

VŠB – Technická univerzita Ostrava  
Fakulta elektrotechniky a informatiky  
Katedra informatiky

# **Rychlá detekce několika beztexturových 3D objektů**

## **Fast detection of multiple 3-D textureless objects**

# Zadání diplomové práce

Student: **Bc. Pavel Kutáč**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: **Rychlá detekce beztexturových 3D objektů**  
**Fast Detection of Textureless 3-D Objects**

Jazyk vypracování: čeština

## Zásady pro vypracování:

Sledování objektů je důležitou součástí počítačového vidění. Cílem práce je implementace algoritmu pro rychlou detekci 3D objektů bez textur v obraze. Hledaný objekt je definován jako sada obrazů, které vyobrazují zadaný objekt pod různými úhly pohledu (pózami objektu). Ve výsledném obraze je pak objekt označen a s přiřazenou pravděpodobností k nalezenému vzoru.

## Ve své práci proveďte:

1. Nastudujte aktuální stav poznání ohledně detekce beztexturovaných objektů.
2. Naimplementujte algoritmus [1] pro detekci objektů ve scéně. Výstupem bude též určení pózy hledaných objektů.
3. Svou implementaci řádně otestujte a zdokumentujte.
4. Proveďte závěrečné shrnutí.

## Seznam doporučené odborné literatury:


[1] Cai, H., Werner, T., Matas, J.: Fast Detection of Multiple Textureless 3-D Objects, Computer Vision Systems: 9th International Conference, ICVS 2013, pp. 103-112, 10.1007/978-3-642-39402-7\_11, (2013)

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.


Vedoucí diplomové práce: **Ing. Jan Gaura, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

  
doc. Ing. Jan Platoš, Ph.D.  
vedoucí katedry



  
prof. Ing. Pavel Brandštetter, CSc.  
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 24. dubna 2018

..........

Rád bych poděkoval panu Ing. Janu Gaurovi, Ph.D. za vedení a odborné rady při tvorbě této diplomové práce. Dále bych také rád poděkoval spolužákům Bc. Janu Šimečkovi za vzájemné konzultace našich prací a Bc. Jakubu Beránkovi za ochotné vysvětlování a tipy ke zlepšení implementace v jazyce C++.

## **Abstrakt**

Práce popisuje implementaci a postupy, díky němž je možné lokalizovat a detekovat beztexturové 3D objekty ve scéně, a následně tak určovat, o jaké objekty se jedná. Vstupem programu jsou obrazy jednotlivých trénovacích objektů, které jsou nasnímány ze všech stran a pod různými úhly. Na základě těchto hodnot probíhá fáze přípravy, která je časově velmi náročná. Po jejím dokončení jsou připravená data uložena. Následuje fáze detekce, během které jsou tato data načtena a jsou upravena tak, aby detekování probíhalo co nejrychleji. Zároveň je možné načtená data použít i opakovaně pro detekci v dalších scénách.

Implementace celé úlohy je provedena v jazyce C++ s použitím knihovny OpenCV a OpenMP. Knihovna OpenCV je volně šiřitelná a určena především pro úlohy spojené se zpracováním obrazu a matematickými operacemi nad maticemi. Pro paralelní zpracování úloh je využita knihovna OpenMP, která je již v základu součástí vývojového prostředí Visual Studio, ve kterém byla celá implementace realizována.

**Klíčová slova:** Detekce beztexturových objektů, detekce hran, 3D objekty, OpenCV, šablony, posuvné okno, obrazová pyramida

## **Abstract**

The goal of this thesis is to describe implementation and steps of detection algorithm. Purpose of the algorithm is to localize and detect textureless 3D objects in scenes and add label, which object was found. Inputs of the program are templates of objects, which are captured from all sides by different angles. The first phase is the preparation and is based on the input images. All prepared data are saved afterward, because this phase takes lots of time. The following step is the detection phase, which starts by loading prepared data. The main advantage of those prepared data is possibility to run detection phase repeatedly without negative time impact.

The implementation of the thesis is made in C++ programming language with OpenCV and OpenMP libraries. OpenCV is open-source computer vision library and contains functions and methods to process digital images and matrix. Library named OpenMP is used for parallelization of implementation and is already present in the development environment Visual Studio.

**Key Words:** Detection of textureless objects, edge detection, 3D objects, OpenCV, templates, sliding window, image pyramid

# Obsah

Seznam použitých zkratk a symbolů	8
Seznam obrázků	9
Seznam tabulek	10
Seznam výpisů zdrojového kódu	11
<b>1 Úvod</b>	<b>12</b>
<b>2 Používané techniky pro detekci a rozpoznávání v obrazech</b>	<b>13</b>
2.1 Použití příznaků a rozdíl mezi detekcí a rozpoznáním . . . . .	13
2.2 Posuvné okno a obrazová pyramida . . . . .	14
2.3 Konvoluce . . . . .	15
2.4 Gaussovo vyhlazení . . . . .	17
2.5 Detekce a orientace hran . . . . .	18
2.6 Distanční mapa . . . . .	20
2.7 Jaccardův index . . . . .	21
<b>3 Fáze přípravy šablon</b>	<b>23</b>
3.1 Požadavky na formát šablon . . . . .	23
3.2 Načtení šablon a vytvoření pomocných obrazů . . . . .	24
3.3 Problém stínů při detekci hran . . . . .	25
3.4 Vytvoření pravidelné mřížky a generování tripletů . . . . .	26
3.5 Výpočet vzdálenosti a orientace hrany . . . . .	26
3.6 Naplnění hashovací tabulky . . . . .	28
3.7 Orientované zkosení hran . . . . .	32
3.8 Mazání hran podle stability pohledu a jejich orientace . . . . .	34
3.9 Ukládání připravených dat . . . . .	36
<b>4 Detekce objektů ve scéně</b>	<b>38</b>
4.1 Načítání předem připravených dat . . . . .	38
4.2 Získání kandidátů pomocí obrazové pyramidy a posuvného okna . . . . .	39
4.3 Non-maxima suppression . . . . .	41
4.4 Výpočet F1 skóre na základě ground truth dat . . . . .	44
<b>5 Výsledná úspěšnost detektoru a další vylepšení implementace</b>	<b>46</b>
<b>6 Časové náročnosti operací</b>	<b>51</b>
<b>7 Použité hodnoty</b>	<b>54</b>

**8 Závěr** 55

**Literatura** 57

## Seznam použitých zkratek a symbolů

API	– Application Programming Interface
CCD	– Charge–Coupled Device, druh snímacího čipu v kamerách a fotoaparátech
CMOS	– Complementary Metal–Oxide–Semiconductor, druh snímacího čipu v kamerách a fotoaparátech
CNN	– Convolutional neural networks
GT	– Ground Truth
IoU	– Intersect over Union
OpenCV	– Open Source Computer Vision Library
RGB	– Barevný model <i>Red, Green, Blue</i>
ROI	– Region Of Interest
SIFT	– Scale–invariant feature transform
XML	– Extensible Markup Language
YAML	– YAML Ain't Markup Language, strukturovaný datový formát



## Seznam obrázků

1	Ilustrace metody posuvného okna . . . . .	14
2	Znázornění jednotlivých stupňů obrazové pyramidy . . . . .	15
3	Ilustrace výpočtu konvoluce . . . . .	17
4	Znázornění Sobelova operátoru a Cannyho detektoru hran . . . . .	20
5	Znázornění distanční transformace . . . . .	21
6	Vizualizace výpočtu Jaccardova koeficientu podobnosti dvou oblastí . . . . .	22
7	Sjednocení datových struktury reprezentující obdélník . . . . .	22
8	Detekční jednotka . . . . .	24
9	Znázornění detekovaných hran před a po prahování . . . . .	25
10	Znázornění pravidelné mřížky a vygenerovaných tripletů . . . . .	26
11	Vizualizace tripletu s konkrétními hodnotami vzdáleností a orientací . . . . .	28
12	Znázornění ukládání šablon do hashovací tabulky . . . . .	29
13	Rozložení vzdáleností a orientací do binů . . . . .	32
14	Vizualizace porovnání dvou šablon a výpočet orientovaného zkosení hran . . . . .	34
15	Vstupní šablona a dalších 7 nejpodobnějších šablon . . . . .	35
16	Ukázka odstranění hran pomocí stability pohledu a orientace hran . . . . .	36
17	Ukázka testovacích scén . . . . .	38
18	Znázornění metody non-maxima suppression . . . . .	42
19	Znázornění evaluace binárního klasifikátoru . . . . .	45
20	Úspěšnost detektoru v jednotlivých scénách . . . . .	46
21	Znázornění hran pouze na okrajích obrazu, které jsou přesto validní kandidáti . . . . .	47
22	Ukázka několika extrémních případů, kdy hrany šablon nezasahují do všech čtyř kvadrantů obrazu . . . . .	47
23	Znázornění scén . . . . .	49
23	Znázornění scén . . . . .	50

## Seznam tabulek

1	Velikost scén v jednotlivých stupních obrazové pyramidy . . . . .	41
2	Porovnání úspěšnosti detektoru a rychlosti detektoru v různých modifikacích algoritmu . . . . .	48
3	Náročnost operací přípravné fáze . . . . .	51
4	Časová náročnost načítání dat ve fázi detekce . . . . .	52
5	Náročnost jednotlivých stupňů obrazové pyramidy ve fázi detekce . . . . .	53
6	Náročnost jednotlivých stupňů obrazové pyramidy ve fázi detekce po dodatečných vylepšeních . . . . .	53
7	Seznam použitých konstant a jejich výchozí hodnoty . . . . .	54

## Seznam výpisů zdrojového kódu

1	Získání vzdálenosti a orientace nejbližší hrany od zadaného bodu . . . . .	27
2	Ukázka hashovací funkce . . . . .	30
3	Implementace non maxima suppression . . . . .	43

# 1 Úvod

Výkon osobních počítačů i dalších zařízení, která používáme v každodenním životě, neustále roste. Proto se do popředí zájmu dostávají i výpočetně náročnější techniky, jako je analýza statických obrazů a videí, které dříve nebylo možné z důvodu technických omezení vůbec provádět. Velká pozornost se dnes například upíná k budoucnosti autonomních vozidel, která jezdí po silnicích prakticky bez nutnosti zásahu člověka. Právě v tomto případě má detekce objektů zvláště důležitou roli. Automobily se totiž na základě analýzy a identifikace překážek rozhodují, jak na ně zareagují. Detekce objektů sice není zdaleka jedinou vstupní informací, se kterou autonomní vozidla při rozhodování operují. Často je to ale ten nejdůležitější vstup, jehož chybná interpretace může vést v krajním případě až ke ztrátám na lidských životech. Proto musí být detekce co nejpřesnější. V zábavním průmyslu se můžeme s detekcí objektů setkat například u tzv. brýlí s rozšířenou realitou, které vykreslují aktuální svět obohacen o počítačovou grafiku. Pro správné vykreslení a dosažení co nejrealističtějšího zážitku slouží detekce 3D objektů a jejich postavení. V této práci jsou popsány techniky pro detekci beztexturových 3D objektů na základě vstupních šablon, které obsahují nasnímané obrazy známých objektů ze všech stran a pod různými úhly.

V předkládané práci jsou podrobně popsány veškeré kroky detekce a požadavky na jednotlivé obrazy objektů (neboli šablon), jejich načítání a příprava do potřebného formátu pro další zpracování. Před spuštěním samotné detekce probíhá vygenerování seznamu tzv. tripletů, neboli trojic bodů. Způsob výběru a metoda jejich generování může ve velké míře ovlivnit další navazující procesy a jejich úspěšnost. Dalším krokem přípravy šablon je vytvoření hashovací tabulky, které bude v práci věnována celá podsekcce. Chybné použití tabulky může celý proces detekce značně zpomalit. Aby byl průběh fáze detekce co nejrychlejší, je nutné, aby tabulka obsahovala co nejvíce unikátních klíčů, protože tak bude výběr možných kandidátů nejužší. K tomu se úzce váže i nutnost kvalitního návrhu přidružených výpočtů. Závěrečnou částí fázi přípravy je odstranění některých hran, které nemají dostatečně vysokou vypovídací hodnotu o objektu. Tyto hrany je nutné nejdříve správně detekovat. V práci jsou popsány dvě techniky, které lze provádět sériově za sebou a dostat tak ideální filtraci dat. Odstranění hran také přináší benefit v podobě znatelného zrychlení fáze detekce. Fáze přípravy může trvat až stovky vteřin. Z tohoto důvodu je zakončena uložením připravených dat do binárního souboru.

Fáze detekce je iniciována načtením a přípravou dat ze souboru, který byl dříve získán dokončením přípravné fáze. Poté je možné spouštět detekci objektů opakovaně v různých scénách. V dalším kroku programu je načtena scéna s objekty, ve které bude probíhat hledání předem natrénovaných objektů. Detekce je řešena pomocí metody posuvného okna, které je v několika krocích postupně zvětšováno. Při každém zastavení jsou na dané pozici detekovány hrany a podobně jako ve fázi přípravy jsou také získávány hodnoty v bodech tripletů. Z těchto hodnot jsou následně vybíráni kandidáti z hashovací tabulky. V případě, že je získáno více kandidátů na stejné pozici, probíhá hlasování na základě skóre orientovaného zkosení hran. Nakonec jsou v posledním kroku pomocí metody non-maxima suppression odfiltrováni slabší kandidáti.

## 2 Používané techniky pro detekci a rozpoznávání v obrazech

V následující sekci je popsáno několik technik, které se přímo či nepřímo používají pro detekci nebo rozpoznávání v obrazech. V mnoha případech se navíc používá i více zmíněných technik zároveň, a to v různých fázích detekce i opakovaně. Několik z níže popsaných metod je také použito v této práci.

### 2.1 Použití příznaků a rozdíl mezi detekcí a rozpoznáním

Na začátek je nutné objasnit dva pojmy, které se často zaměňují, nebo je jim dokonce připisován shodný význam. Těmi pojmy jsou detekce a rozpoznávání. K záměně výrazů často dochází proto, že v některých případech není jasná hranice, kdy se ještě jedná o detekci a kdy už rozpoznávání. Pro zjednodušení si lze pomoci otázkou „Kde se *tento* objekt nebo objekty nacházejí v obraze?“. Vstupními informacemi pro detekci jsou hledané objekty a scéna, ve které probíhá hledání. Odpovědí, neboli výstupem, jsou jednotlivé pozice objektů. Tento postup se označuje jako detekce [1].

K zodpovězení otázky „Jaký druh objektu je přítomen na obrázku?“ je už nutné použít rozpoznávání [2]. Vstupem pro algoritmy rozpoznávání mohou být kromě scény obsahující objekty i detekované pozice a ohraničená část, ve které se objekt nachází. Výstupem jsou u jednotlivých pozic štítky obsahující podrobnější informace o daném objektu. V dnešní době je jádrem většiny algoritmů pro rozpoznávání neuronová síť nebo jiná technika strojového učení.

Rozdíly lze snadno ilustrovat i na příkladech:

1. Nalezení všech obličejů, určení jejich pózy a velikosti v obraze je detekce. Pro zjištění toho, které osobě obličej patří, je-li obličej muže či ženy, nebo pro určení odhadovaného věku, je již potřeba technologie rozpoznávání.
2. Zjištění pózy a ohraničení všech psů a koček v obraze je detekce. Rozpoznáním je pak určení jejich rasy.
3. Detekcí je chápáno nalezení všech dopravních značek při jízdě automobilem. Rozpoznávání slouží k přesnému určení významu značky. V případě značky „STOP“, která má jako jediná svůj specifický tvar, může dojít k určení této značky již pouhou detekcí.

Pro obě výše zmíněné metody, ale i mnoho dalších, lze používat jako výchozí bod příznaky [3] [4](angl. features). Neexistuje přesná definice toho, co příznak je a co ne, a často to přímo odvisí od použité techniky. Mohou to být detekované rohy, hrany, body, nebo také celé oblasti v obraze. Proto se jako příznak označuje cokoli, co je v obraze významné pro následné zpracování. Často je kvalita algoritmu pro detekci či rozpoznání dána právě kvalitou detektoru příznaku, protože další operace probíhají buď nad hodnotami vrácenými tímto detektorem, nebo jsou omezeny pouze na zájmové oblasti. Vzhledem k tomu, že šum v obraze může negativně ovlivnit hledání

příznaků, je prvním krokem téměř všech příznakových detektorů slabé rozmazání vstupního obrazu Gaussovým filtrem, který je podrobněji vysvětlen v podsececi 2.4.

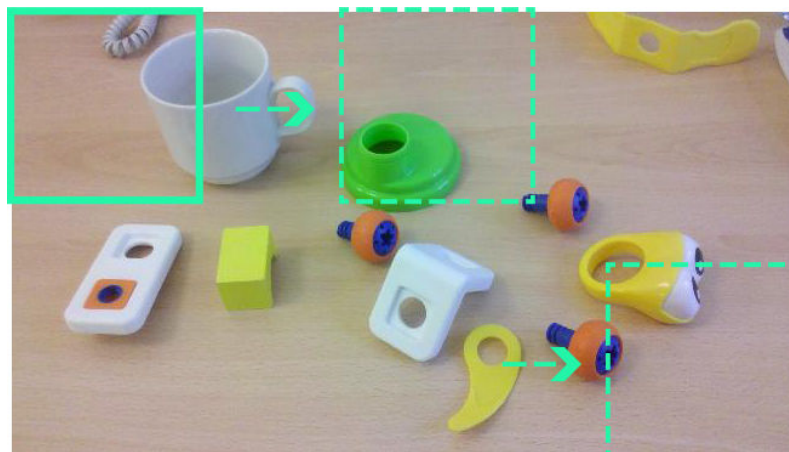
Příklady příznakových typů a možných detektorů:

1. **Hranové příznaky** - Sobelův operátor (podsecke 2.5.1), Cannyho detektor hran (podsecke 2.5.2).
2. **Bodové příznaky** - Harrisův detektor rohů [5], SIFT [6].

## 2.2 Posuvné okno a obrazová pyramida

Vyhledávání objektů ve scéně není pro stroj zdaleka tak jednoduché, jako pro člověka. Objekt se ve scéně může nacházet v jiné velikosti nebo natočení než v šabloně. Navíc může být umístěn kdekoli v obraze. Existují dvě techniky, které se často používají současně. Jedná se o metodu posuvného okna (angl. sliding window) a obrazové pyramidy (angl. image pyramid).

