použitia toho istého vzoru s plným počtom kľúčových bodov presnejšie určí, či v danej pozícii má vzor naozaj najmenšiu hranovú vzdialenosť. Použitie redukovaného počtu kľúčových bodov značne skracuje čas detekcie bez veľkej straty presnosti.

Samotný algoritmus hranového vzdialenostného párovania môže byť pri použití veľkých detekčných snímok a malých vzorov časovo veľmi náročný. Táto práca rieši tento problém použitím navrhnutého pseudo-pyramídového algoritmu, ktorý za pomoci redukcie počtu párovaných vzorov dosahuje zrýchlenie až 70 % - 85 % oproti pôvodnému algoritmu. Súčasne v niektorých prípadoch dokáže detekovať objekt s vyššou presnosťou. Jeho nevýhodou je, že nepočíta hranovú vzdialenosť pre všetky polohy vzorov na snímke, čo niekedy vedie k nesprávnemu výsledku detekcie.

Pre program bolo vytvorené grafické používateľské rozhranie, ktoré je spolu s algoritmom pre hranové vzdialenostné párovanie zapuzdrené v triede `DetectionWindow`. Táto trieda má implementované funkcie vytváranie vzoru z 2D snímok a z 3D modelov. Podporuje nastavenie rotácií a priblíženia pri tvorbe sady vzorov. Ďalej podporuje detekciu objektov na snímkach, videu a na obraze z kamery za použitia obyčajného hranového párovania a pseudo-pyramídového hranového párovania. Výsledky detekcie počíta vo forme percentuálnej zhody a zobrazuje ich pre každú sadu vzorov osobitne. Táto trieda bola písaná vo VS 2015 za použitia jazyka C++ a knižníc OpenCV a Qt. Tým pádom je multiplatformová a voľne šíriteľná.

Všetky stanovené ciele tejto práce boli dosiahnuté v programe s názvom `3D Object Detection`, ktorý bol vytvorený pre túto prácu a je jej výstupom.

LITERATÚRA

[1] ŽÁRA, Jiří, Bedřich BENDEŠ, Jiří SOCHOR a Petr FELKEL. *Moderní počítačová grafika.* 1998. Brno: Computer Press, 2004. 609 s. ISBN 80-251-0454-0.

[2] HAVELKA, Jan. *Image processing — efektivní algoritmy realizující „distance transform"funkci.* [online]. Brno, 2006. Bakalářská práce. Masarykova univerzita Fakulta informatiky. Vedoucí práce RNDr. Jan Hubený. [cit. 2017-05-19] Dostupné z: `https://is.muni.cz/th/60609/fi_b/bakalarka.pdf`

[3] TOMORI, Zoltán a Matej NIKOROVIČ. *Počítačové videnie v praxi* [online]. 1. vydanie. Košice: Ústav experimentálnej fyziky SAV, 2016. 140 s. [cit. 2017-05-19] Dostupné z: `http://home.saske.sk/~tomori/Downloads/Poc_videnie/PV_2016.pdf`

[4] ŠIKUDOVÁ, Elena, Zuzana ČERNEKOVÁ, Wanda BENEŠOVÁ, Zuzana HALADOVÁ a Júlia KUČEROVÁ. *Počítačové videnie: Detekcia a rozpoznávanie objektov* [online]. 1. vydanie. Praha: Wikina, 2011. 397 s. ISBN 978-80-87925-06-5. [cit. 2017-05-19] Dostupné z: `http://sccg.sk/~cernekova/Pocitacove_videnie.pdf`

[5] Katedra aplikovanej informatiky - Univerzita Komenského *Pokročilé spracovanie obrazu - cvičenia*[online]. [cit. 2017-05-19]. Dostupné z: `https://dai.fmph.uniba.sk/w/Pocitacove_videnie_1/sk`

[6] KRŠEK, Přemysl. *Základy počítačové grafiky* [online]. Verze 0.9.1. [cit. 2017-05-19]. Dostupné z: `https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIZG-IT%2Ftexts%2Fizg_opora.pdf`

