



**FAKULTA APLIKOVANÝCH VĚD  
ZÁPADOČESKÉ UNIVERZITY  
V PLZNI**

Semestrální práce z KIV/ZVI

## **Detekce hran ve snímku**

Dominik Zappe  
(A20B0279P)

Květen 2023

# **Obsah**

<b>1</b>	<b>Zadání</b>	<b>1</b>
<b>2</b>	<b>Teoretický popis metod</b>	<b>1</b>
<b>3</b>	<b>Implementace</b>	<b>4</b>
3.1	Grafické uživatelské rozhraní . . . . .	4
3.2	Předzpracování a následné zpracování obrazu . . . . .	4
3.3	Detekce hran . . . . .	4
<b>4</b>	<b>Experimenty a výsledky</b>	<b>6</b>
<b>5</b>	<b>Závěr</b>	<b>11</b>

## 1 Zadání

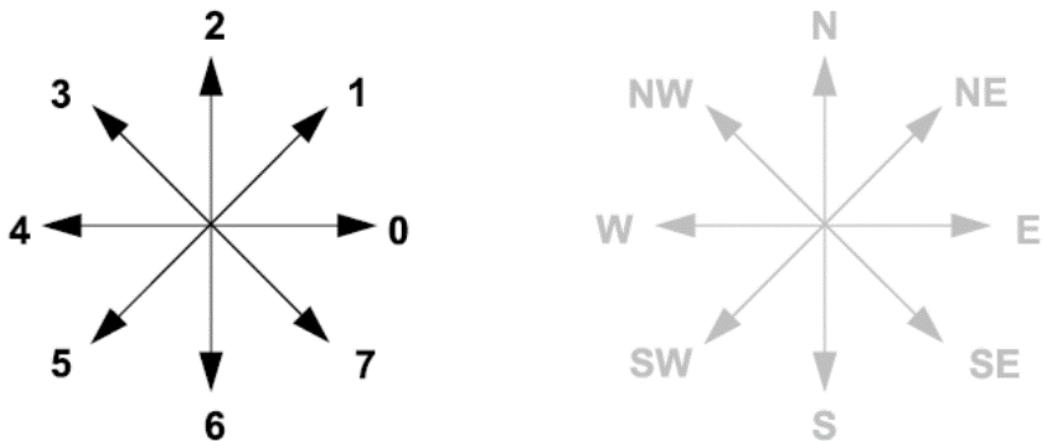
Detekce hran ve snímku (detekce v definovaném směru, směr maximálního gradientu, metody masek, Laplaceův operátor, detekce čáry a bodu, speciální hranové detektory — Canny, Marr-Hildreth, viz literatura).

## 2 Teoretický popis metod

Tato kapitola se věnuje teoretickému popisu jednotlivých metod detekce hran ve snímku.

### Detekce v definovaném směru

Hrany je možné detektovat v různých směrech. Typickou konvenci pro kódování směru je možné vidět na Obr. 1.



Obr. 1: Kódování směrů

Typicky se hrany detekují použitím 1. derivace obrazové funkce. Derivace je nahrazována diferencí – rozdíl jasových hodnot dvou sousedních pixelů.

## Směr maximálního gradientu

Metoda spočívá ve výpočtu vektoru gradientů v každém pixelu pomocí konvoluci. Následně se v každém pixelu spočítá maximální gradient. Částečné derivace při výpočtu je opět možné nahradit za diference.

## Metody masek

Aplikováním masky na obrázek je možné detektovat hrany. Masky musejí obecně musejí splňovat podmínu, že součet jednotlivých prvků masky je roven nule.

Mezi známé masky patří:

- Prewittové operátor:  $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$  a  $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$
- Sobelův operátor:  $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$  a  $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
- Robinsonův operátor:  $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{bmatrix}$  a  $\begin{bmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix}$
- Kirschův operátor:  $\begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$  a  $\begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix}$
- Robertsův operátor:  $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$  a  $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$

## Laplaceův operátor

Laplaceův operátor na rozdíl od předchozích metod neurčuje směr hrany, ale určuje pouze její velikost. Typicky se používají dvě váhové funkce, které je možné vyjádřit maticovým zápisem následovně:  $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$  a  $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

## **Detekce čáry a bodu**

Detektovat čáry v obrazu je možné opět pomocí masek, ale omezuje se tím úhel mezi detekovanou čárou a vodorovnou hladinou. Nabízí se vhodnější řešení – Houghova transformace. Transformace využívá právě zmiňovanou směrnici přímky a vyjadřuje je pak ve tvaru pomocí trigonometrických rovnic

Detekce bodu je snadná za použití znormované Laplaceovy masky:

$$\begin{bmatrix} -1/8 & -1/8 & -1/8 \\ -1/8 & 1 & -1/8 \\ -1/8 & -1/8 & -1/8 \end{bmatrix}$$

Po aplikaci masky je vhodné užít prahování – při maximální hodnotě je tato maska schopna odhalit pouze body velké 1 pixel v binárním obrazu.