**Metoda posuvného okna** spočívá v určení čtverce nebo obdélníku, který je následně posouván o určitý krok definovaný pixely po obraze. Po každém tomto posunutí je provedena potřebná operace nad daným výřezem scény (angl. region of interest, zkráceně ROI). Nejčastěji začíná posuvné okno v levém horním rohu a pokračuje ve vodorovném směru doprava. Po dosažení pravé hrany obrazu se přesune zpět k levé hraně a o zvolenou vzdálenost se posune směrem dolů. Tento proces se opakuje, dokud posuvné okno nedosáhne pravého spodního rohu scény. Výsledky výpočtu z jednotlivých ROI bývají také ukládány pro další zpracování. Znázornění algoritmu posuvného okna je na obrázku 1.



Obrázek 1: Ilustrace metody posuvného okna

**Obrazová pyramida** je metoda, při které se původní scéna zmenšuje v několika krocích dle určeného měřítká. Ve výjimečných případech může také docházet ke zvětšování scény. Pyramida je používána v mnoha různých algoritmech. V případě spojení s metodou posuvného okna

je nejčastěji využita k nalezení objektů různých velikostí. Po každém zpracování scény posuvným oknem je obraz zmenšen a algoritmus spuštěn znova. V tomto konkrétním případě nemusí docházet ke zmenšování scény, ale ke zvětšování posuvného okna.

V mnoha algoritmech, kde se obrazová pyramida používá, nedochází pouze ke zmenšování v jednotlivých krocích, ale také k rozmazání scény [7]. Poté se využívá konvolučního operátoru, který je blíže popsán v podsekcí 2.3. Podle použité techniky se označuje jako Gaussova pyramida nebo Laplaceova pyramida. Při užití konvoluce je ale vhodné, aby bylo zmenšení v každém kroku celočíselné. Nejčastěji jsou zmenšení v jednotlivých stupních vůči původnímu obrazu mocniny dvou, jak je znázorněno na obrázku 2. Techniky obrazové pyramidy je využito například v algoritmu SIFT [6] nebo v konvolučních neuronových sítích, zkráceně CNN [8].



Obrázek 2: Znárodnění jednotlivých stupňů obrazové pyramidy. Levý obraz je v původní velikosti, každý následující je zmenšen  $2\times$  oproti předchozímu. Velikost posledního je  $\frac{1}{8}$  velikosti původního obrazu

### 2.3 Konvoluce

V analýze obrazu se často používají výrazy jako je maska, konvoluční matice, nebo jádro [9]. Ve všech případech se jedná o různé názvy pro malé matice, které se aplikují na zpracovávaný obraz pomocí konvoluce, neboli konvolučního operátoru, což je matematická operace nad dvěma funkcemi. Tato operace má široké uplatnění a již byla zmíněna v podsekcí 2.2. Tato technika je využita také v následujících sekcích při implementaci detekce.

V matematice je operátor konvoluce definován pomocí integrálu. Vzhledem k tomu, že jsou v analýze obrazu rozměry scény známé a konečné, může být integrál nahrazen operací sumy. Princip výpočtu je znázorněn ve výpočtu (1) a na obrázku 3, kde  $I(x, y)$  je hodnota zdrojového a  $R(x, y)$  hodnota cílového obrazu na souřadnicích  $x$  a  $y$ . Proměnné  $K_w$  a  $K_h$  určují šířku a výšku konvoluční masky.

Hodnota  $k_n$  provede normalizaci hodnot, pokud by byl součet hodnot v jádře různý od 1. V tomto případě by hodnoty pixelů při dosazení do cílového obrazu nemusely být v povoleném

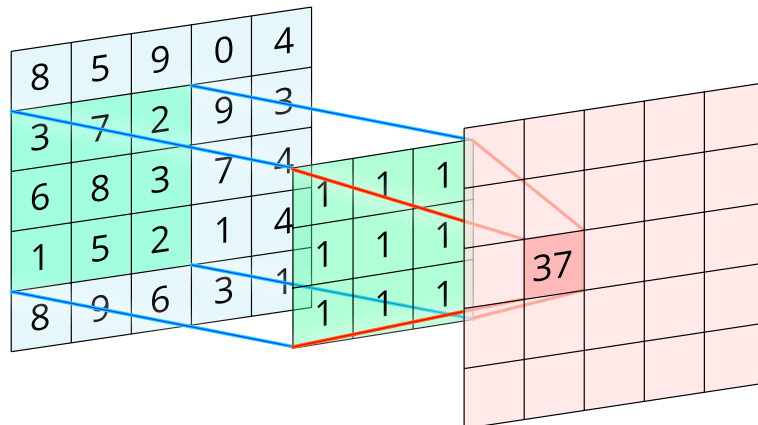
rozsahu. Pokud bude výsledná hodnota konvoluce nadále zpracována ve výpočtech, je v některých případech možné tuto normalizaci vynechat.

$$\begin{aligned}
 k_x &= \frac{K_w - 1}{2} \\
 k_y &= \frac{K_h - 1}{2} \\
 k_n &= \begin{cases} \sum_{i=0}^{K_w} \sum_{j=0}^{K_h} K(i, j) & \text{pokud výsledek součtu} \neq 0 \\ 1 & \text{pokud výsledek součtu} = 0 \end{cases} \\
 R(x, y) &= \frac{1}{k_n} \sum_{i=0}^{K_w} \sum_{j=0}^{K_h} I(x + i - k_x, y + j - k_y) K(i, j) \quad (1)
 \end{aligned}$$

Pro správnou funkčnost výpočtu by měly být hodnoty  $K_w$  i  $K_h$  vždy liché. Přesto, že jsou nejčastěji používané masky čtvercové, nemusí se šířka a výška shodovat a maska může být i obdélníkového tvaru. Při výpočtu (1) na obrazech nastává problém na hranách a v rozích. V těchto krajních situacích maska zasahuje mimo zdrojový obraz a nemůže tak vyčíst všechny hodnoty. Existuje celkem pět možností, jak tuto překážku řešit:

1. **Rozšíření** obrazu spočívá v kopírování pixelů na hranách nebo v rohu tak daleko, jak je nutné pro potřebu vypočítání hodnoty konvoluce.
2. **Zabalením** obrazu, tedy vytvořením virtuálně nekonečného obrazu. V praxi se použijí hodnoty pixelů z opačné hrany či rohu a pokračuje se tak hluboko do obrazu, kam sahá maska.
3. **Zrcadlením** hrany či rohu lze také virtuálně vytvořit nekonečný obraz. Hodnoty jsou ale brány ze stejné hrany či rohu, pouze v opačném pořadí.
4. Metoda **oříznutí masky** ignoruje tu část, která se nachází mimo zdrojový obraz. V tomto případě je nutné dbát na aktualizaci normalizační proměnné  $k_n$  v případě, že se bude aplikovat.
5. **Oříznutí obrazu** je implementačně nejjednodušší, ale také nejméně vhodnou možností. V případě, že nějaká část masky zasahuje mimo zdrojový obraz, se pixely ve výsledném obrazu ignorují. Ignorované pixely jsou pak vyplněny předem definovanou barvou, nebo je nutné výsledný obraz oříznout.





Obrázek 3: Ilustrace výpočtu konvoluce pro daný bod výstupního obrazu. Pro normalizaci výsledku by bylo nutné jej vynásobit  $\frac{1}{9}$

## 2.4 Gaussovo vyhlazení

Obrazový šum jsou náhodné změny jasu jednotlivých pixelů způsobené nedokonalostí CMOS nebo CCD snímacích čipů fotoaparátů a kamer. Ve větší či menší míře jsou obsaženy v každém obrazu, který není uměle vygenerován. Pro lidské oko většinou není šum v obrazech podstatný a přesto člověk dokáže rozpoznat, co se na obrazu nachází. Naopak při analýze obrazu může být šum příčinou chybných výsledků. Existují různé velmi pokročilé metody věnující se redukci šumu při co nejlepším zachování všech obrazových vlastností [10]. Pro základní operace jsou ale dostačující jednodušší metody jako je Gaussovo rozmazání [11], které spadá do oblasti nízkopásmových filtrů (angl. low-pass filtr). Principem je odfiltrování vysokých frekvencí, což má za negativní následek ztrátu detailů. Pozitivním efektem je ale odstranění šumu.

Gaussovo rozmazání lze implementovat pomocí konvoluce, popsané v podsekcí 2.3, s jednoduchou maskou (2). Sílu neboli intenzitu rozmazání lze regulovat velikostí jádra. Znázorněny jsou masky  $G_{3 \times 3}$  a  $G_{5 \times 5}$ , je ale možné využívat i větší. Jednotlivá jádra pro konvoluci jsou definována na základě normálního rozdělení Gaussovy funkce. Existují i jiné metody vyhlazování, pro ukázkou je zmíněn například box blur, jehož konvoluční maska je znázorněna jako  $B_{3 \times 3}$ . Gaussovo vyhlazení ale ponechává více informací o hranách a je proto vhodnější pro následnou detekci hran. Ve vzorcích (2) jsou znázorněny také koeficienty, kterými jsou celé matice vynásobeny. To odpovídá normalizaci výsledné hodnoty popsané v podsekcí 2.3. Pro implementaci s celočíselnými hodnotami je ale možné vynásobit až výslednou hodnotu po dokončení operace konvoluce.

$$G_{3 \times 3} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad G_{5 \times 5} = \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \quad B_{3 \times 3} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (2)$$

## 2.5 Detekce a orientace hran

Detekce či zjištění orientace hran je základní technika používaná v mnoha algoritmech analýzy obrazu a je základem celé techniky popsané v následujících sekcích. Jedná se o techniku kombinující různé matematické výpočty za účelem identifikace prudké změny jasů nebo chyb v návaznostech jasů. Takto detekované body tvoří linie, které mohou být označeny za hranu.

Existuje mnoho metod pro detekci hran či zjištění jejich orientace, většina z nich je ale velmi citlivá na přítomný šum. Proto je vhodné vždy obraz lehce rozmazat Gaussovým filtrem, který je popsán v podsekcí 2.4. Pro detekci hran není zapotřebí RGB barev v obraze. Z tohoto důvodu je vstupní obraz převeden do stupňů šedi. Kromě níže popsaných metod, lze zmínit například operátor Prewittové, Robertsův operátor, nebo Kirschův operátor [12].

### 2.5.1 Sobelův operátor

Sobelův operátor [13], též nazývaný Sobelův filtr nebo Sobelova maska, je dvojice matic o velikosti  $3 \times 3$ , kterou lze aplikovat pomocí konvoluce na zpracovávaný obraz. Operátor provádí aproximaci první derivace a výsledkem je obrazový gradient, neboli orientovaná změna intenzity jasu ve všech bodech obrazu. Výsledná vizualizace je na obrázku 4b).

Matice jsou znázorněny v rovnici (3), kde operátor  $*$  provádí konvoluci nad obrazem  $I$ , jak bylo popsáno výše v podsekcí 2.3. Výsledkem v  $S_x$  je horizontální a v  $S_y$  vertikální aproximovaná derivace obrazu. Pomocí výpočtu (4) lze získat obraz  $S$  s gradienty hran, který je znázorněn na obrázku 4b). V případě že se konvoluce nebude provádět nad celým obrazem, ale pouze nad jeho malou oblastí, která je ideálně shodně velká jako jádro, lze vypočítat i konkrétní orientaci gradientu  $\phi$ .

$$S_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * I \quad S_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I \quad (3)$$

$$S = \sqrt{S_x^2 + S_y^2}$$

$$\phi = \arctan \frac{S_y}{S_x} \quad (4)$$

### 2.5.2 Cannyho detektor hran

Jedním z nejznámějších a nepoužívanějších hranových detektorů je Cannyho detektor hran [14]. Obsahuje několik postupných kroků a, na rozdíl od dříve zmíněného Sobelova filtru, vrací binární obraz. Neobsahuje tedy gradienty hran, ale pouze hrany samotné. Výsledek je na obrázku 4c znázorněn v invertované podobě. Původně jsou totiž hrany zaznačeny bílou barvou (hodnota 1) na černém pozadí (hodnota 0).

Cannyho detektor dodržuje několik pravidel:

1. Detekce má malou chybovost, tj. je přesně zachyceno co možná nejvíce hran.
2. Detekovaný hranový bod by měl ležet přesně uprostřed hrany.
3. Každá hrana v obraze by měla být zaznačena pouze jednou a obrazový šum by neměl generovat falešné hrany.

V porovnání s ostatními metodami pro detekci hran je Cannyho detektor jeden z nejstriktněji definovaných algoritmů. Přesto jsou výsledky jeho detekce velmi kvalitní a spolehlivé [15, pp. 79–82]. Celý proces se skládá z pěti po sobě jdoucích kroků:

**Vyhlazení obrazu** Gaussovým rozmazáním je prvním krokem celého algoritmu. Vzhledem k tomu, že stejně jako všechny ostatní detektory mohou být lehce ovlivněny obrazovým šumem, provede se prvotní vyhlazení, které se snaží tomuto nežádoucímu jevu zamezit. V případě silného rozostření může dojít k opačnému problému, kdy nebudou některé hrany detekovány vůbec. Proto by neměla velikost jádra pro Gaussovo vyhlazení ve většině případů přesáhnout velikost  $5 \times 5$ .

Přesto, že je tento krok velmi důležitý pro správnou funkci, je v některých knihovnách, jako např. OpenCV, vynechán. Proto je důležité dbát při implementaci na rozmazání před předáním obrazu do funkce obsahující Cannyho detektor.

**Nalezení gradientu** v celém obrazu je prvotní krok k detekování hran. Jedná se o shodný postup jako v předcházející podsekcí 2.5.1. Konkrétní použití operátoru závisí na implementaci a kromě Sobelova operátoru, který je použit v implementaci OpenCV, se také může použít operátor Prewittové a další.

**Non-maxima suppression** metody je použito ke ztenčení hran. Třetí výše zmíněné pravidlo říká, že ve výsledné detekci by měla být každá hrana pouze jednou. Jak je ale viditelné na obrázku 4b), při výstupu z funkce vracející obrazový gradient jsou některé hrany velmi tlusté

a neostré. Algoritmus non-maxima suppression ponechá pouze hrany v lokálním maximu, ostatní hrany potlačí a vymaže.

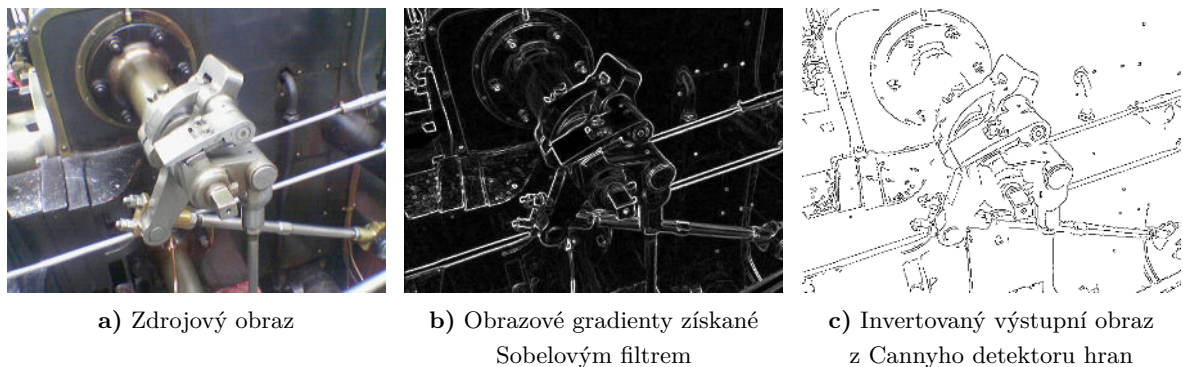
Algoritmus v každém bodě hrany zkontroluje další dva body, jeden v kladném směru gradientu a jeden v záporném směru. Pokud je aktuální bod silnější než dva porovnávané, je zachován, v opačném případě je potlačen. Určení lokálního maxima je možno pro různě natočené hrany zpřesnit pomocí interpolace mezi body.

**Dvojité práhování** slouží k přesnějšímu určení toho, zda se hrana na pozici nachází či nikoliv. Po provedení předchozího kroku jsou již hrany ztenčeny, ale stále mohou být přítomny i ty, které byly způsobeny neodfiltrovaným šumem. Metoda dvojitého práhování používá dva prahy na jejichž základě provádí pro všechny hrany jednoduché hlasování.

Pokud je hodnota pixelu hrany nižší, než spodní práh, je hrana v tomto bodě odstraněna. V případě, že je hodnota pixelu vyšší než horní práh, je pixel označen za tzv. silný pixel. V opačném případě je označen za slabý pixel.

**Sledování hran** je poslední částí algoritmu, která rozhodne o těch pixelech, které byly v předchozím kroku označeny jako slabé. V tomto okamžiku jsou už silné pixely součástí výsledného binárního obrazu a jsou zaneseny jako hrany.

Předpokládá se, že hrany v obraze mají určitou návaznost, a pixely označené jako slabé mohou pocházet jak z opravdové hrany, tak i ze zbytkového šumu. Proto se pro každý pixel označený jako slabý hledá v okolí  $3 \times 3$  alespoň jeden silný pixel. V případě, že je nalezen, je i tento slabý pixel označen za silný pixel a ponechán. Všechny ostatní slabé pixely jsou následně smazány.



Obrázek 4: Znázornění Sobelova operátoru a Cannyho detektoru hran

## 2.6 Distanční mapa

K určení vzdálenosti libovolného bodu v obraze od nejbližší hrany lze použít distanční mapu [16], někdy také nazývanou jako tzv. distanční transformace. Distanční mapa je matice shodně velká jako vstupní obraz. Na výstupu jsou do této matice zapsány vzdálenosti od nejbližší hrany, které

mohou být měřeny libovolnou metrikou. Nejčastěji se používá Euklidovská vzdálenost, která je znázorněna na obrázku 5 a 8c). Použita ale může být i Manhattanská metrika, nebo jakákoliv jiná, nebo dokonce i metrika vlastní.

Algoritmus na vstupu požaduje binární obraz s detekovanými hranami. V případě použití Cannyho detektoru hran, jak je popsáno v podsekcí 2.5.2, je nutné výstup invertovat. V invertovaném obraze je na hranách hodnota 0, což zároveň odpovídá nulové vzdálenosti od hrany. Existuje několik postupů jak distanční transformaci vypočítat. Modernější algoritmy již zvládají celý proces provést v lineárním čase. Jedním z příkladů je třeba Meijsterův algoritmus [17].