[7] JURIŠICA, Ladislav, Peter HUBINSKÝ a Juraj KNOT. Detekcia a sledovanie objektov (1). *AT&P journal* [online]. 2005, 2005(6), 2 s. [cit. 2017-05-19]. Dostupné z: `http://www.atpjournal.sk/buxus/docs/atp-2005-06-69.pdf`

[8] Videnie: *Najčastejšie ochorenia oka* [online]. [cit. 2017-05-23]. Dostupné z: `http://www.videnie.sk/zrak/problemy/41-najcastejsie-ochorenia-oka`

[9] MathWorks: *Distance Transform* [online]. [cit. 2017-05-19]. Dostupné z: `https://www.mathworks.com/help/images/distance-transform.html`

[10] MathWorks: *bwdist* [online]. [cit. 2017-05-20]. Dostupné z: `https://www.mathworks.com/help/images/ref/bwdist.html`

[11] MathWorks: *Matrix Indexing* [online]. [cit. 2017-05-19]. Dostupné z: https://www.mathworks.com/help/matlab/math/matrix-indexing.html

[12] OpenCV: *About* [online]. [cit. 2017-05-20]. Dostupné z: http://opencv.org/about.html

[13] SlideServe: *Chamfer Matching & Hausdorff Distance* [online]. [cit. 2017-05-19]. Dostupné z: http://www.slideserve.com/nau/chamfer-matching-hausdorff-distance

[14] Edward Parrish: *How To Document and Organize Your C++ Code* [online]. [cit. 2017-05-19]. Dostupné z: http://www.edparrish.net/common/cppdoc.html

[15] PyImageSearch: *Multi-scale Template Matching using Python and OpenCV* [online]. [cit. 2017-05-19]. Dostupné z: http://www.pyimagesearch.com/2015/01/26/multi-scale-template-matching-using-python-opencv/

[16] Camera World: *Distance Transform and Shape Matching* [online]. [cit. 2017-05-19]. Dostupné z: https://worldofcameras.wordpress.com/2011/10/20/distance-transform-and-shape-matching/

[17] Learn OpenGL *Face culling* [online]. [cit. 2017-05-19]. Dostupné z: https://learnopengl.com/#!Advanced-OpenGL/Face-culling

[18] OpenGL Tutorial *Model loading* [online]. [cit. 2017-05-19]. Dostupné z: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/

[19] OpenGL Wiki *Calculating a Surface Normal* [online]. [cit. 2017-05-19]. Dostupné z: https://www.khronos.org/opengl/wiki/Calculating_a_Surface_Normal

[20] Brave Learn *Display Image in Qt Creator Widget Application using OpenCV* [online]. [cit. 2017-05-19]. Dostupné z: https://bravelearn.com/display-image-in-qt-creator-widget-application-using-opencv/

[21] EuclideanSpace *Maths - Quaternions* [online]. [cit. 2017-05-19]. Dostupné z: http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/index.htm

[22] iconSweets *Icon Pack* [online]. [cit. 2017-05-19]. Dostupné z: http://www.iconsweets.com/

[23] Microsoft *Integer Limits* [online]. [cit. 2017-05-21]. Dostupné z: https://msdn.microsoft.com/en-us/library/296az74e.aspx

[24] Computer vision. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation. [cit. 2017-05-23]. Dostupné z: https://en.wikipedia.org/wiki/Computer_vision

[25] Microsoft Visual Studio. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation. [cit. 2017-05-23]. Dostupné z: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

[26] 3D projection. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation. [cit. 2017-05-19]. Dostupné z: https://en.wikipedia.org/wiki/3D_projection

[27] Chebyshev distance. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation. [cit. 2017-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Taxicab_geometry

[28] Taxicab geometry. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation. [cit. 2017-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Chebyshev_distance

[29] Pyramid (image processing). *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation. [cit. 2017-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Pyramid_(image_processing)

[30] Otsu's method. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation. [cit. 2017-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Otsu%27s_method

[31] Qt (software). *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation. [cit. 2017-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Qt_(software)

[32] Lenna Söderberg. In: *ImageProcessing/VideoCodecs/Programming* [online]. [cit. 2017-05-19]. Dostupné z: http://www.hlevkin.com/TestImages/lenna.bmp

