Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Dekonvoluce jednokanálových snímků pro systém s grafickou kartou podporující CUDA

Bakalářská práce

Autor: Ladislav Zítka
Studijní obor: Aplikovaná informatika

Vedoucí práce: prof. RNDr. PhDr. Antonín Slabý, CSc.

Hradec Králové Duben 2024

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 24.4.2024

vlastnoruční podpis

Poděkování:
Děkuji vedoucímu mé bakalářské práce profesoru Antonínu Slabému za metodické vedení a RNDr. Janu Horáčkovi za poskytnutí příležitosti a zdrojů pro vypracování této práce.

Abstrakt

Dekonvoluce je tématem spadajícím pod mnoho vědních oborů, jako jsou seismologie, radioastronomie, biologie, zdravotnictví a zpracování obrazu. Bakalářská práce se zabývá právě tématikou dekonvoluce na jednokanálových snímcích, specificky testováním časové náročnosti a efektivity při použití různých metod dekonvoluce s různými typy šumu. Teoretická část se zabývá základními znalostmi o Fourierově transformaci, konvoluci, Lucy-Richardson dekonvoluci, Wienerově dekonvoluci a vlastnostech šumu. V praktické části jsou zobrazeny výsledky dekonvolucí s popisem a poznatky. Závěrem práce je shrnutí výsledků a poznatků získaných z praktické části. Implementace algoritmů je provedena v jazyce C++ s využitím API CUDA a API cuFFT. Každá implementace má také jednodušší implementaci bez využití paralelizace a GPU.

Abstract

Title: Deconvolution of single-channel images for system with CUDA-enabled graphics card.

Deconvolution is a topic concerning many fields of study, such as seismology, radioastronomy, biology, healthcare and image processing. This bachelor's thesis is dealing with the topic of deconvolution of mono-channel images, more specifically with the time complexity and effectivity of different deconvolution methods with different forms of noise. The theoretical part of the thesis will introduce basics needed to understand Fourier transform, convolution, Lucy-Richardson deconvolution, Wiener deconvolution and forms of noise. The practical part is going to describe results and information gained from testing. The conclusion will then summarize these results. Implementation of algorithms has been done in C++ with CUDA API and cuFFT API. There is also a simplified implementation in CPU alongside the GPU ones.

Klíčová slova:

Dekonvoluce, Wienerova dekonvoluce, Lucy-Richardson dekonvoluce, Fourierova transformace, PSF (rozptylová funkce) / kernel, cuFFT, CUDA

Key words:

Deconvolution, Wiener deconvolution, Lucy-Richardson deconvolution, Fourier transform, PSF (point spread function) / kernel, cuFFT, CUDA

Obsah

1 Úvod

[1] Ve světě rekonstrukce obrazu existuje mnoho metod pro rekonstrukci, které se mohou dělit podle typu známých informací nebo vad. Pro vady na obraze ve formě aditivního šumu se často využívají různé filtry. Primitivnější filtry mohou snížit šum rozmazáním nebo nahrazením hodnotou v okolí. Komplexnější adaptivní filtry ale využívají charakteristiku obrazu pro zlepšení výsledků a snížení poškození původního obrazu.

Tato práce se ale zaměřuje na rekonstrukci obrazu, na kterém byla nejdříve provedena konvoluce, a následně se k němu přidal šum. Tento proces způsobí, že není možné použít jednoduchou inverzi konvoluce, protože šum nebyl degradován konvolucí. To vede u nízkých hodnot v konvolučním kernelu k zesílení šumu a potenciálně i ke kompletní ztrátě informací v obraze. Proto se pro dekonvoluci používají komplikovanější metody.

Zejména to jsou Wienerova a Lucy-Richardson dekonvoluce. Pro obě tyto metody platí, že známe degradovaný obraz a konvoluční kernel. Metody, u kterých kernel neznáme, se nazývají slepé. Slepé metody se většinou snaží získat PSF pomocí estimace, která může být i iterativní. [2] Pro dekonvoluci se v moderní době také používají machine a deep learning, které umožňují automatizaci procesu. Ty mívají formu konvolučních neuronových sítí nebo generativních modelů.

Výpočty v těchto metodách se často opírají o vlastnosti Fourierovy transformace, primárně [3] větu o konvoluci. Z té je pro dekonvoluci důležitá informace, že konvoluce dvou funkcí v časové doméně odpovídá násobku jejich formy ve frekvenční doméně. Zasazeno do kontextu této práce to znamená, že násobek Fourierových transformací obrazu a kernelu odpovídá jejich konvoluci.