Používá se i modifikace, ve které může výsledná vzdálenost nabývat i záporných hodnot. V tomto případě je ale nutné mít ve vstupním obraze plné obrysy objektů. Při výpočtu se porovnává, zda je bod uvnitř či vně objektu, a podle toho je následně rozhodnuto o tom, zda bude znaménko kladné či záporné.

4,24	3,61	3,0	2,0	1,0	0	1,0	2,0	3,0
3,61	2,83	2,24	2,0	1,0	0	1,0	2,0	2,83
2,83	2,24	1,41	1,0	1,0	0	1,0	1,41	2,24
2,24	1,41	1,0	0	0	0	0	1,0	1,41
1,41	1,0	0	1,0	1,0	1,0	1,0	0	1,0
1,0	0	1,0	1,41	2,0	2,0	1,0	0	1,0
0	1,0	1,41	2,24	2,83	2,0	1,0	0	1,0

Obrázek 5: Znázornění distanční transformace. Černě jsou zaznačeny hrany a číslice určují vzdálenosti od hran vypočtené pomocí Euklidovské metriky

## 2.7 Jaccardův index

Jaccardův index, také známý jako Jaccardův koeficient podobnosti nebo anglicky Intersect over Union (zkráceně IoU), je statistická metoda používaná k porovnání dvou množin [18]. Je definována vzorcem (5). Zde  $J(A, B)$  značí Jaccardův index mezi množinami  $A$  a  $B$  a slouží k získání podobnosti mezi dvěma množinami, kterou lze následně převést na procentuální podobnost.

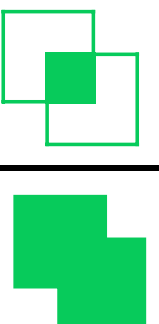
Definice je natolik obecná, že je možné tuto techniku použít v širokém spektru aplikací. Při různých modifikacích lze například hledat podobnosti mezi slovy. Existuje také pojem Jaccardova vzdálenost, která měří rozdílnost mezi dvěma množinami. Ta je ve vzorci (5) označena jako  $d_J(A, B)$ . V extrémním případě, kdy jsou množiny  $A$  i  $B$  prázdné, by ve vzorci vycházelo dělení nulou. Pro tuto situaci je definováno  $J(A, B) = 1$ .

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

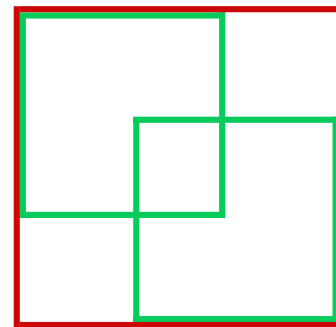
$$d_J(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \quad (5)$$

V analýze obrazu se Jaccardův index nejčastěji využívá pro určení úspěšnosti detektorů [19]. Výsledným výstupem detektorů jsou tzv. bounding boxy, neboli ohraničené oblasti, ve kterých leží detekovaný objekt. Aby bylo možné zjistit úspěšnost detekce, je potřeba ručně vymezit oblast, kde objekt opravdu leží. Takovým datům se říká ground truth, zkráceně GT. Následně se vypočítá koeficient podobnosti pomocí (5), kde  $A$  a  $B$  v tomto případě značí detekovanou oblast a oblast GT. Výsledkem operace  $A \cap B$  je průnik obdélníků a  $A \cup B$  sjednocení obdélníků tak, jak je znázorněno na obrázku 6.

Menší problém ale nastává při sjednocení datových struktur reprezentujících obdélník. Při této operaci bývá opět navrácen obdélník, což je ale v rozporu s reálným výsledkem sjednocení, kterým může být i polygon. To lze vidět ve jmenovateli zlomku na obrázku 6. K tomuto problému dochází rovněž při použití knihovny OpenCV. Výsledek takové operace sjednocení je znázorněn na obrázku 7. V implementacích je potřeba použít druhé možnosti ze vzorce (5). Zde je operace sjednocení nahrazena součtem velikostí obou množin a odečtena velikost průniku, který je jinak započítán dvakrát.

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$


Obrázek 6: Vizualizace výpočtu Jaccardova koeficientu podobnosti dvou oblastí



Obrázek 7: Sjednocení datových struktur reprezentujících obdélník může vygenerovat znovu pouze obdélník

### 3 Fáze přípravy šablon

Je nutné mít šablony, neboli snímky známých objektů, které se budou ve scéně vyhledávat, k dispozici v jednotném formátu a připravené pro co nejrychlejší následné vyhledávání. Samotná fáze přípravy je výpočetně velmi náročná a celý proces zabere značné množství času. Hlavní výhodou ale je to, že lze tuto fázi provést pro danou množinu šablon pouze jednou. Výsledná připravená data je možné uložit a následně využívat u různých scén opakovaně. Celý proces přípravy klade na šablony určité požadavky a skládá se z mnoha kroků, které jsou popsány v následujících podsekcích.

#### 3.1 Požadavky na formát šablon

Princip detekce je založen na porovnávání objektů ve scéně se známými objekty, šablonami. Aby mohlo být takové porovnávání úspěšné, je nutné, aby bylo u těchto vstupních šablon splněno několik kritérií. Prvním z nich je dostatečné množství informací o vzhledu objektu. Každý objekt je nutné nasnímat ze všech úhlů, aby byla následná shoda mezi objektem ve scéně a známou šablonou co nejlepší. Tím se eliminují detekce objektů na špatných místech, chybějící detekce, nebo označení objektu za objekt jiný. Pro implementaci a testování zde byly využity již nasnímané objekty a scény [20]. Ve zmíněném datovém setu je celkem 8 objektů, které byly nasnímány pomocí videa a následně vyextrahovány do 1 620 šablon pro každý objekt. Celkem je tedy k dispozici 12 960 šablon.

Druhý a velmi důležitý požadavek je velikost šablon, která musí být shodná pro všechny šablony u všech objektů. V případě použitého datového setu jsou rozměry šablon  $48 \times 48$  px. Každý pohled na objekt je nutné zvětšit či zmenšit tak, aby při zachování poměru stran vyplnil co možná největší plochu obrazu. Shodná velikost všech šablon velmi ulehčuje následující techniky, které by jinak vůbec nebylo možné provádět. Případně by se tak výpočty značně zkomplikovaly, což by mohlo negativně ovlivnit správnou funkčnost algoritmu, nebo jej znatelně zpomalit.

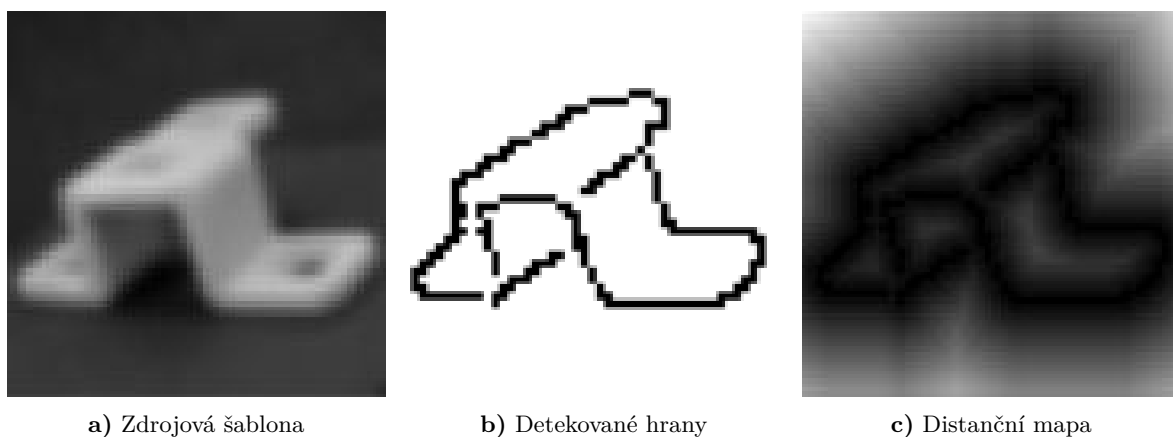
Jednotná velikost má ale i své nevýhody a to při velkých rozdílech v reálných velikostech objektů. Snímek velkého objektu musí být hodně zmenšen, což může vést až ke ztrátě důležitých informací. Pokud se naopak bude snímek velmi malého objektu zvětšovat nebo bude detailnější, může šablona obsahovat informací mnohem více, než stejný objekt umístěný ve scéně. Rozměr šablon je tedy nutné vhodně určit adekvátně k množství detailů, které budou při dané velikosti viditelné, ale i s ohledem na paměťovou náročnost. Pro detekci jsou využívána nejen data zdrojových šablon, ale také z pomocných matic a obrazů, které mají shodný rozměr. Zvětšení rozměrů by tak vedlo k většímu paměťovému vytížení všech těchto obrazů. Získání a význam dalších pomocných dat je popsán v následující podsekcí.

### 3.2 Načtení šablon a vytvoření pomocných obrazů

Jak již bylo zmíněno výše, barvy objektů nehrají při detekci žádnou roli, proto jsou obrazy ihned po načtení převedeny do stupňů šedi. Vzhledem k tomu, že je nutné šablony uchovávat v paměti po celou dobu fáze přípravy i detekce, je tím paměťová náročnost snížena  $3\times$  v porovnání s uchováváním barevných RGB obrazů. Základem pro detekci jsou hrany objektů, pro určení orientace a vzdálenosti bodů od hran je využita distanční mapa popsaná v podsekcí 2.6. Obraz s detekovanými hranami i maticí distanční mapy je taktéž nutné uchovávat po celou dobu fáze přípravy i detekce. Znázornění šablony ve stupních šedi, detekovaných hran i distanční mapy je ilustrováno na obrázku 8. V konkrétní implementaci je tato trojice nazývána jako detekční jednotka.

Hrany jsou v šablonách detekovány pomocí Cannyho detektorů hran, jak je popsáno v podsekcí 2.5.2. Kvůli tomu, že implementace Cannyho detektoru v OpenCV neobsahuje počáteční rozmazání obrazu, je nutné tento krok předem provést samostatně. Použitý algoritmus pro Gaussovo rozmazání obrazu, popsáný v podsekcí 2.4, vyžaduje jako vstupní parametr velikost jádra. Změnou velikosti jádra se mění intenzita rozmazání, ale také rychlost celého výpočtu. Čím větší jádro, tím je operace náročnější a pomalejší. V konkrétní implementaci jsou obrázky rozmazány ihned po načtení a při ukládání připravených dat se šablony ukládají už včetně tohoto rozmazání. Nutnost zvětšování síly rozostření tak nemá vliv na výslednou rychlost fáze detekce.

Získané hrany se využijí také pro vytvoření distanční mapy, která byla získána pomocí postupu zmíněného v podsekcí 2.6. Distanční transformace je dále použita v podsekcí 3.5 k výpočtu vzdálenosti bodu od hrany a její orientace. Pro vizualizaci na obrázku 8 byla výsledná matice normalizována. Tato operace ale současně znehodnotí získané informace a nesmí být nikdy provedena, pokud se budou hodnoty v matici nadále využívat při výpočtech.



Obrázek 8: Detekční jednotka



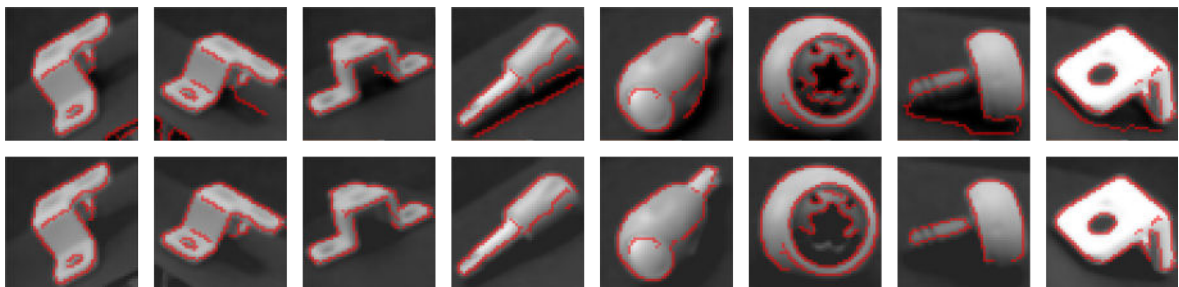
### 3.3 Problém stínů při detekci hran

Všechny objekty v datovém setu [20] byly snímány na otočném podstavci pomocí videa a následně bylo vyextrahováno 1620 šablon pro každý objekt. Při tomto procesu bohužel nebyly objekty dostatečně nasvíceny, což na jednotlivých snímcích způsobilo přítomnost stínů. Při nastavování parametrů pro Cannyho detektor hran v podsekcí 3.2 se nepodařilo dosáhnout ideálního kompromisu a to mělo za následek dvě situace: buď byly u některých objektů detekovány hrany pouze na jeho části a ve druhé části nebyly vůbec žádné, nebo naopak byly vždy v detekovaných hranách přítomny i nežádoucí stíny. U objektu pojmenovaného *block* byla v některých pohledech dokonce detekována i hrana otočného podstavce, jak je patrné v rohu prvních dvou šablon na obrázku 9.

$$T(x, y) = \begin{cases} \text{threshold} & \text{if } T(x, y) < \text{threshold} \\ T(x, y) & \text{v ostatních případech} \end{cases} \quad (6)$$

Tyto falešné hrany mohou samozřejmě způsobit nepřesnosti při detekci, protože ovlivňují získané vzdálenosti a orientace hran při následujících výpočtech. Objekty jsou nasnímány na pozadí s téměř uniformní barvou. Pouze stíny nebo hrany otočné plochy působí při detekci hran rušivě. Je tedy možné určit práh a celý snímek po načtení prahovat pomocí výpočtu (6), kde  $T(x, y)$  značí pixel šablony  $T$  na pozici  $[x, y]$  a  $\text{threshold}$  je určen proměnnou  $t_b$ . Všechny pixely, které jsou tmavší než daný práh, se nastaví na hodnotu prahu, čímž tmavé plochy zmizí. Ostatní pixely se ponechají beze změny. Prah se musí zvolit tak, aby ideálně zmizely stíny, ale nebyly ovlivněny hrany jednotlivých šablon. Konkrétní použité hodnoty jsou popsány v samostatné sekci 7.

Na obrázku 9 jsou znázorněny některé šablony před a po prahování včetně detekovaných hran. U většiny lze pozorovat značné zlepšení, v některých případech ale došlo ke zhoršení. Při porovnání obou metod měla detekce s prahováním lepší výsledky a proto je i nadále použita.



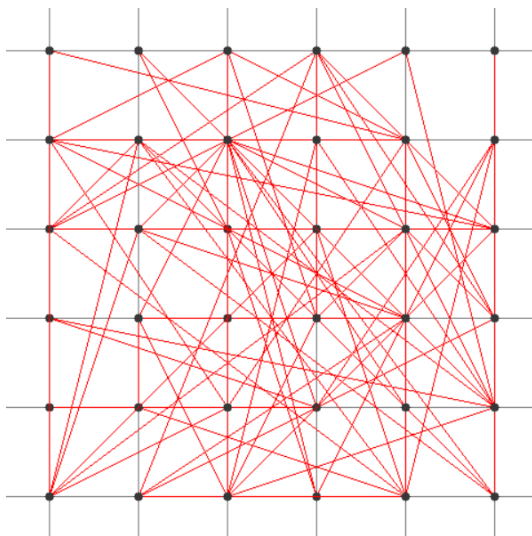
Obrázek 9: Znárodnění detekovaných hran před prahováním (nahore) a po prahování (dole)

### 3.4 Vytvoření pravidelné mřížky a generování tripletů

Kdyby porovnávání mezi šablonou a scénou, případně mezi šablonami navzájem, probíhalo na všech souřadnicích v obrazu bylo by velmi zdlouhavé a neefektivní. Proto je nejdříve vygenerována pravidelná čtvercová mřížka  $m$  bodů a pouze nad těmito body bude dále probíhat porovnávání. Mřížku je vhodné generovat s určitým odsazením od všech krajů obrazu, protože v jejich blízkosti není příliš mnoho informací a body na okrajích by měly nižší vypovídací hodnotu, než body uvnitř obrazu.

Pro potřeby hashování, které je popsáno v podsekcí 3.6, je ale výhodné, aby body nebyly samostatné, ale seskupené do trojic neboli tripletů. Dva triplety jsou shodné, pokud jsou složeny ze tří totožných bodů v libovolném pořadí. Pokud jsou mezi triplety shodné pouze jeden nebo dva body, jsou triplety považovány za různé. Nejlepších výsledků lze dosáhnout, když bude podobnost mezi všemi triplety bude co nejmenší. Postup pro vygenerování takového seznamu ale obnáší složité a komplikované výpočty, proto jsou pro zjednodušení triplety vygenerovány náhodně. Při tomto zjednodušeném postupu je ale nutné dbát na to, aby byl za účelem pokrytí všech bodů v mřížce vygenerovaný dostatečný počet tripletů.

Na obrázku 10 je znázorněna mřížka bodů a také vygenerované triplety. Každý bod, který je součástí nějakého tripletu, je následně zbarven černou barvou. Lze tedy pozorovat 100 % pokrytí všech bodů mřížky.



Obrázek 10: Znázornění pravidelné mřížky a vygenerovaných tripletů

### 3.5 Výpočet vzdálenosti a orientace hrany

Obraz s detekovanými hranami se dá považovat za binární matici. Obsahuje totiž pouze informace o tom, zda se na dané pozici nachází hrana či nikoliv. Porovnávání pouze těchto hodnot by nebylo dostačující, protože by bylo nemožné stanovit dostatečnou míru tolerance. Mnohem účinnější je v každém bodě tripletu určit vzdálenost k nejbližší hraně a orientaci dané hrany.