## **Canny**

Tato komplexní metoda detekování hran se dá popsat pěti kroky:

1. Aplikace Gaussovského filtru pro rozmazání obrazu a vyhlazení šumu
2. Nalezení gradientů v obrazu
3. Aplikace detekce hran pomocí směru maximálního gradientu s prahováním pro potlačení lokálně maximálních hodnot
4. Aplikace dvojího prahování pro nalezení potencionální hranice
5. Finalizace hran – potlačení slabých hodnot, resp. napojení na silné hodnoty

## **Marr-Hildreth**

Tato metoda detekování hran se dá popsat dvěma kroky:

1. Aplikace LoG operátoru (Laplacian of Gaussian)
2. Detekce hran pomocí nulových bodů

## 3 Implementace

Aplikace je napsána v jazyce Python. Potřebné knihovny pro správný běh aplikace jsou numpy, opencv-python, imgui[glfw] a wxPython. Spustitelný skript se jmenuje **main.py**.

### 3.1 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní je implementováno převážně za pomoci knihovny Dear ImGui. Jelikož knihovna nenabízí žádné řešení dialogových oken s přímým přístupem na disk, je ukládání a načítání obrázků řešeno pomocí knihovny wxWidgets.

Po spuštění aplikace se načte okno, které obsahuje hlavní menu. Položky tohoto menu jsou File, Edit, Settings a About. Nabídka File dále umožňuje vytvořit si nový projekt, načíst obrázky z disku, uložit obrázky na disk a ukončit aplikaci. Edit nabízí možnost detekovat hrany v obrázku, rozmazat obrázek a prahovat obrázek. V Settings jsou grafická nastavení – velikost okna a barevný styl. Položka Help pouze nabízí About.

### 3.2 Předzpracování a následné zpracování obrazu

Mimo hlavní téma této práce byla implementována možnost obrázky rozmazávat a prahovat. Využití těchto možností je vhodné hlavně u detekování čar a bodů.

Pro rozmazání byl vybrán Gaussovský kernel, uživatel si může zvolit jeho velikost.

Pro prahování je možné si práh ručně vybrat na posuvníku nebo využít přepínač na Otsu prahování.

### 3.3 Detekce hran

Po každé operaci je vytvořen nové modální okno do hlavního okna. Toto modální okno drží ve svém názvu veškerou historii použitých operací v takovém pořadí, v jakém byly užity. Byla navíc implementována detekce kolizí v názvech, aby se tak předešlo kolizím při ukládání výsledků na disk.

## **Detekce v definovaném směru**

Implementace této části se liší od popisu v teoretické části. Hlavním důvodem je, aby se nepoužívaly diference zbytečně dvakrát. Možné směry jsou horizontální, vertikální a oba. Detekce hranice neprobíhá výpočtem první derivace, ale používají se masky. Dostupné masky jsou Sobelův operátor, Prewittové operátor a Robertsův operátor.

## **Směr maximálního gradientu**

Implementace směru maximálního gradientu je v souladu s teoretickou částí. Uživatel si může zvolit zda chce první derivaci nahradit dopřednou differencí, zpětnou differencí nebo centrální differencí. Nabízené směry jsou opět horizontální, vertikální a oba naráz.

## **Metody masek**

Tato část se opět liší od teoretického popisu, neboť zmiňované masky již byly užity v detekci v definovaném směru. Tato část byla implementována tak, že uživatel si může zvolit velikost masky – nabízené velikosti jsou 2, 3 nebo 5. Uživatel může jednotlivé prvky masek měnit libovolně, je tak možné s touto částí provádět různé experimenty.

## **Laplaceův operátor**

Laplaceův operátor je implementován podle teoretického popisu. V grafickém uživatelském rozhraní je pro uživatele implementována možnost volby mezi 4–okolím a 8–okolím (kříž a čtverec).