Implementace těchto metod je provedena v jazyce C++. Tyto implementace se dělí podle přesnosti datového typu a implementace v GPU nebo CPU. Pro implementaci v GPU se využívá API CUDA a API cuFFT.

Praktická část následně ukazuje výsledky těchto implementací s ohledem na čas, kvalitu a reakci na různé druhy šumu.

2 Cíl a metodika práce

2.1 Cíl práce

Tato bakalářská práce se snaží představit problematiku dekonvoluce při rekonstrukci obrazu a zpřehlednit informace pro implementaci dvou často používaných metod. Podrobně se soustředí na Wienerovu a Lucy-Richardson metodu otestováním při použití různých kernelů, rozlišení a šumu. Praktická část je zaměřena na zobrazení výsledků s poukázáním na problematiku implementace. Cílem práce je také vytvoření knihovny obsahující implementace použitých algoritmů v jazyce C++.

2.2 Metodika

Podklady pro zpracování bakalářské práce jsou založené na analýze odborné literatury převážně zahraničních tištěných a elektronických zdrojů. Využití dekonvoluce pro image processing je jedna z často používaných technik a existuje o ní mnoho článků a odborných knih v angličtině. Tato práce si přebírá základy z tisku odborné knihy [4] Digital Image Processing jejíž autoři jsou Rafael C. Gonzalez a Richards E. Woods. Tato kniha poskytuje některé informace pro pochopení tématu dekonvoluce a výpočtů s konvolucí spojených. Informace neposkytnuté touto knihou jsou většinou dohledány v odborných článcích veřejně dostupných na internetu. Pro pochopení tématu bylo také využito článků na Wikipedii, které byly následně ověřené odbornou literaturou. Většina zdrojů je volně přeložena autorem a pokud využívají informace z odborné literatury, tak jsou řádně citována.

Zaměření této práce je na jednokanálové snímky homogenně poškozené kernelem, s následným přidáním šumu. Metody, kterými jsou v této práci dosaženy výsledky jsou Wienerova dekonvoluce a Lucy-Richardson dekonvoluce. Testování se provádí na obrazech získaných z CT skenu od firmy Lometom s.r.o. Kosmonosy.

Práce začíná vysvětlením technik a metod použité pro uskutečnění práce, jako jsou Fourierova transformace, konvoluce, kernel, metody dekonvoluce a typy šumu.

Vysvětlení je v dostatečné hloubce a návaznosti, aby jednotlivé kapitoly byly pochopitelné z údajů v této práci.

Praktická část bude ze začátku obsahovat informace potřebné pro využití knihovny, předávané společně s prací. Následují výsledky testování byly pořízeny na laptopu Lenovo Legion s procesorem Intel i5-11400H a grafickou kartou NVIDIA GeForce RTX 3060 Laptop.

Správnost výsledků a efektivita výsledků se kontroluje pomocí vizuální inspekce a informací získaných z programu (čas, průměrný rozdíl od původního obrázku, …).

3 Teoretická část

3.1 Fourierova transformace

3.1.1 Historie

[5]Chápání Fourierovy transformace ve stavu, v jakém je známý teď, představil francouzský matematik a fyzik Jean Baptiste Joseph Fourier (1768 - 1830). Poprvé byla transformace rozepsána Fourierem v memoáru „***Théorie analytique de la chaleur***“ (Analytická teorie tepla) z roku 1807. Tento memoár byl později pečlivě obohacen a rozšířen a vydán jako kniha se stejným jménem. Tuto knihu přeložil v roce 1878 do angličtiny Alexander Freeman.

3.1.2 Základy

[6]Základní myšlenka Fourierovy transformace spočívá ve schopnosti rozdělit libovolnou periodickou funkci na sumu sinusoid o různé frekvenci a vynásobenou různým koeficientem, tomuto se říká Fourierova řada. Tento koncept byl ze začátku přivítán skepticismem.

Podobným způsobem lze řešit i neperiodické funkce, pokud mají konečnou plochu pod čarou. V tomto případě se funkce rozdělí na integrály sinusoid vynásobené koeficienty. Právě tomuto procesu se říká Fourierova transformace.