Tyto hodnoty lze velmi jednoduše získat z připravené distanční mapy. Vzdálenost bodu od hrany je hodnota v mapě na daných souřadnicích, orientaci lze vypočítat pomocí  $x$ -ové a  $y$ -ové derivace. Ta je znázorněna ve vzorci (7), kde  $D(x, y)$  označuje hodnotu v distanční mapě  $D$  na souřadnicích  $x$  a  $y$ . Výpočet je rychlý a jednoduchý, nicméně čím blíže k hraně se bod nachází, tím méně je přesný. Výsledná orientace se po získání kvantizuje, a proto drobné nepřesnosti nepůsobí problémy v následném zpracování. Detaily ohledně kvantizace jsou popsány v podsekcí 3.6.

$$\phi = \arctan \frac{D(x, y) - D(x, y - 1)}{D(x, y) - D(x - 1, y)} \quad (7)$$

Pokud se ale bod nalézá přímo na hraně, může výpočet (7) může vrátit pouze čtyři hodnoty a takovouto nepřesnost už nelze zanedbat. Pro výpočet orientace v takovém případě je nutné využít Sobelova filtru, vysvětleném v podsekcí 2.5.1, na daných souřadnicích vstupního obrazu. Obraz šablony nebo scény je ihned po načtení převeden do stupňů šedi a následně také rozmazán. To je zároveň i požadavek pro vstupní obraz Sobelova filtru, takže není potřeba provádět další kroky před samotným aplikováním operátoru nad daným obrazem. Tato skutečnost má pozitivní vliv na rychlost celé fáze přípravy i detekce samotné. Implementace řešení tohoto problému je velmi jednoduchá a je zobrazena v kódu 1. Datová struktura `DetectionUnit` neboli detekční jednotka obsahuje všechny 3 matice popsané v podsekcí 3.2.

---

```
void getEdgeDistAndOri(DetectionUnit &img, int x, int y, float &distance, float
    &orientation, bool onlyPositive = true)
{
    distance = img.distanceTransform_32f.at<float>(y, x);
    orientation = (distance == 0.0f) ?
        getEdgeOrientationWithSobel(img.img_8u, x, y, onlyPositive) :
        getEdgeOrientationFromDistanceTransform(img.distanceTransform_32f, x, y,
            onlyPositive);
}
```

---

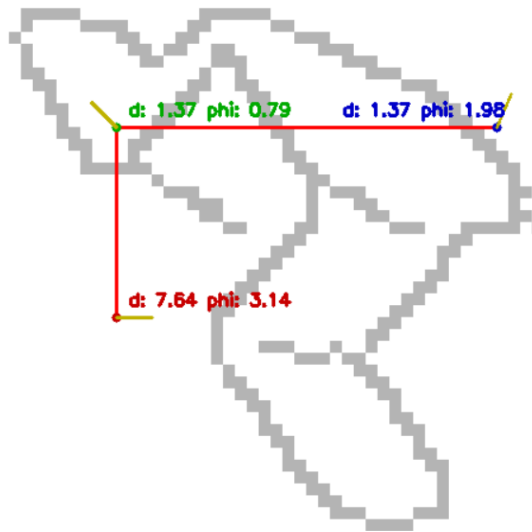
Výpis 1: Získání vzdálenosti a orientace nejbližší hrany od zadaného bodu

Výpočet orientace pomocí vzorce (7) i pomocí Sobelova filtru vrací hodnoty v rozsahu  $\langle -180^\circ; 180^\circ \rangle$ . Před samotným kvantizováním jsou ale hodnoty převedeny pouze do rozsahu  $\langle 0^\circ; 180^\circ \rangle$ , protože orientace stejné hrany může nabývat jak kladné, tak i záporné hodnoty úhlu. Znaménko v tom případě pouze určuje, zda je na hranu nahlíženo zprava, nebo zleva, což je pro další výpočty irelevantní informace. Pokud je tedy výsledný úhel záporný, je k němu přičteno  $180^\circ$ . Po této normalizaci je možné kvantizovat, neboli rozčleňovat úhly do méně tříd, protože i celkový rozsah je menší.

### 3.6 Naplnění hashovací tabulky

Výhoda hashovací tabulky je rychlost, jakou dokáže vyhledat prvek na základě klíče. Průměrná asymptotická složitost této operace je konstantní. Zvětšující se velikost tabulky tedy neovlivní čas vyhledávání. Této vlastnosti je využito pro prvotní vyfiltrování možných kandidátů ve fázi detekce. Hashovací tabulka vždy obsahuje dvojice klíč–hodnota. Aby bylo možné docílit vyhledání v konstantním čase, je nutné pro klíč, který může být libovolného datového typu, vygenerovat hash. V ideálním případě by hashovací funkce měla pro dva různé klíče vždy vygenerovat dvě různé hash hodnoty, což ale někdy může být velmi složité.

Na každou šablonu je postupně přikládán každý vygenerovaný triplet, na základě kterého jsou získány hodnoty ve všech třech bodech. Z každého tripletu je tak získáno celkem šest hodnot, tři jsou vzdálenosti bodů od hran a tři orientace daných hran, což je znázorněno na obrázku 11. Získané hodnoty jsou poté kvantizovány do  $n_d$  tříd pro vzdálenosti a  $n_\phi$  tříd pro orientace. K těmto šesti kvantizovaným hodnotám je přiřazena sedmá hodnota, kterou je pořadové číslo tripletu. Následně je vytvořen klíč obsahující všech sedm hodnot a pod tímto klíčem je daná šablona uložena do hashovací tabulky. V případě vyhledávání je možné pomocí shodného klíče tuto šablonu nalézt v konstantním čase. Protože tripletů je celkem  $l$ , bude i šablona do tabulky vložena celkem  $l \times$  pod  $l$  různými klíči.



Obrázek 11: Vizualizace tripletu na šabloně s konkrétními hodnotami vzdáleností a orientací ve všech třech bodech. Žlutá čára značí orientaci hrany modulo  $\pi$

Ve více různých šablonách mohou kvantizované hodnoty pro jeden triplet vyjít shodné, což znamená, že se do hashovací tabulky musí umístit více různých šablon pod jeden klíč. Proto hashovací tabulka pod každým klíčem obsahuje pole, do kterého se postupně umísťují šablony, jejichž sedmice hodnot je totožná. V poli pod jedním klíčem se tedy nacházejí šablony, které jsou ve třech bodech tripletu na základě kvantizovaných hodnot vzdáleností a orientací podobné.

Umístění šablon v hashovací tabulce je vizualizováno na obrázku 12, kde jsou jednotlivé hodnoty vzdáleností a orientací již kvantizovány. V tomto případě už není možné vyhledat šablonu v konstantním čase, protože po nalezení klíče je nutné prohledat celé pole. Tento fakt ale není nijak omezující, protože se ve fázi detekce pracuje vždy s celým nalezeným polem, nikoli s jednotlivými šablonami. Navíc dochází při rovnoměrném rozložení hodnot do všech tříd ke zkracování jednotlivých polí v tabulce.

Triplet: 7 d1: 2    phi1: 4 d2: 1    phi2: 5 d3: 3    phi3: 0	
Triplet: 10 d1: 3    phi1: 0 d2: 1    phi2: 4 d3: 1    phi3: 4	
Triplet: 17 d1: 0    phi1: 5 d2: 3    phi2: 2 d3: 2    phi3: 0	
Triplet: 31 d1: 0    phi1: 3 d2: 3    phi2: 1 d3: 3    phi3: 2	
Triplet: 42 d1: 1    phi1: 5 d2: 2    phi2: 0 d3: 0    phi3: 3	

Obrázek 12: Znázornění ukládání jednotlivých šablon v hashovací tabulce. V levém sloupci je všech sedm hodnot určující klíč, v pravém sloupci je pole obsahující všechny šablony jejichž kvantizované hodnoty v bodech tripletu jsou totožné

### 3.6.1 Hashovací funkce

Klíčem tabulky je struktura sedmi hodnot. Nyní je nutné pro tuto strukturu vytvořit hashovací funkci. V konkrétní implementaci v jazyce C++ byla ze začátku využita třída `unordered_map`, která je součástí standardní knihovny. Ta vyžaduje, aby hash klíče byl celočíselný. Protože jsou jednotlivé hodnoty klíče relativně malé celočíselné hodnoty, lze hash vytvořit jednoduše pomocí poskládání všech hodnot za sebe binárními posuny, což je znázorněno ve výpis kódu 2. Pokud celkový počet tripletů a počty tříd vzdáleností  $n_d$  a orientací  $n_\phi$  nebudou příliš vysoká čísla, lze tímto způsobem generovat naprosto unikátní hash pro celý rozsah možných hodnot.

Z principu vnitřní implementace třídy `unordered_map` není unikátní hash klíče dostačující a dalším požadavkem je, aby se vzor posledních  $h_p$  významných bitů příliš neopakoval. Hodnotu  $h_p$  nelze jednoduše bez dodatečných informací stanovit předem, protože se mění dle celkové velikosti tabulky a to i v průběhu plnění. V případě použitého datového setu tabulka obsahuje po naplnění více než 100 000 řádků, neboli unikátních klíčů, a významné bity obsahují opakující se vzor. To způsobuje problém, protože při modulárních výpočtech hashe jsou často navraceny shodné hodnoty. Na základě těchto výpočtů probíhá ukládání klíčů do stále shodného sektoru a tabulka pak není v paměti rovnoměrně zaplněná. Dochází tak jak ke zpomalení vkládání, tak i vyhledávání v tabulce, a tím v důsledku i zpomalení celého procesu detekce.

Aby se zpomalení z výše popsaných důvodů zamezilo, je ještě před navrácením hodnoty nutné použít vhodnou funkci, která má dobrou statistickou distribuci v rámci všech bitů celého rozsahu. V ukázce kódu 2 je znázorněna vlastní implementace hashovací funkce pomocí binárních posunů všech kvantizovaných hodnot, které generují unikátní hash pro všechny možné kombinace klíčů. Výsledná hodnota je ale ještě upravena tak, aby bylo zamezeno opakujícímu se vzoru. Úprava je provedena dalšími binárními výpočty, které jsou přejaty z open-source projektu H2 [21].

---

```
namespace std
{
    template <>
    struct hash<QuantizedTripletValues>
    {
        std::size_t operator()(const QuantizedTripletValues& k) const
        {
            unsigned int h = (k.tripletIndex << 18) | (k.d1 << 15) | (k.d2 << 12) |
                (k.d3 << 9) | (k.phi1 << 6) | (k.phi2 << 3) | (k.phi3 << 0);
            // Výpočet níže přejat z~open-source projektu H2
            h = ((h >> 16) ^ h) * 0x45d9f3b;
            h = ((h >> 16) ^ h) * 0x45d9f3b;
            h = (h >> 16) ^ h;
            return h;
        }
    };
}
```

---

Výpis 2: Ukázka hashovací funkce

Později byla třída `unordered_map` nahrazena implementací `tsl::hopscotch_map` od uživatele Tessila [22]. Ta obsahuje stejné API jako `unordered_map`, ale na základě přiložených testů dosahuje lepších výsledků při vkládání i vyhledávání. To si lze také ověřit v sekci 5 a 6. Navíc obsahuje metodu `overflow_size`, která vrací v případě špatné implementace hashovací funkce

hodnotu různou od nuly. Na základě této funkce byly zjištěny a odladěny problémy s hashovací funkcí, kdy docházelo k opakování vzoru posledních  $h_p$  bitů.

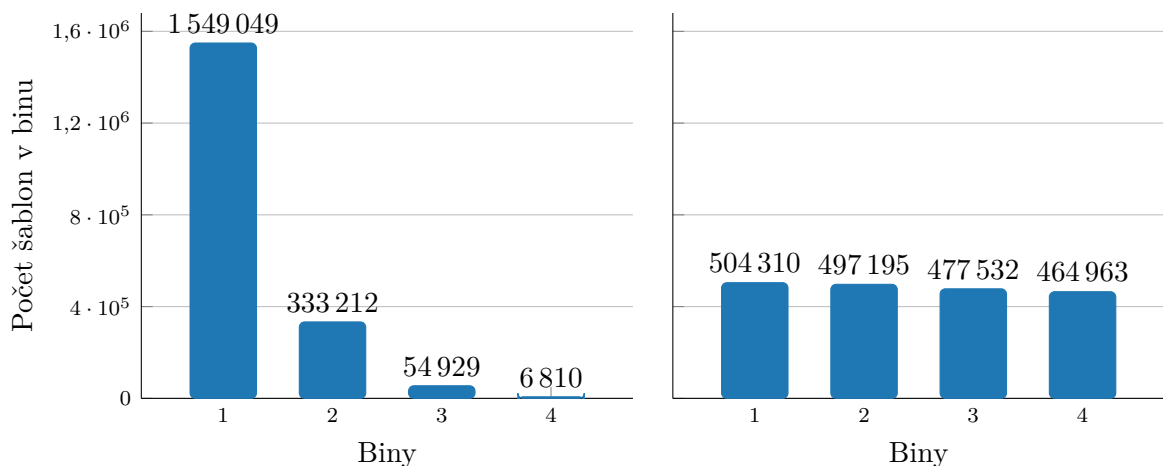
### 3.6.2 Rovnoměrné rozdělení do binů

Binem se nazývají jednotlivé třídy histogramu, které jsou definovány rozsahem hodnot, jejichž četnost se započítává do stejné třídy, neboli binu. Správné rozdělení do binů může značně ovlivnit nejen rychlost detekce, ale také velikost hashovací tabulky. Ta se používá k prvotní filtraci možných kandidátů a je tedy žádoucí, aby jich vrátila co nejméně. Pokud se ale bude do stejných binů kvantizovat velké množství hodnot, bude i výsledný hash shodný. Následkem poté bude rostoucí délka polí v hashovací tabulce obsahujících šablony znázorněných na obrázku 12. Ve fázi detekce to v důsledku povede k navrácení velkého množství kandidátů, čímž bude účinnost hashovací tabulky pro prvotní detekci značně snížena. Cílem tedy je, aby se do všech binů kvantizovalo přibližně stejné množství hodnot, což zároveň povede k rovnoměrnému rozložení šablon v tabulce.

Nejjednodušší možnost, jak určit horní a dolní mez intervalu každého binu, je získání minimální a maximální možné hodnoty vzdáleností i orientací a rozdělit je na  $n_d$  stejných intervalů pro vzdálenosti a  $n_\phi$  intervalů pro orientace. Pokud by byly hodnoty v celém rozsahu rovnoměrně rozložené, bylo by do všech binů zařazeno přibližně stejné množství hodnot. K této situaci ale téměř nikdy nedojde a rozdělení tedy rovnoměrné nebude. Pro datový set [20] to je patrné na obrázcích 13a) a 13c). Zde jsou vzdálenosti rozdělovány do 4 binů a orientace do 6 binů. Všechny intervaly pro vzdálenosti jsou shodně velké, stejně jako všechny intervaly pro orientace.

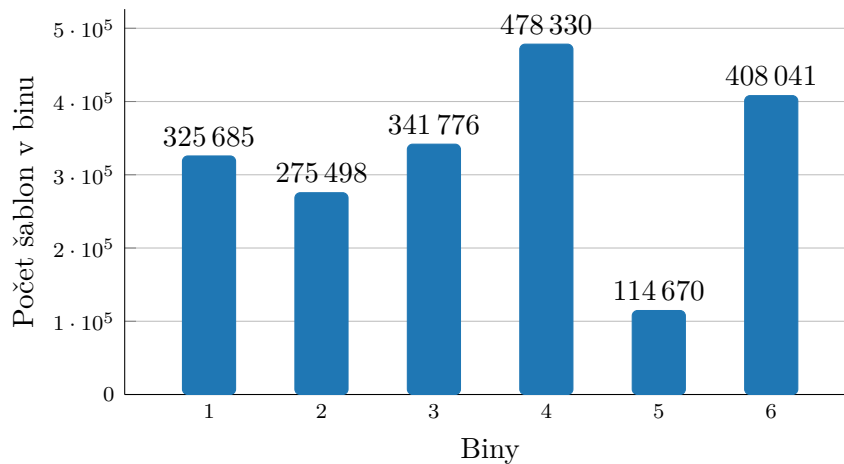
Mnohem lepších výsledků lze dosáhnout, pokud se intervaly pro jednotlivé biny určí pomocí kvantilů. Kvantily lze jednoduše získat uložením všech získaných hodnot do pole, které je následně seřazeno. Počáteční a koncové hodnoty jednotlivých intervalů se určí tak, aby do každého binu spadalo procentuálně stejné množství hodnot. Rozložení do jednotlivých binů je poté mnohem rovnoměrnější, než při použití shodně velkých intervalů. To lze porovnat v grafech na obrázcích 13a) a 13b).

Když byly všechny intervaly vzdáleností stejně velké, byla délka hashovací tabulky přibližně 37 000 řádků a nejdelší pole obsahovalo více než 1 000 šablon. Po určení intervalů pomocí kvantilů se celková délka tabulky zvýšila na více než 160 000 řádků a nejdelší seznam se zkrátil na přibližně 300 šablon. Rozdělení pomocí kvantilů se aplikuje pouze na intervaly pro vzdálenosti. Intervaly pro orientace jsou i nadále rozloženy rovnoměrně.



a) Rozložení vzdáleností do binů se stejně velkým rozsahem

b) Rozdělení vzdáleností do binů s intervaly určených pomocí kvantilů



c) Rozložení orientací hran do binů se stejně velkým rozsahem

Obrázek 13: Rozložení vzdáleností a orientací do binů

### 3.7 Orientované zkosení hran

Podobnost mezi dvěma šablonami nebo mezi šablonou a scénou lze definovat pomocí podobnosti vzdáleností a zkosení jednotlivých hran. Na každé hraně vstupní šablony je určena její orientace a také orientace nejbližší hrany z druhého vstupního obrazu. Dále je mezi těmito hranami vypočítaná vzdálenost. Pokud je rozdíl orientací a spočtená vzdálenost nižší, než je definovaný práh, je hrana označena jako podobná. Aby bylo možné skóre porovnávat mezi šablonami s různým počtem hran, je výsledné skóre vyděleno celkovým počtem hran vstupní šablony.

Na základě pokusů [23] bylo zjištěno, že v případě velkého rozptylu celkového počtu hran mezi šablonami jednotlivých objektů má výsledek výpočtu podobnosti tendenci konvergovat



k jednodušším objektům. To znamená, že při porovnání objektu ve scéně s jednodušší šablonou objektu může být výsledné skóre vyšší, než při porovnání se šablonou složitějšího objektu. A to i v případě, že je šablona složitějšího objektu ve skutečnosti mnohem podobnější, či dokonce totožná s objektem detekovaným ve scéně.