## **Detekce čáry a bodu**

Detekce čáry je implementována za pomoci knihovny OpenCV a probabilistické Houghovy transformace. Pro správné fungování detekce čar je vhodné snímek nejprve naprahat na binární obrázek. Výsledné detekované čáry jsou vykresleny červeně do původního obrazu.

Detekce bodu odpovídá teoretickému popisu, je tak pro uživatele umožněno volit práh pomocí posuvníku. Detekce bodu probíhá užitím znormovaného Laplaceova operátoru.

## Canny

Cannyho detekce hran je implementována dvojím způsobem pro možnosti srovnání výsledků.

První způsob implementace je vlastní implementace. Nejprve je na obrázek aplikován Gaussovský filtr pro rozmazání, je tak umožněno uživateli volit parametr sigma, který je užit při generování Gaussovské matice. Následně jsou nalezeny jednotlivé gradienty v horizontálním a vertikálním směru. Potom je užito potlačování lokálních maxim na základě okolí pixelu a úhlů. Nakonec je užito dvojí prahování pro zvolení slabých a silných hran.

Druhý jednodušší způsob implementace je užití OpenCV implementace. Tato implementace je podstatně efektivnější, neboť OpenCV přímo volá funkce z jazyka C++.

## Marr-Hildreth

Marr-Hildreth detekce hran nejprve aplikuje LoG (Laplacian of Gaussian) operátor na obraz. Následně se volí hrany na základě nulových bodů. Pro každý pixel se kontroluje jeho jasová hodnota, resp. znaménko jasové hodnoty, v relaci s jeho sousedy.

## 4 Experimenty a výsledky

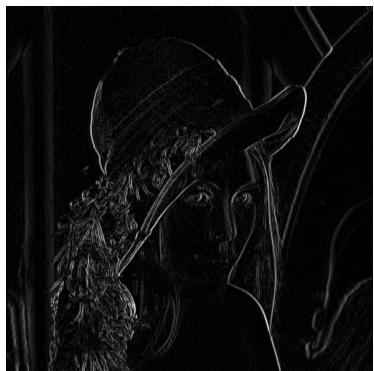
Veškeré experimenty byly provedeny na osobním počítači s operačním systémem Windows 10 Home. Počítač obsahuje procesor AMD Ryzen 5 4600H 3.00 GHz, 16 GB RAM a grafickou kartu NVIDIA GeForce GTX 1660 Ti.

Základním obrázkem pro experimenty je Obr. 2 – jedná se o známý obrázek „Lenna“, který má  $512 \times 512$  pixelů.

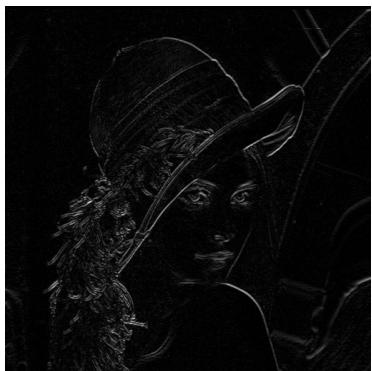


Obr. 2: Základní obrázek pro experimenty

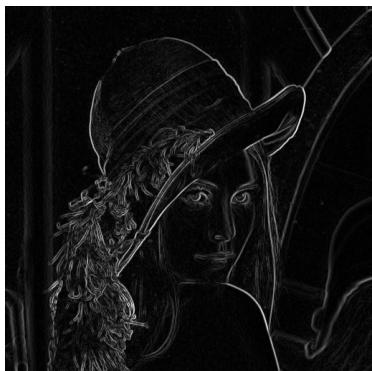
## Detekce v definovaném směru



Obr. 3: Sobelův operátor – horizontálně



Obr. 4: Sobelův operátor – vertikálně



Obr. 5: Sobelův operátor – oba směry



Obr. 6: Prewittové operátor – horizontálně



Obr. 7: Prewittové operátor – vertikálně



Obr. 8: Prewittové operátor – oba směry



Obr. 9: Robertsův operátor – horizontálně



Obr. 10: Robertsův operátor – vertikálně



Obr. 11: Robertsův operátor – oba směry

## Směr maximálního gradientu



Obr. 12: Zpětná differenční horizontálně



Obr. 13: Zpětná differenční vertikálně



Obr. 14: Zpětná differenční obou směrů



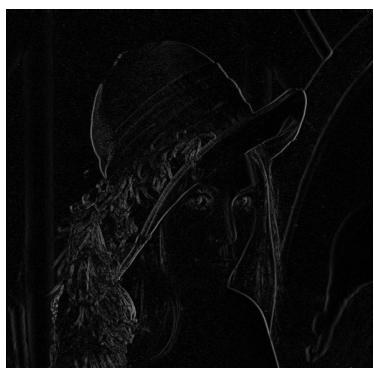
Obr. 15: Dopředná differenční horizontálně