Vzorec Fourierovy transformace pro spojité funkce:

$$F(\mu) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\mu x}\, dx$$

Vzorec zpětné Fourierovy transformace pro spojité funkce:

$$f(x) = \int_{-\infty}^{\infty} F(\mu) e^{i2\pi\mu x}\, d\mu$$

F(μ) – Fourierova transformace funkce f(x)

μ – spojitá proměnná odpovídající počítané frekvenci

x – spojitá proměnná odpovídající hodnotě funkce

Původní využití této transformace bylo pro výpočet šíření tepla. Za poslední století se pokrokem v technologii Fourierova teorie rozběhla do mnoha odvětví průmyslu a vědy.

Příchod digitálních počítačů a objev algoritmu rychlé Fourierovy transformace (FFT) důležitost této teorie zvýšil ještě více, obzvlášť v oboru zpracování signálu různého druhu.

3.1.3 Samplování

[7]Pro zpracování na počítači není spojitá funkce vhodná, je tedy potřeba spojitou funkci převést samplováním na sekvenci diskrétních hodnot. Tento proces má většinou podobu sbírání hodnot funkce po konstantním intervalu.

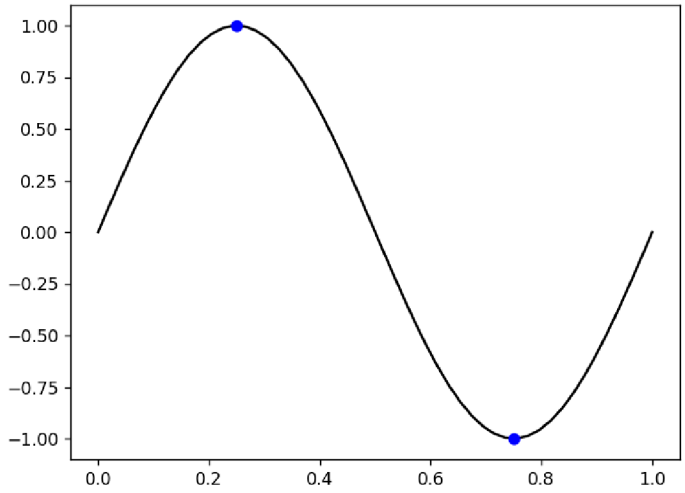
Matematicky lze tento problém počítat pomocí následujícího vzorce:

$$\tilde{f}(t) = \sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta T)$$

$\tilde{f}$ – samplovaná funkce
f(t) – spojitá funkce
$\delta(t - n\Delta T)$ – hodnota funkce na místě impulzu
ΔT – délka intervalu

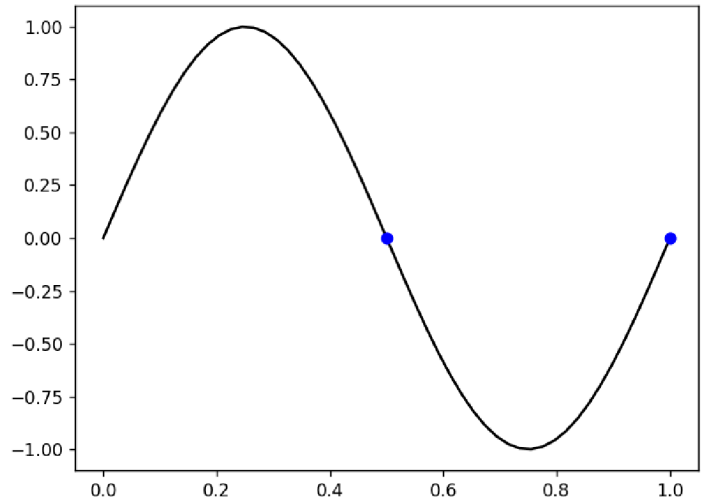
Jeden z problémů samplování je aliasing. Pro nasamplovanou funkci je nekonečně mnoho funkcí s postupně vyšší frekvencí, které budou odpovídat samplovanému vzorku. Je proto potřeba omezit testované frekvence. Podle Nyquistova limitu je potřeba mít tempo samplování více než dvakrát vyšší než nejvyšší měřená frekvence.

Teoreticky je možné při omezení identifikovat frekvenci pomocí pouze dvojnásobné samplovací frekvence.