Z tohoto důvodu byl výpočet podobnosti (8) orientovaného zkosení hran definován už se zohledněním kompenzace této skutečnosti, definovanou pomocí průměrného počtu hran napříč všemi šablonami všech objektů. Bez této modifikace by bylo možné výsledek výpočtu použít jako procentuální podobnost mezi šablonami. Při její přítomnosti to ale již možné není, protože výsledná hodnota  $s_\lambda(T, I)$  může nabývat hodnot i mimo rozsah  $\langle 0; 1 \rangle$ . Konkrétně může být vyšší než 1.

$$\begin{aligned} \overline{|T|} &= \frac{1}{n} \sum_{i=1}^n |T_i| \\ s_\lambda(T, I) &= \frac{\sum_{e \in T} |d_I(e) \leq \theta_d, |\phi_T(e) - \phi_I(e)|_\pi \leq \theta_\phi|}{\lambda|T| + (1 - \lambda)\overline{|T|}} \end{aligned} \quad (8)$$

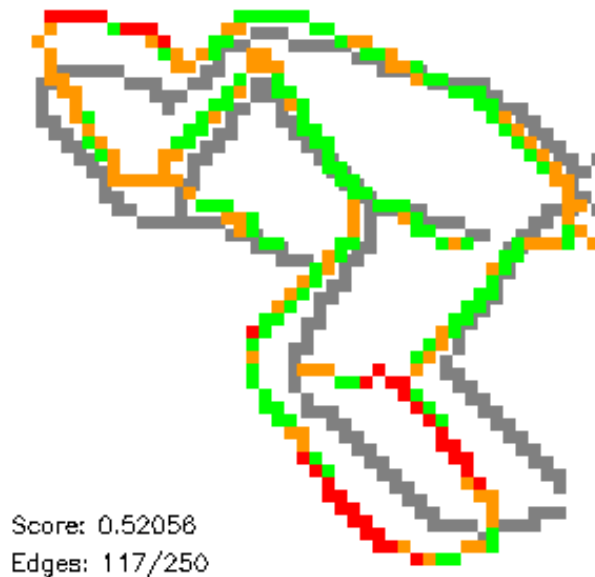
Parametr  $\lambda$  určuje sílu významnosti průměrného počtu hran napříč všemi šablonami a může nabývat hodnot  $\langle 0; 1 \rangle$ . Funkce obrazu  $d_I(e)$  získá vzdálenost a  $\phi_I(e)$  orientaci nejbližší hrany v obraze  $I$  z bodu, na kterém leží hrana  $e$  v obraze  $T$ . Hodnota  $\phi_T(e)$  je konkrétní orientace hrany  $e$  v obraze  $T$ . Vzdálenost i orientace se počítají dle techniky dříve popsané v podsekcí 3.5. Operátor  $|\cdot|_\pi$  je znázornění výpočtu modulo hodnotou  $\pi$ , což je další možný výpočet pro převedení hodnot do rozsahu  $\langle 0^\circ; 180^\circ \rangle$ . Celkový počet hran v šabloně  $T$  je značen  $|T|$  a  $\overline{|T|}$  označuje průměrný počet hran napříč všemi šablonami. Konstrukcí  $|\cdot|$  je označen enumerátor neboli jednoduché počítadlo splněných podmínek.

Je nutné, aby byly obrazy  $T$  a  $I$  stejných rozměrů, neboť mezi oběma obrazy probíhá porovnávání na shodných souřadnicích. Tento požadavek je zde automaticky splněn, protože všechny použité šablony mají shodnou velikost. Při porovnávání mezi šablonou a scénou je scéna nejdříve zvětšena či zmenšena tak, aby měla oblast zájmu shodnou velikost jako šablona. Následně je ze scény vyříznuta a použita k porovnání.

Na obrázku 14 je graficky znázorněno porovnání dvou šablon pomocí orientovaného zkosení hran. Šedé hrany pocházejí z porovnávané šablony, barevné hrany pochází z původní šablony. Zeleně jsou zaznačeny hrany, jejichž rozdíl orientací i vzdálenost je nižší než definovaný práh. Pokud je vzdálenost nižší než práh, ale rozdíl orientací je vyšší, je hrana zaznačena oranžově. Pokud jsou obě hodnoty vyšší, než definované práhy, je hrana znázorněna červeně. V levém spodním rohu obrázku je vypsáno, kolik hran z celkového počtu splňuje obě podmínky, a jaké je výsledné skóre spočítané pomocí výpočtu (8).

Ve výsledku v rohu obrázku je zřetelné, jak ve výpočtu orientovaného zkosení hran splnila svůj účel kompenzace průměrným počtem hran napříč všemi šablonami. Hran, které splňují obě

podmínky a jsou označeny jako odpovídající, je méně než polovina z celkového počtu. Přesto se výsledné skóre pohybuje lehce nad hodnotou 0,5.



Obrázek 14: Vizualizace porovnání dvou šablon a výpočet orientovaného zkosení hran

### 3.8 Mazání hran podle stability pohledu a jejich orientace

Ve fázi detekce objektů ve scéně bude docházet k porovnávání scény s možnými kandidáty pomocí detekovaných hran na základě výpočtu bližší popsaného v podsekcí 3.7. Za účelem zrychlení detekce je možné část těchto hran ze šablon odebrat, a to bez negativního efektu na výslednou kvalitu celého procesu. Odebírání hran probíhá ve dvou krocích, které se provádí postupně. Kritéria pro odebrání hran jsou následující:

1. stabilita pohledu - hrany, které se při mírném natočení kamery příliš liší, jsou odebrány,
2. orientace - část hran s největším zastoupením shodné orientace jsou odebrány.

Jako první probíhá odstranění hran dle stability k pohledu, neboli k pozici kamery. Pokud se daná hrana při drobném natočení kamery nemění, její důležitost stoupá, protože objekt nemusí být ve scéně natočen naprosto přesně jako v jedné z mnoha dodaných šablon. Pokud se naopak hrana mění, mohla by její přítomnost způsobit snížení celkového skóre podobnosti. V důsledku by tak mohla nastat situace chybějící detekce, nebo by mohlo dojít k označení jiného objektu. Protože dodaná sada šablon neobsahuje dodatečné informace o tom, které objekty jsou si podobné, je nutné tyto informace získat.

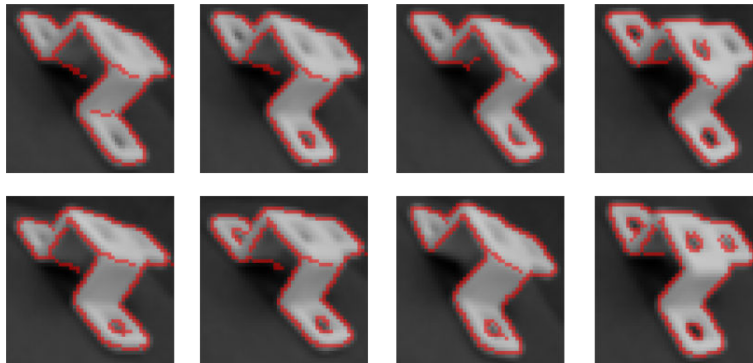
Pro porovnávání podobnosti dvou šablon lze využít stejného postupu jako v případě detekce, neboli porovnávání hodnot orientovaného zkosení hran popsaného v podsekcí 3.7. Tento

výpočet bude značně výpočetně i časově náročný, protože je nutné každou šablonu porovnat se všemi ostatními. V případě datového setu [20] bude tedy třeba porovnat každou šablonu se všemi dalšími 1619 šablonami daného objektu. Celý proces se bude opakovat celkem 8×, protože set obsahuje 8 různých objektů. Znázornění doby trvání operace nad jednotlivými objekty je znázorněn v sekci 6 v tabulce 3.

$$n_T(e) = |\{T' \in N(T) \mid |d_T(e) - d_{T'}(e)| \leq \theta_d, |\phi_T(e) - \phi_{T'}(e)|_\pi \leq \theta_\phi\}| \quad (9)$$

Následně dochází k samotnému odstraňování hran v šabloně  $T$  s ohledem na  $k$  podobných šablon, označených jako  $N(T)$ . Hrana je odstraněna, jestliže v  $k$  šablonách neobsahuje podobné hrany, které jsou vypočítány pomocí vzorce (9). Tj  $n_T(e) < \tau k$ , kde  $0 < \tau < 1$ . Ve výpočtu (9) jsou za  $T'$  postupně dosazovány šablony z  $N(T)$ . U všech se kontroluje, jestli je vzdálenost mezi hranami  $e$  v šabloně  $T$  a  $T'$  menší než definovaný práh  $\theta_d$  a také jestli je rozdíl orientací hran  $e$  v šabloně  $T$  a  $T'$  menší než práh  $\theta_\phi$ . Pokud ano, jsou spočteny enumerátorem  $|\cdot|$  podobně jako ve výpočtu (8).

Při příliš vysoké hodnotě  $k$  už dochází k nalezení podobnosti mezi šablonami, u kterých se úhel natočení kamery značně liší od šablony  $T$ . Vhodnější je proto udržovat hodnotu  $k$  spíše nižší a raději zvýšit práh  $\tau$ . Ukázka prvních sedmi podobných šablon je znázorněna na obrázku 15.

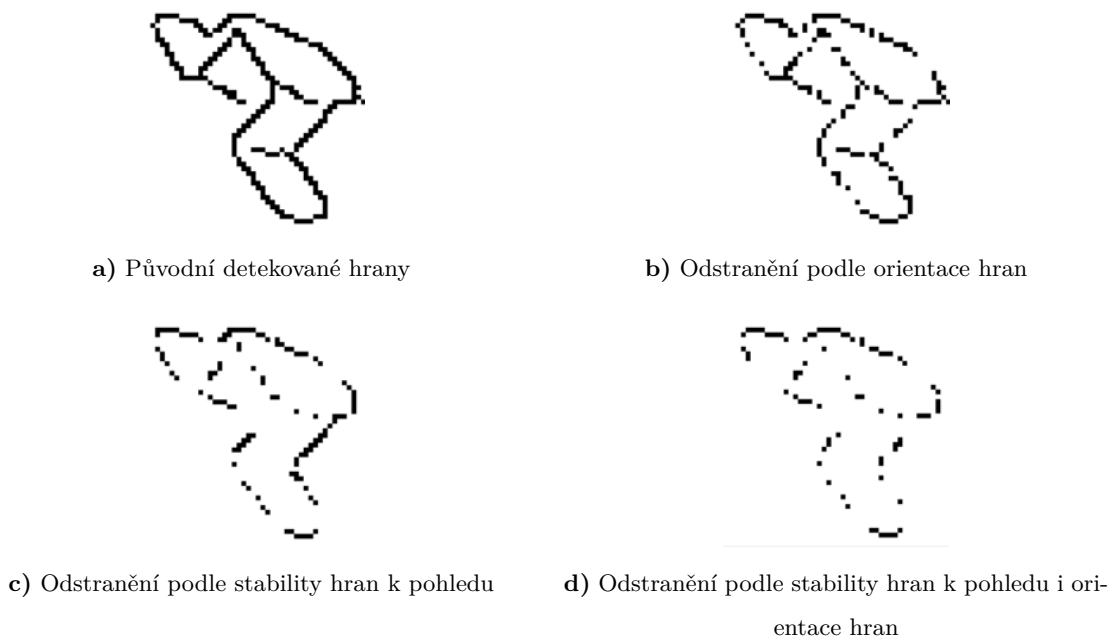


Obrázek 15: Vstupní šablona a dalších 7 podobných šablon seřazených po řádcích od nejpodobnějších. U některých šablon lze pozorovat velký rozdíl v pozici kamery

Proces odebrání hran pokračuje druhým krokem, který pracuje na základě výsledků získaných v prvním kroku. Již odstraněné hrany jsou ještě dále filtrovány, v tuto chvíli podle orientace vůči všem hranám dané šablony. Pokud hrany objektu obsahují dlouhé rovné čáry, budou tyto hrany méně významné, protože se v jejich délce nic nemění. Pokud jim je ale přikládána stejná důležitost, výsledné porovnání může objekty lišící se pouze na jednom z konců označit za shodné.

Jestliže bude část z těchto hran chybět, měl by se efekt výsledné detekce zlepšit a zároveň také zrychlit, protože se bude mezi šablonami navzájem porovnávat menší počet hran. V prvním kroku je získán histogram orientací jednotlivých hran, které jsou poté rozřazeny do  $n_\phi$  binů, podobně

jako v podsekcí 3.6. Následně je náhodně vybráno a odstraněno  $p\%$  hran, jejichž orientace náleží do dvou binů s nejvyšším počtem hran. Na obrázku 16 je znázorněno jak vypadá odstranění hran pomocí jednotlivých metod a zvláště a následně kombinace obou metod dohromady.



Obrázek 16: Ukázka odstranění hran pomocí stability pohledu a orientace hran

Po odstranění hran je důležité si ponechat distanční mapu získanou z původních neodfiltrovaných hran. Hrany jsou odebírány z důvodu urychlení výpočtu (8), kdy bude porovnávání probíhat pouze na tom zlomku hran, které v obraze zůstaly. Vzdálenost a orientace smazaných i pozůstalých hran by se však změnila, kdyby došlo k přegenerování distanční mapy. Také v případě, kdy se určuje orientace hrany pomocí Sobelova operátoru, popsaném v podsekcí 3.5, probíhá výpočet v původním obraze, který se nezměnil. Kdyby se tedy distanční mapa přegenerovala, výpočty by nebyly z těchto důvodu nadále konzistentní.

### 3.9 Ukládání připravených dat

Fáze přípravy šablon je výpočetně náročná. Jak je patrné z dat uvedených v sekci 6 v tabulce 3, trvá téměř čtyři minuty. Je tedy výhodné si připravená data uložit tak, aby je bylo možné opakovaně používat bez zbytečné prodlevy. Při tomto postupu je ale vhodné si uložit pouze ta nejnutnější data, protože paměťová náročnost celé aplikace v paměti RAM při použití datového setu [20] je více než 200 MB. Přitom velikost souborů trénovacích šablon z datového setu, které jsou navíc barevné, je pouze 50 MB.

Z údajů v tabulce 6 lze vyčíst, že je časově nejnáročnější proces odebírání hran. Naopak naplnění hashovací tabulky je velmi rychlé, stejně jako vypočítání a seřazení hodnot, které jsou pro plnění potřebné. Proto není nutné si tyto informace vůbec ukládat a mohou být velmi rychle vypočítány znovu. Navíc bude při menším množství zapisovaných dat do souboru na pevný disk i tato operace provedena rychleji.

V implementaci jsou po rozboru časové a paměťové náročnosti uloženy tripletety, zdrojové šablony po rozmazání a matice s detekovanými a vyfiltrovanými hranami. Tripletety není nutné ukládat, v případě potřeby je možné je po načtení všechny znovu přegenerovat. Pokud se ale použije shodná množina tripletetů, vrací detekce při opakovaném spouštění stále stejné výsledky. To ale v případě přegenerování neplatí, protože tripletety jsou rozmístěny v mřížce vždy náhodně.

Nejnáročnější operací je při diskovém zápisu vytváření velkého množství souborů. Naopak zapsání jediného velkého souboru může být i několikanásobně rychlejší. Proto jsou všechny šablony včetně tripletetů uloženy do jediného souboru, který zabírá při použití datového setu [20] přibližně 58 MB diskového prostoru.

Pro zapsání všech dat do jediného souboru je nejjednodušší použít binární zápis. V knihovně OpenCV, která je v implementaci použita, ale existuje hotové řešení pro ukládání a načítání matic pouze do formátu XML nebo YAML. Při použití binárních souborů je nutné dbát na fakt, že data nejsou strukturována. Při ukládání pole nebo matic je proto nezbytné uložit i počty či velikosti dat, které se zapisují. V případě pole pak i jeho délku, u matice její velikost a datový typ buněk. Pořadí zapisovaných dat je podrobněji rozebráno v následující podsekcí 4.1, protože je přímo ovlivněno potřebnými mezivýpočty.

## 4 Detekce objektů ve scéně

Ve fázi detekce už probíhá hledání objektů ve scénách s využitím dříve připravených dat. V datovém setu [20] je celkem 60 testovacích scén, na kterých se nacházejí kromě natrénovaných objektů i různé jiné objekty, které fungují jako rušivé elementy detekce. Prvních třicet scén je nasnímáno na černém kontrastním pozadí, zbylých třicet je nasnímáno na stole se světle hnědou dřevěnou texturou. Jedna scéna z každé podskupiny je znázorněna na obrázku 17. Ke všem 60 scénám jsou také dostupná ground truth data, která jsou použita pro zjištění úspěšnosti detektoru.



a) Scéna s tmavým kontrastním pozadím

b) Scéna nasnímaná na světle hnědém stole s dřevěnou texturou

Obrázek 17: Ukázka testovacích scén

### 4.1 Načítání předem připravených dat

V posledním kroku fáze přípravy jsou do jediného binárního souboru uložena připravená data, jak je popsáno v podsekcí 3.9. Z důvodu úspory času při ukládání a načítání a také kvůli úspoře diskového prostoru nejsou uložena úplně všechna data potřebná pro fázi detekce. Proto je potřeba chybějící data před zahájením samotné detekce znovu dopočítat.

V průběhu načítání je nutné provádět také několik dalších operací. Jako první je v souboru uloženo celé pole všech tripletů, které je potřeba během plnění hashovací tabulky. Následují trénovací šablony, které jsou již uloženy v odstínech šedi, protože podobně jako ve fázi přípravy je jejich barevná složka nadbytečná a byla by po načtení stejně odstraněna. Šablony jsou dále rozostřeny Gaussovým filtrem, stejně jako v podsekcí 3.2, a obsahují částečně vyhlazené stíny pomocí práhování. Tato technika je popsána a zdůvodněna v podsekcí 3.3.

Po načtení každé ze zdrojových šablon je ihned vygenerována detekční jednotka, neboli trojice matic obsahující zdrojový obraz, detekované hrany a distanční mapu. Detekční jednotka je blíže popsána v podsekcí 3.2. V tuto chvíli jsou ale v maticích hran přítomny všechny detekované hrany, ne jen ty už vyfiltrované z kroku popsaném v podsekcí 3.8. Důvodem je nutná přítomnost původních hran pro vytvoření distančních map, které se využijí při plnění hashovací

tabulky. Distanční mapy by nebyly konzistentní s přípravnou fází, kdyby byly vytvořeny z již vyfiltrovaných hran.