[33] Cameraman. In: *ImageProcessing/VideoCodecs/Programming* [online]. [cit. 2017-05-19]. Dostupné z: http://www.hlevkin.com/TestImages/cameraman.bmp

[34] Cube. In: *Deke* [online]. [cit. 2017-05-19]. Dostupné z: http://www.deke.com/files/images/blog-20130723-DT%20cube/07-cube.jpg

[35] Suzanne. In: [online]. [cit. 2017-05-19]. Dostupné z: http://blogorama.nerdworks.in/content/images/nerdworks/images/suzanne.png

[36] Dragon. In: [online]. [cit. 2017-05-19]. Dostupné z: http://liris.cnrs.fr/meshbenchmark/images/dragon_snopshot.jpg

[37] *Stanford Bunny OBJ Model* [online]. [cit. 2017-05-19]. Dostupné z: http://www.mrbluesummers.com/3562/downloads/stanford-bunny-model

[38] *Teapot OBJ Model* [online]. [cit. 2017-05-19]. Dostupné z: https://groups.csail.mit.edu/graphics/classes/6.837/F03/models/teapot.obj

[39] *Tetrahedron OBJ Model* [online]. [cit. 2017-05-19]. Dostupné z: http://math.purduecal.edu/~rlkraft/cs455-2009/chap7/tetrahedron.obj

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

GUI	grafické používateľské rozhranie (z angl. Graphical User Interface)
RGB	farebný model pozostávajúci z troch farebných zložiek: červená, zelená a modrá
RMS	kvadratický priemer (z angl. Root Mean Square)
VS	vývojové prostredie Visual Studio
IDE	integrované vývojové prostredie (z angl. Integrated Development Environment)
OpenCV	Open Source Computer Vision Library
OBJ	typ súboru reprezentujúci priestorovú geometriu
CMT	Chamfer template: typ súboru určený pre zapuzdrenie sady vzorov
HP	hranové párovanie
P-PHP	pseudo-pyramídové hranové párovanie
CD	kompaktný disk (z angl. compact disc)

ZOZNAM PRÍLOH

A OBSAH PRILOŽENÉHO CD

Na CD vloženom v tejto bakalárskej práci je priložený projekt triedy `DetectionWindow`, vytvorený vo VS 2015. Tento projekt bol testovaný na operačnom systéme Windows 10 (64-bit). Ďalej sú tu priložené snímky z ktorých boli vytvorené sady vzorov, modely použité pri renderovaní, vytvorené sady vzorov a scény a videá na ktorých bola testovaná detekcia.

- `/` ..... koreňový adresár priloženého CD
  - `1. Projekt VS` ..... projekt tiedy DetectionWindow vytvorený v IDE VS 2015
    - `3D_OD_GUI`
    - `Debug`
    - `x64`
    - `3D_OD_GUI.sln`
    - `3D_OD_GUI.VC.db`
  - `2. Snímky` ..... snímky použité na vytvorenie sád vzorov
    - `cube.jpg`
    - `dragon.jpg`
    - `Lenna.bmp`
    - `suzanne.jpg`
    - `vonavka.jpg`
  - `3. Modely` ..... OBJ modely použité na renderovanie
    - `bunny.obj`
    - `teapot.obj`
    - `tetrahedron.obj`
  - `4. Sady vzorov` ..... vytvorené sady vzorov použité na detekciu
    - `cube (1).cmt`
    - `cube (2).cmt`
    - `dragon.cmt`
    - `suzanne.cmt`
    - `vonavka.cmt`
  - `5. Scény` ..... scény použité na detekciu
    - `640x480 (1).png`
    - `640x480 (2).png`
    - `640x480 (3).png`
    - `800x600.png`
    - `1280x720.png`
    - `2000x1000.png`
    - `4096x2160.png`
  - `6. Video-scény` ..... video-scény použité na detekciu
    - `video1.mp4`
  - `7. Ikony` ..... ikony použité v GUI
  - `8. Elektronická verzia` ..... elektronická verzia bakalárskej práce
    - `bakalárska práca.pdf`