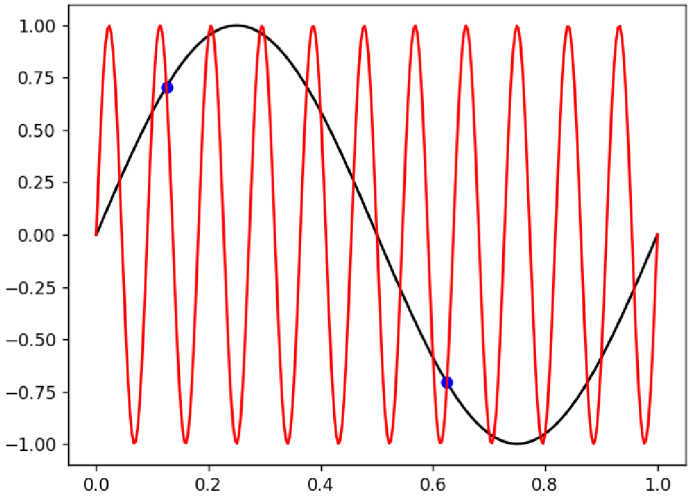
Obrázek 1 - sinusoida s frekvencí 1Hz a samplovací frekvencí 2Hz [zdroj: autor]

Může ale nastat problém, kdy samplování bude odpovídat nulté frekvenci (konstantní funkci).



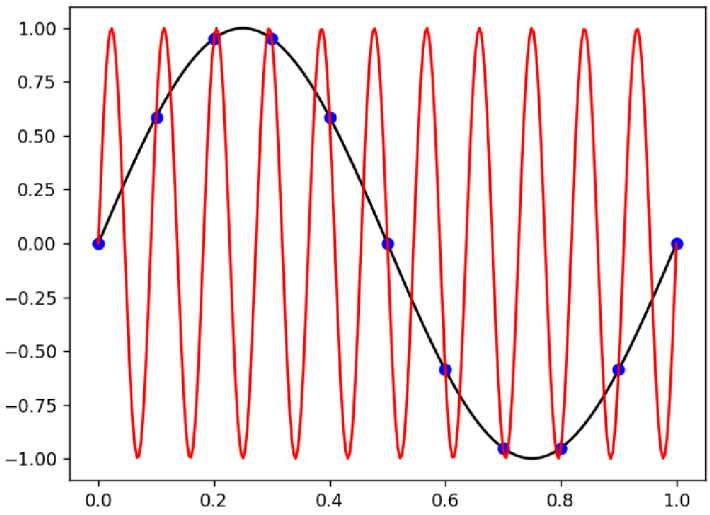
Obrázek 2 - Sinusoida s frekvencí 1Hz a špatně zvoleném offsetu samplovaní o frekvenci 2Hz [zdroj: autor]

Z tohoto důvodu by samplovací frekvence měla být více než dvakrát vyšší.



Obrázek 3 - Černou barvou zvýrazněná sinusoida o frekvenci 1Hz, červenou barvou sinusoida o frekvenci 11Hz a modře samplování o frekvenci 2Hz [zdroj: autor]

Za těchto okolností vzniká aliasing z důvodu neomezení vyšších frekvencí a vzorky sinusoidy s frekvencí 1Hz odpovídají také vzorkům sinusoidy s frekvencí 11Hz.



Obrázek 4 - Černou barvou zvýrazněná sinusoida o frekvenci 1Hz, červenou barvou sinusoida o frekvenci 11Hz a modře samplování o frekvenci 10Hz [zdroj: autor]

V tomto případě dochází k aliasingu, protože samplovací frekvence je nižší než Nyquistův limit. Samplování tak odpovídá také frekvenci nižší.

3.1.4 Diskrétní Fourierova transformace

[8]Pokud se na spojité funkci provede samplování, je možné použít diskrétní Fourierovu transformaci. V tomto případě se pracuje s konečným počtem hodnot, který může zastupovat část spojité funkce.