Po načtení tripletů, zdrojových šablon a vytvoření detekčních jednotek následuje plnění hashovací tabulky. Získání všech potřebných dat pro plnění a plnění samotné je naprosto totožné jako v průběhu přípravné fáze popsané v podsekcí 3.6. Obsahuje tedy získání vzdáleností a orientací ve všech bodech tripletů, určení rozsahů hodnot pro kvantizaci pomocí kvantilů i následné plnění hashovací tabulky.

V závěru jsou ze souboru načteny matice s odfiltrovanými hranami, které nahradí detekované hrany během přípravy detekčních jednotek. Odfiltrované hrany slouží pouze k urychlení fáze detekce, jak je vysvětleno v podsekcí 3.8, a na přípravu dat nemají žádný vliv. Dalším nutným krokem je vypočítání průměrného počtu hran napříč všemi šablonami. Tento údaj je potřebný při výpočtu orientovaného zkosení hran popsaného v podsekcí 3.7.

Triplety i původní šablony v odstínech šedi jsou v souboru uloženy už na začátku a až poté následují matice s vyfiltrovanými hranami, které jsou potřeba až po naplnění hashovací tabulky. Toto uspořádání má za následek sekvenční čtení souboru po celou dobu operace, což je mnohem rychlejší, než náhodné přístupy do souboru. Sekvenční čtení ze souboru a zpracování by se velmi obtížně paralelizovalo, přesto lze při porovnání mezi tabulkami 4 a 3 vidět velké rozdíly v časech potřebných pro operaci načítání. Načtení šablon bez paralelizace z jednotlivých souborů je přibližně 4× pomalejší, než načtení stejného počtu šablon z jediného binárního souboru. Pokud jsou původní šablony načítány paralelně, je operace téměř dvojnásobně pomalejší než načítání z jednoho binárního souboru.

Protože jsou při prvotním načítání v sekci 3.2 šablony také rozmazány, nelze údaj v tabulce 3 porovnávat přímo s hodnotou v tabulce 4. Z tohoto důvodu bylo provedeno jedno měření navíc, při kterém došlo při načítání šablony z připravených dat i k jejímu rozmazání. Tyto časy jsou uvedeny v tabulce jako akce načtení s kompenzací operace rozmazání. Pro další výpočty je ale tento postup nepoužitelný, protože při rozmazání již rozmazaného obrazu by totiž nedošlo ke stejné detekci hran. To by ve svém důsledku ovlivnilo i všechny následující výsledky. Proto byla tato operace po testovacím měření z implementace opět odebrána. I přes drobné zpomalení při aplikování rozostření obrazů, je načítání z jediného binárního souboru stále značně rychlejší, než načítání z mnoha jednotlivých souborů.

## 4.2 Získání kandidátů pomocí obrazové pyramidy a posuvného okna

Po načtení a přípravě všech dat z předchozí podsekcí je už možné zahájit detekci objektů ve scénách. Pro detekci ve všech scénách se využívají shodně načtená data, takže je možné detekování neustále opakovat bez nutnosti znovu načítat připravená data. Každá scéna je po nahrání převedena do stupňů šedi a také bezprostředně poté rozostřena. Při detekci je použita obrazová pyramida v kombinaci s metodou posuvného okna. Obě techniky jsou vysvětleny v podsekcí 2.2.

### 4.2.1 Metoda obrazové pyramidy

Je několik možností, jak implementovat metodu obrazové pyramidy. Obraz lze zmenšovat, a to buď vždy z původního obrazu, anebo z předchozího stupně pyramidy. Protože je nutné obraz zároveň i rozostřit, lze toto rozmazání provést pouze na začátku nebo v každém stupni pyramidy zvláště.

Při prvním pokusu byla scéna rozostřena vždy až po zmenšení, které probíhalo v každém stupni pyramidy přímo ze zdrojového obrazu. Při druhém pokusu byla scéna rozostřena ihned po načtení a následně zmenšována vždy z předešlého stupně obrazové pyramidy. V obou pokusech docházelo ke shodným výsledkům. Varianta s jediným rozmazáním před zahájením obrazové pyramidy je ale časově méně náročná, protože k rozostření dochází pouze jednou pro všechny stupně. Ve třetím pokusu byla testována varianta obrazové pyramidy, kdy dochází po každém zmenšení z předchozího stupně i k rozmazání. Při tomto přístupu ale docházelo už k tak velkému rozmazání a zhoršení detekce, že byla tato varianta implementace pro další testování úplně zamítnuta. Výsledná implementace tedy provede pouze jedno rozmazání ihned po načtení obrazu a následné zmenšování probíhá vždy z předchozího stupně, nikoliv z originálního obrazu.

### 4.2.2 Technika posuvného okna a získání kandidátů

V každém stupni obrazové pyramidy je aplikována technika posuvného okna, které je velikostí stále shodné pro všechny stupně, ke zmenšování totiž dochází u scény. Existují i řešení s opačným postupem, při kterých je zachována stále stejná velikost scény a dochází ke zvětšování posuvného okna. Hodnoty vzdáleností a orientace je možné v daném poměru přepočítat na základě podobnosti trojúhelníků, problém ale nastává v posuvném okně o větší ploše, protože to už obsahuje více detailů. Pokud by mělo posuvné okno jiný rozměr, než jaký mají šablony, byl by také problémový výpočet orientovaného zkosení hran.

V každém zastavení posuvného okna jsou z hashovací tabulky získána pole všech potenciálních kandidátů, kteří se mohou na dané pozici a v dané velikosti nacházet. Každé pole je vráceno podle klíče, který je vypočten na základě jednotlivých tripletů a šesti hodnot nalezených ve všech třech bodech. Tvorba klíče je podrobněji popsána v podsekcí 3.6. Celkově je tedy vráceno  $l$  polí, nad kterými probíhá ve dvou krocích filtrace potenciálních kandidátů. V první fázi jsou ponecháni pouze ti kandidáti, kteří se ve všech  $l$  polích vyskytují nejméně  $\theta_v$  krát, ostatní jsou z dalšího zpracování odstraněni. Pokud existuje i nadále pro dané posuvné okno více než jeden možný kandidát, nastává druhá fáze filtrace. Během té jsou všichni kandidáti porovnáváni na základě skóre orientovaného zkosení hran, které je vysvětleno v sekci 3.7. Ponechán je jen ten kandidát, která má nejvyšší skóre a ostatní jsou zahozeni.

Scéna je zmenšována celkem  $w_n \times s$  poměrem zmenšení  $w_r$  oproti předchozímu stupni. V tabulce 1 jsou znázorněny jednotlivé velikosti scén ve všech stupních pro  $w_n = 10$  a  $w_r = 1, 2$ . Na konkrétním datovém setu není potřeba provádět i zvětšování, protože nejmenší objekty ve scéně nejsou menší nebo shodné velikosti jako trénovací šablony. Posuvné okno má po celou



Tabulka 1: Velikost scén v jednotlivých stupních obrazové pyramidy

Stupeň	Poměr zmenšení	Výsledná velikost
0	1,000	640 × 480 px
1	1,200	533 × 400 px
2	1,440	444 × 333 px
3	1,728	370 × 278 px
4	2,074	308 × 232 px
5	2,488	257 × 193 px
6	2,986	214 × 161 px
7	3,583	178 × 134 px
8	4,300	148 × 112 px
9	5,160	123 × 93 px

dobu shodnou velikost  $48 \times 48$  px, stejně tak, jak jsou velké i jednotlivé šablony, na kterých proběhlo naplnění hashovací tabulky. Posuvné okno se nejdříve pohybuje po vodorovné ose po krocích  $w_s$  px. Po dosažení pravého kraje obrazu je navraceno k okraji levému a po svislé ose rovněž posunuto o  $w_s$  px.

Náročnost detekce objektů v jednotlivých stupních pyramidy je znázorněna v sekci 6 v tabulce 5. Protože je časová náročnost úměrná počtu objektů a složitosti scény, jsou v tabulce uvedeny průměrné hodnoty. Podrobnější popsání hodnot je v sekci, na kterou je odkazováno.

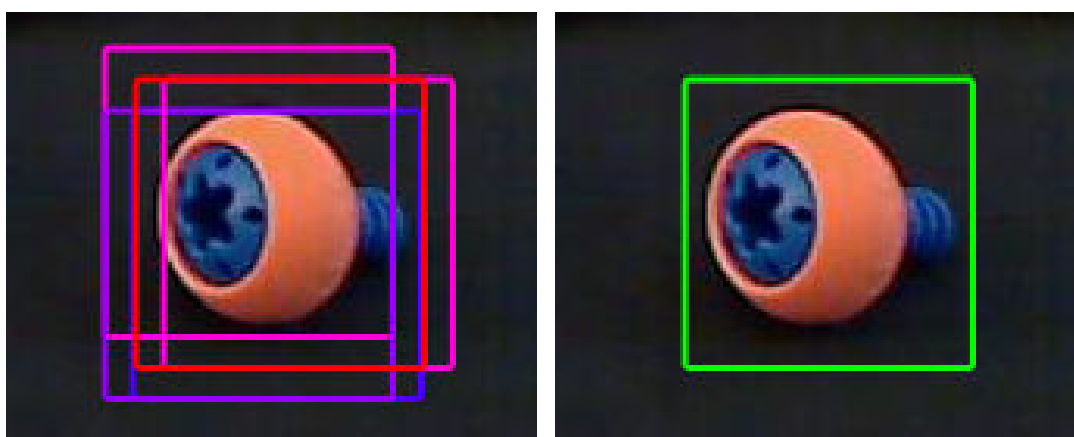
### 4.3 Non-maxima suppression

Technika non-maxima suppression, zkráceně NMS, už byla zmíněna v popisu postupu Cannyho detektoru hran v podsekci 2.5.2. Tato metoda je využívána v mnoha algoritmech pro zpracování obrazu, vždy ale trochu jiným způsobem, protože je jako taková velmi obecná. Jedná se o techniku, kdy jsou jednotlivé entity porovnávány s dalšími entitami v určitém okolí. Následně jsou ponechány pouze lokální maxima neboli ty entity s maximální hodnotou pro určené okolí. Všechny ostatní entity jsou odstraněny. Entita může být téměř cokoli. V případě zmíněného Cannyho detektoru to jsou pixely hran, u detektoru objektů jsou to nejčastěji bounding boxy, stejně jako v dále popsaném případě. Určení lokálního maxima může být prováděno různými způsoby. Ve většině případů jsou ale jednotlivé entity číselně ohodnoceny, aby bylo možné jednoduše porovnávat jejich skóre.

V předchozím kroku může být navraceno obrovské množství potencionálních kandidátů, protože z každého zastavení posuvného okna ve všech stupních pyramidy může být navracen jeden. To způsobuje, že je jeden objekt detekován několikrát, a to i v jeho blízkém okolí, což

je patrné na obrázku 18a). Právě technika non-maxima suppression je v tomto případě vhodná k určení nejsilnějšího kandidáta a zahazení všech ostatních.

Vstupem algoritmu NMS je pole bounding boxů seřazené podle jejich souřadnice x a následně souřadnice y. Algoritmus NMS postupně prochází všechny bounding boxy a pokud se překrývají o více než  $o_m\%$ , jsou navzájem porovnány. Bounding box s vyšším skóre orientovaného zkosení hran je ponechán, druhý je zahozen. Takto je zpracována celá množina všech potencionálních kandidátů, až zůstanou pouze lokální maxima. V ukázce kódu 3 je vlastní implementace algoritmu non-maxima suppression a na obrázku 18 jsou znázorněny jeho vstupy a výstupy. Skóre bounding boxu je vyznačeno přechodem barvy z modré do červené.



a) Vstupním parametrem je pole bounding boxů. Čím červenější bounding box je, tím je skóre vyšší

b) Ponechány jsou pouze lokální maxima, neboli box s nejlepší skóre

Obrázek 18: Znázornění metody non-maxima suppression

Pro porovnání procentuálního překrytí se často využívá Jaccardův index podobnosti, který už byl vysvětlen v sekci 2.7. Pokud je malý bounding box umístěn uvnitř velkého, může dle Jaccardova indexu nastat situace, kdy budou oba ponechány. Pro aktuální situace ale není toto chování příliš žádoucí. Objekty se totiž ve scénách příliš nepřekrývají. Z tohoto důvodu bylo nutné použít jiné měření, které není tak striktní.

$$O_P(A, B) = \frac{|A \cap B|}{\min(|A|, |B|)} \quad (10)$$

Upravený výpočet pro procentuální překrytí je znázorněn ve vzorci (10) a řeší problém umístění malého bounding boxu uvnitř velkého, kdy je výsledkem 100% překrytí. Při použití upraveného vzorce došlo k velkému poklesu false positive detekcí. Další zlepšení nastalo i v poklesu potřebné doby pro operaci NMS, protože bylo odfiltrováno více potencionálních kandidátů. To sebou ale neslo riziko toho, že zároveň dojde i k zahazení true positive detekcí. Výsledné F1 skóre, které je vysvětleno v následující sekci, ale vychází lépe v případě použití vzorce (10).

Aplikováním algoritmu non-maxima suppression je dokončena fáze detekce a aktuální výstup je již finální. Časová náročnost je, stejně jako v případě popsaném v předchozí podsekcí 4.2, závislá na celkovém počtu vrácených kandidátů. Proto je v tabulce 5 uvedena i průměrná náročnost pro jednotlivá měření.

---

```
void nonMaximaSupression(std::vector<Candidate> &candidates)
{
    std::sort(candidates.begin(), candidates.end());
    for (int i~= 0; i< candidates.size(); i++)
    {
        bool startFromBeginning = false;
        if (!candidates[i].active) { continue; }
        for (int j = i+1; j < candidates.size(); j++)
        {
            if (!candidates[j].active) { continue; }
            if (candidates[i].percentageOverlap(candidates[j]) >= NMSMinOverlap)
            {
                if (candidates[i].chamferScore > candidates[j].chamferScore)
                {
                    candidates[j].active = false;
                }
                else
                {
                    candidates[i].active = false;
                    i~= j;
                    startFromBeginning = true;
                }
            }
            else if (candidates[i].rect.x + candidates[i].rect.width <= candidates[
                j].rect.x)
            {
                break;
            }
        }
        if (startFromBeginning){ i~= -1; }
    }
}
```

---

Výpis 3: Implementace non maxima suppression

#### 4.4 Výpočet F1 skóre na základě ground truth dat

Bounding boxy, neboli oblasti zaznačené člověkem určující správnou pozici objektu, se nazývají ground truth data, zkráceně GT. V případě detektoru různých objektů nebo při rozpoznávání jsou dostupné také informace o jaký objekt se jedná, případně jaké jsou jeho vlastnosti. V datovém setu [20] jsou u testovacích scén přítomny textové soubory s příponou gt, ve kterých jsou na každém řádku vypsány názvy objektů následované souřadnicemi a velikostmi bounding boxů. Podle souboru readme, který je u datového setu přítomen, jsou za názvem následovány souřadnice  $[x, y]$  levého horního rohu a souřadnice pravého spodního rohu. Při bližším zkoumání však bylo zjištěno, že se jedná o chybu a ve skutečnosti je druhá dvojice hodnot šířka a výška bounding boxu.

$$\begin{aligned} \textit{precision} &= \frac{TP}{TP + FP} \\ \textit{recall} &= \frac{TP}{TP + FN} \\ F_1 &= 2 * \frac{\textit{precision} * \textit{recall}}{\textit{precision} + \textit{recall}} \end{aligned} \quad (11)$$

Ground truth data se používají pro určení úspěšnosti detektoru pomocí porovnávání GT dat s detekovanými oblastmi. Použitý výpočet se nazývá F1 skóre a je popsán ve vzorci (11). Nejprve je ale nutné znát úroveň přesnosti (angl. precision) a citlivosti (angl. sensitivity nebo recall) detektoru, které se získají z následujících hodnot:

- **True positive**, zkráceně TP, je detekce správného objektu na správném místě.
- **False positive**, zkráceně FP, je označení detekce objektu, který se na daném místě nenachází.
- **False negative**, zkráceně FN, značí chybějící detekci. V případě více druhů objektů jde také o označení objektu za objekt jiný.
- **True negative**, zkráceně TN, při aktuálně řešeném detektoru vypočítat nelze a pro výpočet F1 skóre není ani potřebný.