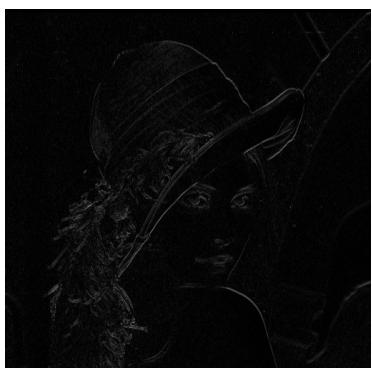
Obr. 16: Dopředná differenční vertikálně



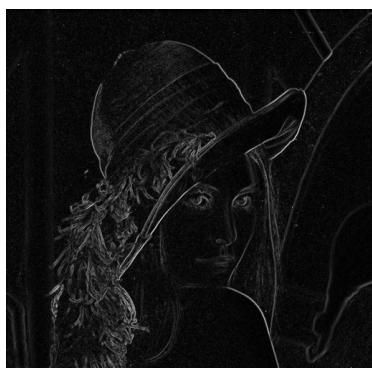
Obr. 17: Dopředná differenční obou směrů



Obr. 18: Centrální differenční horizontálně

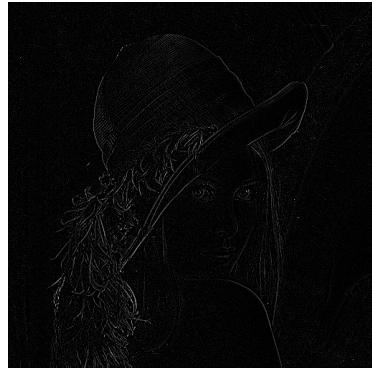


Obr. 19: Centrální differenční vertikálně



Obr. 20: Centrální differenční obou směrů

## Laplaceův operátor

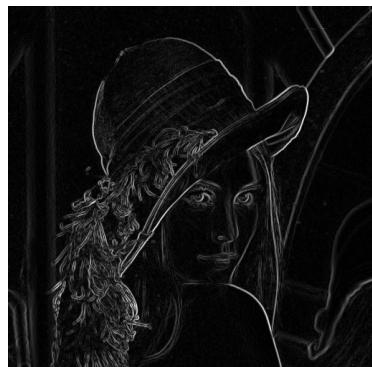


Obr. 21: Laplaceův operátor 4-okolí



Obr. 22: Laplaceův operátor 8-okolí

## Detekce čáry a bodu



Obr. 23: Základ obrázku pro detekci čar a bodů (Sobelův operátor v obou směrech)



Obr. 24: Obrázek pro detekci čar a bodů – Otsu prahování



Obr. 25: Detekce čar  
(probabilistická verze)



Obr. 26: Detekce bodů  
(práh 240)



Obr. 27: Detekce bodů  
(práh 100)

## Canny



Obr. 28: Vlastní  
implementace Cannyho  
detekce hran



Obr. 29: OpenCV  
implementace Cannyho  
detekce hran

## Rychlost konvergence

Testovaný obrázek má  $512 \times 512$  pixelů. Vlastní implementace v průměru vygeneruje výsledek do 5.25s. Implementace od OpenCV výsledek generuje do 0.003s. Rozdíl je tak vysoký z důvodu, že OpenCV implementace má optimalizované matematické operace v C++, kdežto vlastní implementace využívá pouze jazyk Python a jeho konstrukce.

## Marr-Hildreth



Obr. 30: Marr-hildreth detekce hran

## 5 Závěr

Cílem semestrální práce bylo implementovat aplikaci pro detekci hran v obrázcích. Implementovány byly všechny metody ze zadání. Výstupy jednotlivých metod odpovídají očekávání.

Vylepšení do budoucna by mohlo být přidání více možností jak k detekci hran, tak k předzpracování, resp. následnému zpracování obrazů. Dalším vylepšením by mohla být optimalizace vlastních implementací, aby se více vyrovnaly knihovním implementacím.