Pro lepší představu jsou jednotlivé hodnoty TP, FP a FN znázorněny i na obrázku 19. Vypočtené F1 skóre vždy vychází v rozsahu  $(0; 1)$ , takže jej lze převést na procentuální úspěšnost detektoru. Pro správný výpočet hodnot true positive a false negative je také nutné zjistit, zda je detekováno správné místo nebo nikoliv. Pro porovnání mezi daty ground truth a detekovaným bounding boxem se používá Jaccardův index, již zmíněný v sekci 2.7. Pokud je index podobnosti mezi oběma oblastmi alespoň  $o_g\%$ , je detekce považována za shodu. Pokud je na daném místě i správně detekovaný objekt, jedná se o true positive detekci, v opačném případě o false negative detekci. Implementace algoritmu pro porovnání všech detekcí vůči všem ground truth bounding

boxům je podobná jako v případě non-maxima suppression popsaného v sekci 4.3 a ve zdrojovém kódu 3.



a) True positive detekce,  
označení správného objektu  
na jeho pozici

b) False positive detekce,  
označení neexistujícího  
objektu

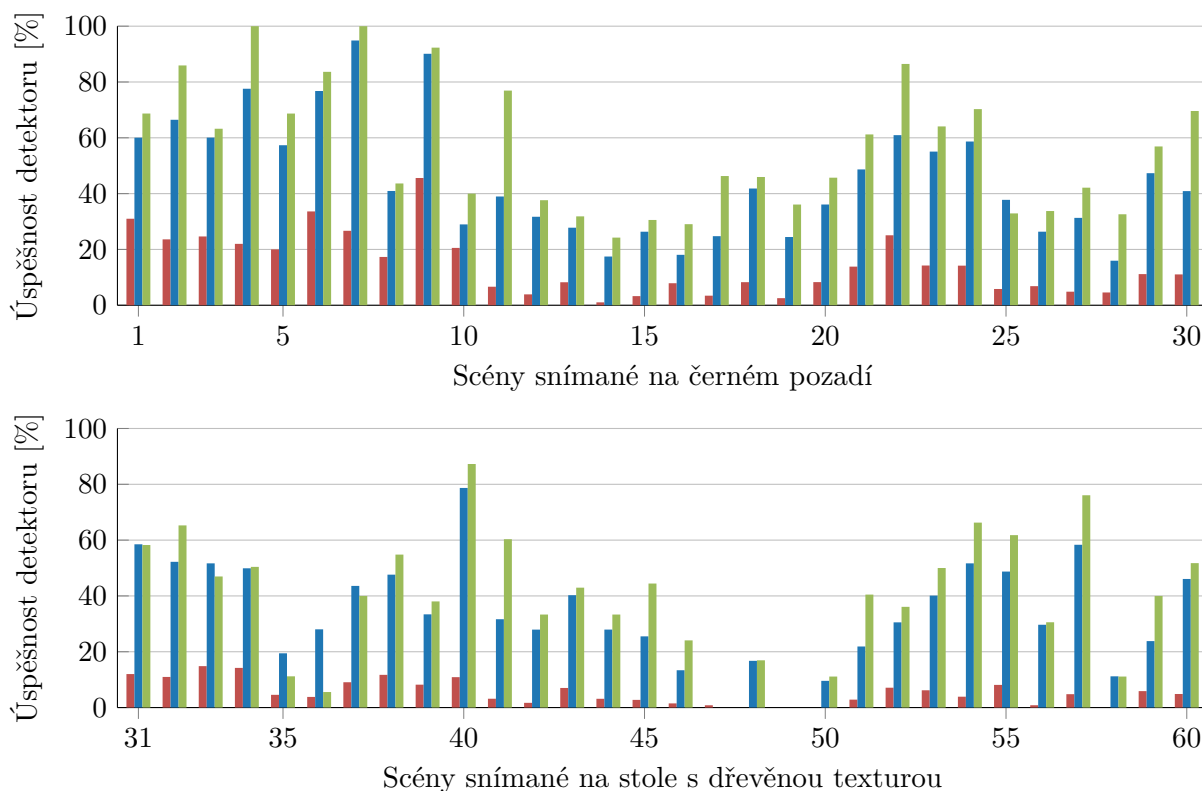
c) False negative detekce,  
označení objektu za objekt  
jiný

Obrázek 19: Znázornění evaluace binárního klasifikátoru při porovnávání ground truth bounding boxů s detekovanými lokacemi

## 5 Výsledná úspěšnost detektoru a další vylepšení implementace

Implementace detektoru obsahuje postupy, které již byly popsány v předchozích sekcích. Tyto postupy byly vytvořeny na základě dříve provedeného výzkumu [23]. Protože ale některé z nich nebyly v práci dostatečně specifikovány, byly na základě detekovaných problémů s cílem dosáhnout co nejlepších výsledků částečně opraveny. I přes tato drobná zlepšení byla ale už v průběhu implementace výsledná úspěšnost detektoru velmi slabá. Na obrázku 20 je znázorněno vypočtené F1 skóre pro jednotlivé scény a v tabulce 2 jsou údaje o celkové úspěšnosti detektoru. V tabulce 5 jsou také uvedeny časové náročnosti jednotlivých stupňů obrazové pyramidy.

Největším problémem detektoru je extrémní množství false positive detekcí, což je patrné z tabulky 2. Následkem toho bylo velké zpomalení celého průběhu detekce, jelikož muselo být zpracováno mnohem více získaných informací. Cílem níže popsaných vylepšení je nalezení metody či metod, které co možná nejvíce slabých kandidátů odfiltrují už v počáteční fázi procesu. Při podrobnějším zkoumání průběhu detekce byly odhaleny příčiny velkého množství false positive detekcí.



Obrázek 20: Úspěšnost detektoru v jednotlivých scénách při původní implementaci (červená), po dodatečných úpravách (modrá) a při ponechání všech hran v šablonách (zelená). Každá hodnota byla určena vypočtením průměru ze tří měření

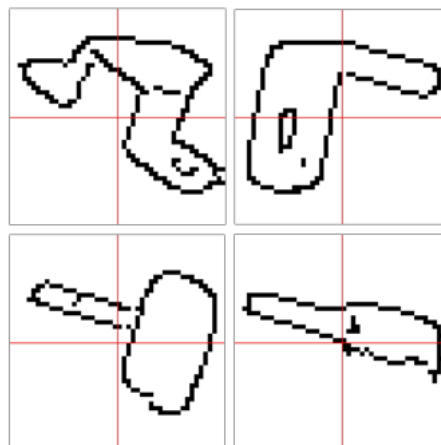
Všechny objekty z datového setu [20] jsou po nasnímání oříznuty tak, aby zabíraly co možná největší plochu šablony. Tím se maximalizuje i počet detekovaných hran v každé šabloně. Při detekci ve scéně ale kandidáti často obsahovali jen velmi málo hran nebo byly hrany přítomny pouze v malé části obrazu, jak je znázorněno na obrázku 21. Z důvodu, že v původní implementaci nebyly definovány žádné prahy, nedošlo ani k odstranění takových detekcí z dalšího zpracování. Odstraněny byly až v průběhu operace non-maxima suppression, anebo byly, v horším případě, ponechány jako false positive detekce.

Pro odstranění těchto jednoznačně špatných kandidátů byla použita metoda, která kontroluje dostatečný počet hran v jednotlivých kvadrantech obrazu. Po detekování hran ve výřezu scény proběhlo spočtení hran v jednotlivých kvadrantech. Ve všech kvadrantech musí být přítomno alespoň  $q_e$  hran. Tento přístup by ale v krajních případech mohl odfiltrovat i true positive detekce, protože některé objekty nejsou ze všech úhlů zcela pravidelné. Na obrázku 22 je patrné, že při některých pohledech je jeden ze čtyř kvadrantů prázdný a neobsahuje žádné hrany. Proto je i při detekci dostačující, pokud alespoň tři kvadranty obsahují minimálně  $q_e$  hran. Poslední kvadrant pak může obsahovat méně hran, případně žádné.

Ve složitějších scénách, kde jsou jednotlivé objekty blíže u sebe, byli špatní kandidáti ponecháváni i nadále. A to i v případě, kdy byl ve všech čtyřech kvadrantech dostatečný počet hran. Důvodem byl nízký celkový počet hran ve výřezu. Zvýšení prahu  $q_e$  by však mohlo vést ke snížení true positive detekcí. Proto byl definován další práh, který určuje minimální počet hran v celém výřezu. Tento práh je určen automaticky na základě načtených šablon. Před plněním hashovací tabulky v sekci 4.1 byl zjištěn minimální počet hran napříč všemi šablonami. Ten je následně lehce ponížěn vynásobením proměnnou  $e_r$ , protože ve scéně nemusí být hrany detekovány s takovou přesností a kvalitou jako je tomu ve scéně. Pokud výřez ze scény neobsahuje dostatečný počet hran, je z dalšího zpracování také vyřazen a ignorován.



Obrázek 21: Znárodnění hran pouze na okrajích obrazu, které jsou přesto validní kandidáti



Obrázek 22: Ukázka několika extrémních případů, kdy hrany šablon nezasahují do všech čtyř kvadrantů obrazu

Potlačení kandidátů generujících false positive detekce jen na základě počtu hran ale není dostačující. Ve scénách, kde je kromě objektů i pozadí se silnou texturou, dochází k nalezení velkého počtu hran. V takové situaci nezamezí špatným detekcím ani určení prahu minimálního počtu hran v celém výřezu nebo jeho kvadrantech.

Podobnost mezi výřezem a šablonou určuje skóre orientovaného zkosení hran popsané v sekci 3.7. Kvůli kompenzaci průměrným počtem hran není možné považovat výsledek skóre za procentuální podobnost. Přesto je ale možné definovat nejnižší práh skóre pro podobnosti  $t_c$ , kterého musí kandidát minimálně dosáhnout, aby nebyl z následného zpracování odebrán. Rozsahy hodnot skóre pro true positive a false positive detekce se budou téměř vždy částečně překrývat. Proto je nutné hodnotu prahu určit tak, aby bylo dosaženo co největšího odfiltrování false positive detekcí, ale zároveň co nejmenšího počtu true positive detekcí. Konkrétní hodnoty použité při implementaci jsou popsány blíže v sekci 7.

Kombinací obou zmíněných postupů pro odstranění false positive detekcí bylo dosaženo značného zlepšení úspěšnosti detektoru. Toto zlepšení ilustruje tabulka 2 a je patrné i z obrázku 20. Při porovnání tabulek 5 a 6 v sekci 6 lze pozorovat zkrácení doby v jednotlivých fázích detekce. Pokud jsou kandidáti odstraněni na základě malého počtu hran, nedochází ani k výpočtu skóre orientovaného zkosení hran. To vede k velkému zrychlení, protože je výpočet samotný značně komplikovaný a časově náročný. Pokud je kandidát odstraněn až na základě nedostatečného skóre, k tak velkému zrychlení už nedochází. Tato metoda ale odfiltruje větší množství false positive detekcí a vede ke zvýšení F1 skóre.

Tabulka 2: Porovnání úspěšnosti detektoru a rychlosti detektoru v různých modifikacích algoritmu

Modifikace algoritmu	Úspěšnost detektoru				Průměrný čas [s]
	TP	FP	FN	F1 [%]	
1. Původní implementace	420	9726	643	7,49	157,27
2. Přidání prahů	592	1457	471	38,05	115,56
3. Přidání prahů a výkonnější hashovací tabulky	592	1457	471	38,05	104,16
4. Ponechání všech hran v šablonách	564	706	498	48,37	126,19
5. Použit Jaccardova indexu v non-maxima suppression	823	3725	480	28,13	105,64

V tabulce 2 je změřeno pět modifikací algoritmu. Pro každou variantu proběhlo měření celkem třikrát a do tabulky byly zaneseny celkové počty TP, FP a FN. Skóre F1 je vypočítáno ze všech měření najednou. Čas byl ale získán průměrem ze tří měření a značí průměrnou dobu pro zpracování všech šedesáti dostupných scén. Za „původní implementaci“ je označována verze algoritmu ještě před aplikováním změn popsaných v této sekci.

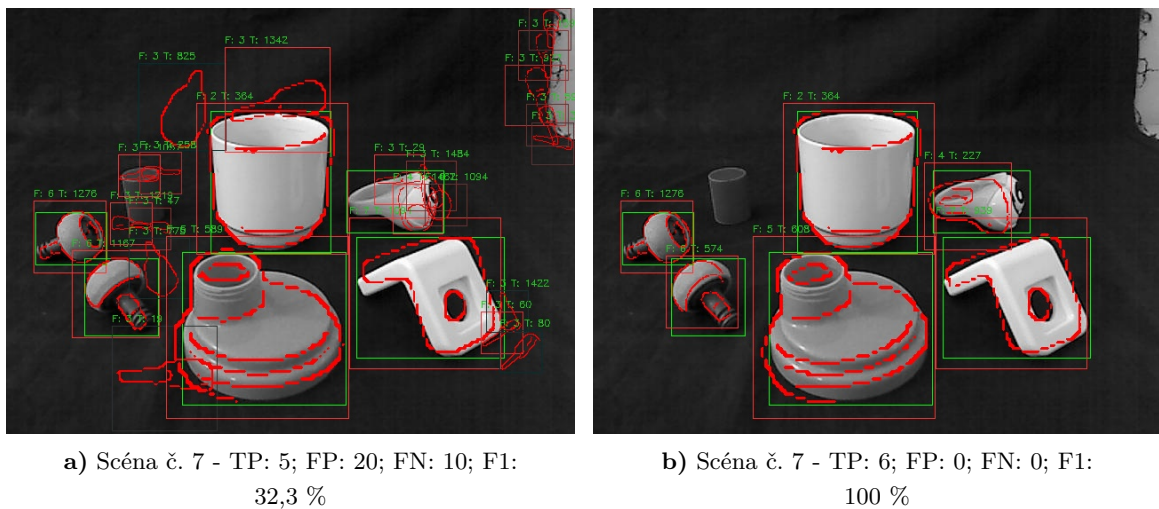


Druhé měření proběhlo po přidání všech tří dříve zmíněných prahů. Ve třetím měření byly prahy ponechány a byla vyměněna implementace hashovací tabulky za výkonnější (popsáno v sekci 3.6). Mezi druhým a třetím měřením nedošlo ke změně úspěšnosti detektoru, ale pouze ke zrychlení celé operace. Pro všechna další měření již byly ponechány jak prahy, tak i výkonnější hasovací tabulka.

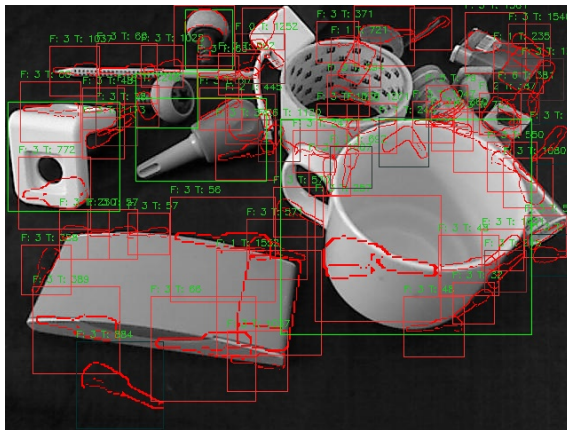
Ve čtvrtém měření byly beze změny zachovávány všechny detekované hrany v šablonách, nikoliv ale odfiltrované ze sekce 3.8. Výsledné F1 skóre bylo v této modifikaci vyšší, protože došlo k velkému poklesu false positive detekcí. Zároveň ale došlo k poklesu true positive a nárůstu false negative detekcí. Protože je nutné provádět porovnávání mezi výřezem ze scény a šablonou na více hranách, došlo také k prodloužení času výpočtu.

V posledním, pátém, měření byl při operaci non-maxima suppression nahrazen vlastní výpočet překrytí, popsaným v podsekcí 4.3, Jaccardovým indexem podobnosti. Celkový počet true positive detekcí znatelně vzrostl, bohužel ale přibýlo ještě více false positive detekcí, což vedlo k celkovému snížení F1 skóre.

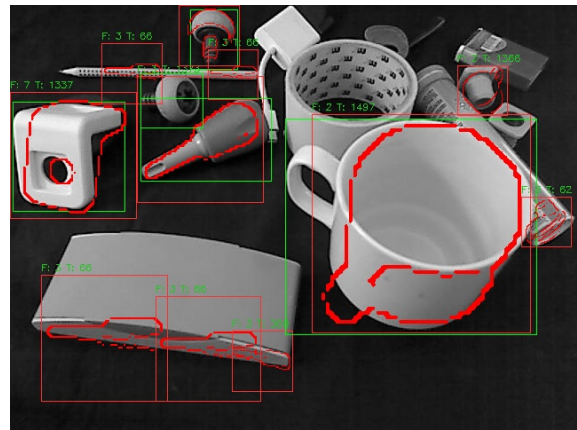
Na snímcích v obrázku 23 jsou vyobrazeny vybrané scény po dokončení fáze detekce. V levém sloupci je detekce prováděna původní implementací, zatímco vpravo po vylepšení na základě dodaných prahů. Zeleně jsou zaznačeny ground truth bounding boxy. Červené čtverce znázorňují detekci objektů včetně zakreslených hran předpokládaného objektu. Pod každým obrázkem je uvedena úspěšnost detekce.



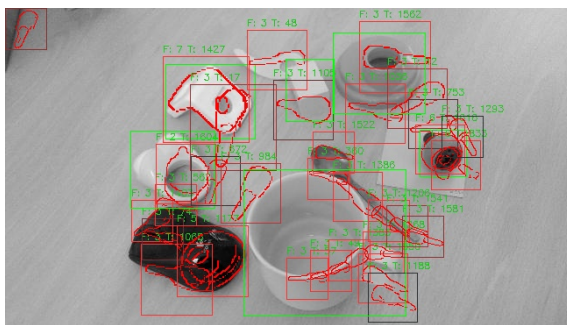
Obrázek 23: Znázornění scén



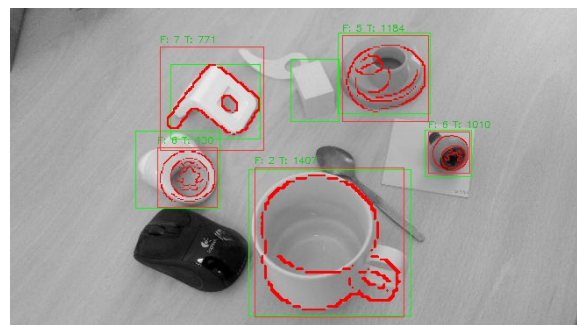
c) Scéna č. 12 - TP: 1; FP: 61; FN: 4; F1: 2,99 %



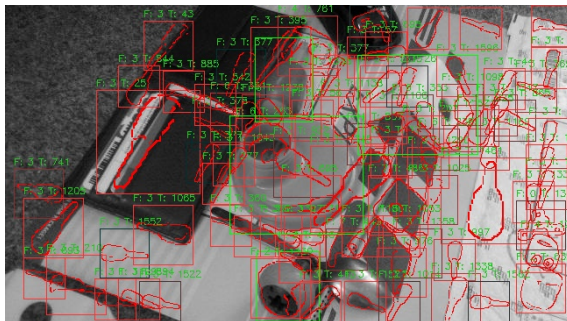
d) Scéna č. 12 - TP: 4; FP: 7; FN: 1; F1: 50 %



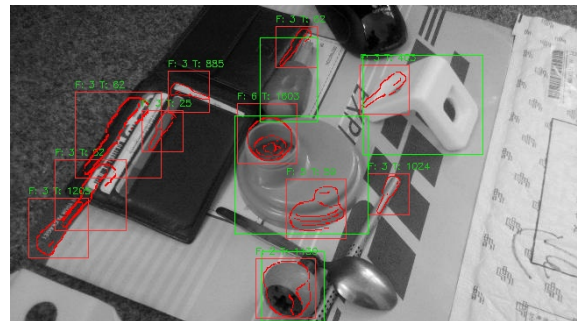
e) Scéna č. 40 - TP: 2; FP: 29; FN: 4; F1: 10,8 %



f) Scéna č. 40 - TP: 4; FP: 0; FN: 2; F1: 80 %



g) Scéna č. 47 - TP: 0; FP: 82; FN: 4; F1: 0 %



h) Scéna č. 47 - TP: 0; FP: 10; FN: 4; F1: 0 %

Obrázek 23: Znázornění scén

## 6 Časové náročnosti operací

V následujících tabulkách jsou znázorněny časové náročnosti jednotlivých výpočtů ze všech předchozích sekcí v milisekundách. Aby nedošlo ke zkreslení výsledků, bylo při měření na testovaném notebooku vždy vypnuto co možná nejvíce jiných služeb a programů, a to včetně připojení k internetu. Tím by mělo být dostatečně zamezeno všem nežádoucím vnějším vlivům na prováděné výpočty. Přesto byla k dosažení co nejpřesnějších výsledků provedena všechna měření 3× a následně vypočítán jejich průměr. Za názvem každé akce je v následujících tabulkách v závorce uvedeno číslo sekce, ve které je popsána prováděná daná operace.

Všechna měření byla prováděna na notebooku Asus ZenBook UX303LA s procesorem Intel Core i5-4210U a taktem 2,40 GHz, který obsahuje 2 fyzická a 4 logická jádra. Instalovaná je DDR3 RAM paměť o kapacitě 8 GB s frekvencí 1600Mhz. Pevný disk je typu SSD se sběrníci SATA III a kapacitou 128 GB. Instalovaný operační systém je Windows 8.1, 64-bitová verze.

Tabulka 3: Náročnost operací přípravné fáze

Akce	Měření [ms]			
	1.	2.	3.	Průměr
Načtení šablon bez paralelizace (3.2)	4600	4498	4813	<b>4637,00</b>
Načtení šablon s paralelizací (3.2)	2018	2044	1966	<b>2009,33</b>
Spočtení hodnot vzdáleností a orientací všech tripletů a šablon (3.4, 3.5)	143	152	148	<b>147,67</b>
Seřazení hodnot vzdáleností pro určení rozsahů kvantizace (3.6)	54	58	54	<b>55,33</b>
Naplnění hashovací tabulky (3.6)	249	236	261	<b>248,67</b>
Filtrace hran - šablona block (3.8)	28298	28402	27860	<b>28186,67</b>
Filtrace hran - šablona bridge (3.8)	24754	24775	24384	<b>24637,67</b>
Filtrace hran - šablona cup (3.8)	29457	29326	28950	<b>29244,33</b>
Filtrace hran - šablona driver (3.8)	16618	16588	16365	<b>16523,67</b>
Filtrace hran - šablona eye (3.8)	32113	31905	31658	<b>31892,00</b>
Filtrace hran - šablona lid (3.8)	42204	42388	41783	<b>42125,00</b>
Filtrace hran - šablona screw (3.8)	27976	28046	27796	<b>27939,33</b>
Filtrace hran - šablona whiteblock (3.8)	23250	23195	22973	<b>23139,33</b>
Uložení připravených dat do souboru (3.9)	184	192	191	<b>189,00</b>
Trvání celé fáze přípravy (3)	227364	227350	224436	<b>226383,33</b>

Tabulka 4: Časová náročnost načítání dat ve fázi detekce

Akce	Měření [ms]			
	1.	2.	3.	Průměr
Načtení dat - triplety (4.1)	<< 1	<< 1	<< 1	<< 1
Načtení dat - původní šablony (4.1)	1031	1017	1039	<b>1029,00</b>
Načtení dat - původní šablony s kompenzační operací rozmazání (4.1)	1178	1154	1187	<b>1173,00</b>
Načtení dat - plnění hashovací tabulky včetně podpůrných výpočtů (4.1)	445	444	443	<b>444,00</b>
Načtení dat - vyfiltrované hrany (4.1)	93	87	96	<b>92,00</b>
Celková doba načtení a přípravy dat (4.1)	1572	1550	1581	<b>1567,67</b>

Vzhledem k velkému počtu scén nejsou v tabulce 5 zaznačeny výsledky měření ze všech scén, ale pouze jejich průměry. Opět byla provedena tři měření, na základě kterých byla scéna s číslem 6 označena jako nejjednodušší a scéna s číslem 12 jako nejsložitější. Výpočet pro nejjednodušší scénu trval nejkratší dobu, naopak pro nejsložitější scénu trval nejdéle. V tabulce již jsou průměry hodnot ze všech tří měření, proto další průměrné hodnoty nejsou počítány. V posledním řádku tabulky je uveden v závorce údaj, kolik potencionálních kandidátů muselo být vyfiltrováno metodou non-maxima suppression a potřebná doba k provedení operace.

Data v tabulce 6 byla měřena shodně, jako v případě tabulky 5. Měření ale proběhlo po všech úpravách a vylepšeních popsáných v podsekcí 5, kde je rovněž popsán důvod výrazného zrychlení viditelného při porovnání tabulek 5 a 6. Mezi vylepšení patří také implementace výkonnější hashovací tabulky vysvětlené v sekci 3.6. I po úpravách je scéna číslo 6 shledána jako nejjednodušší a scéna číslo 12 jako nejsložitější na základě celkové doby potřebné k operaci detekce.

Tabulka 5: Náročnost jednotlivých stupňů obrazové pyramidy ve fázi detekce

Stupeň pyramidy nebo akce	Trvání operace získání kandidátů v šabloně [ms]		
	Nejjednodušší	Průměr všech	Nejsložitější
0	453,67	879,84	1415,00
1	310,67	612,23	1080,33
2	230,67	416,36	795,67
3	186,33	269,64	587,00
4	129,33	173,49	414,33
5	97,33	109,83	272,67
6	63,33	68,54	184,00
7	44,00	41,79	100,00
8	26,67	24,81	42,67
9	18,00	13,52	23,00
NMS - v závorce uveden počet kandidátů	1,93 (4471)	11,09 (8253)	26,81 (17105)

Tabulka 6: Náročnost jednotlivých stupňů obrazové pyramidy ve fázi detekce po dodatečných vylepšeních

Stupeň pyramidy nebo akce	Trvání operace získání kandidátů v šabloně [ms]		
	Nejjednodušší	Průměr všech	Nejsložitější
0	282,67	492,27	753,67
1	207,33	378,14	659,67
2	139,33	286,22	561,00
3	108,67	203,70	408,00
4	79,00	143,45	308,00
5	54,00	96,32	228,33
6	36,33	61,51	170,67
7	25,00	38,02	114,33
8	16,67	22,96	53,00
9	13,67	13,40	26,33
NMS - v závorce uveden počet kandidátů	0,03 (128)	0,04 (125)	0,12 (297)

## 7 Použité hodnoty

Ve všech předchozích sekcích byly použity konstanty, které mohou nabývat různých hodnot, a proto je potřeba je stanovit už před samotným začátkem detekce. Toto určení hodnot není triviální úkol a jejich optimální volba je také závislá na detekovaných objektech a vzhledu scény. V následující tabulce 7 jsou vypsány všechny konstanty a jejich hodnoty, které byly při testování a implementaci na základě získaných dat [20] použity.

Tabulka 7: Seznam použitých konstant a jejich výchozí hodnoty

Konstanta	Hodnota	Význam
$n_d$	4	Počet tříd histogramu vzdáleností
$n_\phi$	6	Počet tříd histogramu orientací
$\theta_d$	3,1	Maximální vzdálenost mezi hranami, aby mohly být prohlášeny za podobné
$\theta_\phi$	$\frac{\pi}{9}$	Maximální rozdíl orientací hran, aby mohly být prohlášeny za podobné
$m$	36 ( $6 \times 6$ )	Počet bodů v pravidelné čtvercové mřížce
$l$	50	Počet generovaných tripletů
$\lambda$	0,5	Koeficient určující vliv průměrného počtu hran napříč všemi šablonami při výpočtech skóre orientovaného zkosení hran
$k$	4	Počet šablon, mezi kterými jsou porovnávány podobné hrany na základě stability pohledu
$\tau$	0,6	Koeficient určující minimální procento stabilních hran
$p$	0,5	Procento hran, které jsou odstraněny
$t_b$	37	Spodní práh barvy pozadí pro odstranění stínů
$w_s$	3	Krok posuvného okna v pixelech po ose $x$ i $y$
$w_n$	10	Počet stupňů obrazové pyramidy
$w_r$	1,2	Poměr zmenšení obrazu do dalšího stupně obrazové pyramidy
$o_m$	0,5	Minimální procentuální překrytí bounding boxů pro porovnání při filtraci algoritmem non-maxima suppression
$o_g$	0,5	Minimální překrytí pro správnou detekci při srovnání s GT
$q_e$	10	Minimální počet hran v každém kvadrantu posuvného okna
$e_r$	0,95	Koeficient minimálního počtu hran v posuvném okně vůči šabloně s minimálním počtem hran
$t_c$	0,4 - 0,5	Minimální kandidátovo skóre orientovaného zkosení hran

## 8 Závěr

V této práci byl popsán postup a jednotlivé kroky algoritmu, který na základě snímku scény a množství šablon objektů detekuje jednotlivé objekty ve scéně. Celý postup lze rozdělit na dvě fáze, fázi přípravy a fázi detekce. První sekce je zaměřena na teoretické vysvětlení algoritmů, které jsou poté dále aplikovány v celé práci. Druhá sekce je již ryze praktická a zaměřuje se na techniky použité v přípravné fázi, které mají za cíl zpracovat data pro další fázi. Následuje třetí sekce s popisem fáze detekce, která využívá data z fáze přípravy a probíhá v ní hledání objektů ve scénách.

Během přípravné fáze jsou načítány šablony z datového setu a je u nich provedena řada optimalizačních kroků. Předtím je ale nutné stanovit požadavky na formát dat, aby fungoval detekční algoritmus správně. Hlavním kritériem je shodná velikost všech šablon, které obsahují každý objekt nasnímaný ze všech stran a pod různými úhly. Detekce je založena pouze na porovnávání nalezených hran mezi trénovacími šablonami a scénou a proto je nutné, aby byly hrany detekovány co nejlépe. Vzhledem k tomu, že nebylo osvětlení při snímání šablon datového setu dostatečné, byly hrany často detekovány i na stínech jednotlivých šablon. Proto bylo bezprostředně po jejich načtení provedeno práhování tmavých částí šablon, čímž se tyto chybné detekce eliminovaly.

Porovnávání mezi dvěma obrazy by bylo zdlouhavé, kdyby probíhalo na všech bodech hran. Proto bylo v pravidelné mřížce vygenerováno pole tripletů neboli trojic bodů a pouze na těchto bodech následně probíhal výpočet podobnosti. Pro určení konkrétní hodnoty podobnosti mezi šablonou a výřezem ze scény je využito skóre orientovaného zkosení hran, kterému byla věnována celá podsekce předkládané práce. Nevýhodou výpočtu tohoto skóre je dlouhá doba nezbytná k dokončení výpočtu. Proto byla v prvním kroku pro určení co nejmenšího počtu potenciálních kandidátů použita hashovací tabulka s využitím její schopnosti vyhledávat prvky dle klíče v konstantním čase. Správná implementace hashovací tabulky vyžaduje kvalitní hashovací funkci. Vzhledem k tomu, že bylo v průběhu implementace nalezeno několik problémů, které zhoršovaly výkonnost hashovací tabulky, byla tomuto problému věnována celá podsekce

Nejnáročnější částí přípravné fáze bylo odstraňování detekovaných hran v šablonách. V práci jsou popsány celkem dvě techniky, které se provádí postupně, čímž dojde k odstranění nevýznamných hran pro výpočet skóre orientovaného zkosení hran. Protože tato část přípravné fáze trvá i několik minut, bylo implementováno ukládání připravených dat. Na základě uložených dat je možné spouštět fázi detekce opakovaně a také bez zbytečného prodlení.

Další sekce práce byla věnována postupům při fázi detekce. V jejím úvodu je zmíněna příprava dat a scén, ve kterých bude hledání objektů probíhat. Pro hledání jsou využity techniky obrazové pyramidy a posuvného okna. V každém zastavení posuvného okna jsou potenciální kandidáti vyhledávání v hashovací tabulce na základě získaných hodnot v jednotlivých bodech tripletů. Pro získání podobnosti následně probíhá mezi výřezem ze scény a šablonou porovnávání na základě skóre orientovaného zkosení hran. Výpočty jsou ve fázi detekce totožné jako v přípravné fázi,

čímž se celá implementace zjednodušuje. Posledním krokem fáze detekce je použití algoritmu non-maxima suppression, který zachová pouze tu nejlepší detekci v určitém okolí. Takových detekcí může být navráceno v jednotlivých zastaveních posuvného okna několik. Pro zlepšení výsledné úspěšnosti byla definována vlastní metoda pro určení procentuálního překrytí mezi jednotlivými detekovanými bounding boxy.

Celková úspěšnost detektoru byla měřena pomocí F1 skóre na základě přítomný ground truth dat v datovém setu. Měření je v práci rovněž věnována celá sekce, ve které je podrobně popsáno, jak měření probíhalo. Výsledná úspěšnost měření ale nedosahovala hodnot, které jsou zmíněny v práci, ze které se vychází. Základní implementace, která byla provedena podle daného výzkumu, dosahovala celkové úspěšnosti okolo 7% a průměrný čas pro detekci v jedné scéně byl téměř tři sekundy. To je přibližně 5× pomaleji s 10× horší úspěšností, než zmiňuje autor práce. Pomocí několika přidávaných prahů byla dosažena úspěšnost 38% a došlo ke snížení průměrné časové náročnosti pro jednu scénu na 1,7 sekundy. Dalšího zlepšení úspěšnosti se podařilo dosáhnout při ponechání všech hran. Pokud se v přípravné fázi neodebíraly hrany ze šablon za účelem zrychlení detekce, celková úspěšnost byla téměř 50%. Tím ale také došlo ke zpomalení celé fáze detekce. Po zlepšení detekční fáze pomocí dodaných prahů bylo pro některé scény dokonce dosaženo 100% úspěšnosti.

V dané implementaci by bylo možné provádět další vylepšení, které by mohly mít vliv na rychlost nebo úspěšnost detekce. Generování tripletů probíhá náhodně. Proto může být v případě, kdy vygenerování proběhne nerovnoměrně, negativně ovlivněna celková úspěšnost detektoru. Řešením by bylo generovat tripletety tak, aby byla mezi všemi navzájem dosažena co nejmenší podobnost. Nejnáročnější fází přípravy byla filtrace hran, která při výsledném testování zhoršovala úspěšnost detekce. Podrobnější analýzou tohoto přístupu by bylo možné nalézt takový kompromis, aby při zrychlení algoritmu nedocházelo ke snížení úspěšnosti detektoru.



## Literatura

- [1] HE, Ruotao; ROJAS, Juan; GUAN, Yisheng. A 3D Object Detection and Pose Estimation Pipeline Using RGB-D Images. *arXiv preprint arXiv:1703.03940*. 2017.
- [2] SCHIELE, Bernt; CROWLEY, James L. Recognition without correspondence using multidimensional receptive field histograms. *International Journal of Computer Vision*. 2000, roč. 36, č. 1, s. 31–50.
- [3] WAH, Benjamin W. *Wiley encyclopedia of computer science and engineering*. John Wiley, 2009.
- [4] LINDEBERG, Tony. Feature detection with automatic scale selection. *International journal of computer vision*. 1998, roč. 30, č. 2, s. 79–116.
- [5] HARRIS, Chris; STEPHENS, Mike. A combined corner and edge detector. In: *Alvey vision conference*. 1988, sv. 15, s. 10–5244. Č. 50.
- [6] VEDALDI, Andrea. An open implementation of the SIFT detector and descriptor. *UCLA CSD*. 2007.
- [7] ADELSON, Edward H; ANDERSON, Charles H; BERGEN, James R; BURT, Peter J; OGDEN, Joan M. Pyramid methods in image processing. *RCA engineer*. 1984, roč. 29, č. 6, s. 33–41.
- [8] KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. 2012, s. 1097–1105.
- [9] PARKER, Jim R. *Algorithms for image processing and computer vision*. John Wiley & Sons, 2010.
- [10] TURAJLIĆ, Emir; BEGOVIĆ, Alen. Noise estimation using adaptive Gaussian filtering and variable block size image segmentation. In: *Smart Technologies, IEEE EUROCON 2017-17th International Conference on*. 2017, s. 249–254.
- [11] DAVIES, ER. *Machine Vision: Theory, Algorithms and Practicalities*. 1990. Academic Press, 1997.
- [12] MORSE, Bryan S. *Lecture 13: Edge Detection*. Brigham Young University, 2000. Dostupné také z: [http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL\\_COPIES/MORSE/edges.pdf](http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MORSE/edges.pdf).
- [13] SOBEL, Irwin; FELDMAN, Gary. A 3x3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project in*. 1968, s. 271–272.
- [14] CANNY, John. A computational approach to edge detection. In: *Readings in Computer Vision*. Elsevier, 1987, s. 184–203.
- [15] SOJKA, Eduard. *Digitální zpracování a analýza obrazů*. Vysoká škola báňská-Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2000.
- [16] BORGEFORS, Gunilla. Distance transformations in digital images. *Computer vision, graphics, and image processing*. 1986, roč. 34, č. 3, s. 344–371.

- [17] MEIJSTER, Arnold; ROERDINK, Jos BTM; HESSELINK, Wim H. A general algorithm for computing distance transforms in linear time. In: *Mathematical Morphology and its applications to image and signal processing*. Springer, 2002, s. 331–340.
- [18] NIWATTANAKUL, Suphakit; SINGTHONGCHAI, Jatsada; NAENUDORN, Ekkachai; WANAPU, Supachanun. Using of Jaccard coefficient for keywords similarity. In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. 2013, sv. 1. Č. 6.
- [19] SHI, Ran; NGAN, King Ngi; LI, Songnan. Jaccard index compensation for object segmentation evaluation. In: *Image Processing (ICIP), 2014 IEEE International Conference on*. 2014, s. 4457–4461.
- [20] *Dataset CMP-8objs*. Dostupné také z: <http://cmp.felk.cvut.cz/data/textureless/CMP-8objs/>.
- [21] *Open-source H2 embeddable RDBMS*. Dostupné také z: <https://github.com/h2database/h2database>.
- [22] *A C++ implementation of a fast hash map using Hopscotch hashing*. Dostupné také z: <https://github.com/Tessil/hopscotch-map>.
- [23] CAI, Hongping; WERNER, Tomáš; MATAS, Jiří. Fast detection of multiple textureless 3-D objects. In: *International Conference on Computer Vision Systems*. 2013, s. 103–112.