Vzorec diskrétní Fourierovy transformace je derivován ze vzorce spojité transformace se substitucí výše zmíněného vzorce samplování za transformovanou funkci.

$$F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-i2\pi\mu t}\, dt$$

$$F(\mu) = \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta T)\, e^{-i2\pi\mu t}\, dt$$

$$F(\mu) = \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} f(t)\delta(t - n\Delta T)\, e^{-i2\pi\mu t}\, dt$$

$$F(\mu) = \sum_{n=-\infty}^{\infty} f_n e^{-i2\pi\mu\Delta T}$$

Na charakterizaci Fourierovy transformace stačí jeden cyklus samplované funkce a s tímto faktem se derivovaní funkce upraví do finální verze vzorce pro diskrétní Fourierovu transformaci.

$$F(u) = \sum_{n=0}^{M-1} f(x) e^{-i2\pi ux/M} \quad u = 0,1,2,\dots,M-1$$

Pro zpětnou transformaci:

$$f(x) = \frac{1}{M} \sum_{n=0}^{M-1} F(u) e^{i2\pi ux/M} \quad x = 0,1,2,\dots,M-1$$

F(u) – hodnota Fourierovy transformace pro frekvenci u

F(x) – hodnota diskrétní funkce pro x

M – celkový počet samplovaných prvků

3.1.5 Vztah Fourierovy transformace a komplexních čísel

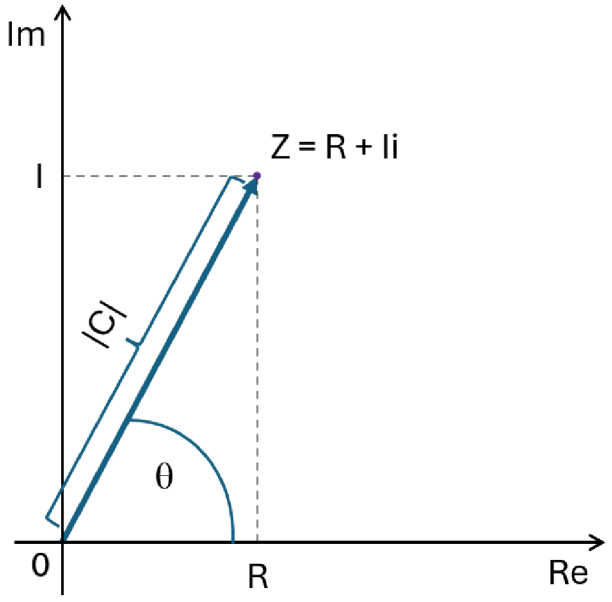
[3] Výsledkem Fourierovy transformace jsou komplexní čísla pro každou počítanou frekvenci. Z těchto komplexních čísel je možné odvodit dvě pole reálných hodnot, kde první odpovídá magnitudě vyskytovaných frekvencí a druhé fázi těchto frekvencí.

Vzorec magnitudy $$|C(u,v)| = \sqrt{R(u,v)^2 + iI(u,v)^2}$$

Vzorec fáze $$\theta(u,v) = \arctan\left[\frac{I(u,v)}{R(u,v)}\right]$$

R – reálná složka komplexního čísla
I – komplexní složka komplexního čísla

Tyto výpočty vychází z převodu komplexního čísla do polární soustavy souřadnic. U výpočtu arctan je potřeba počítat s celým rozsahem [-π, π], a tak se musí řešit pomocí 4 kvadrantových arctangent funkcí nebo vlastního hlídání znamének.



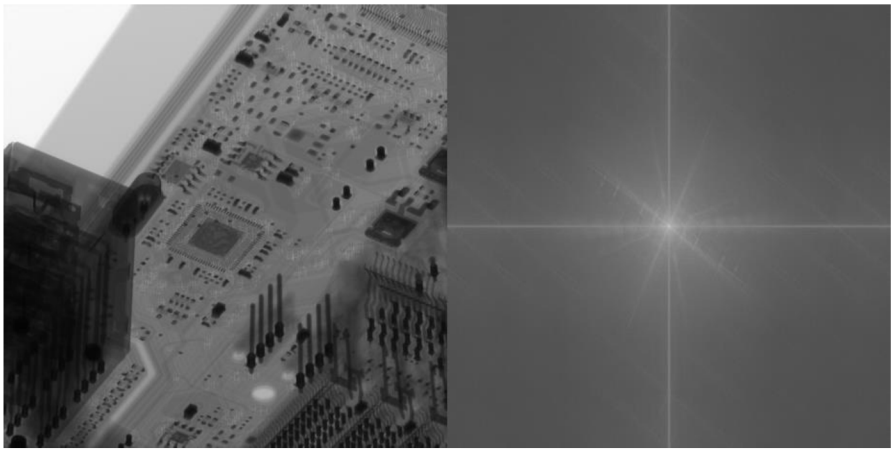
Obrázek 5 - Zobrazení komplexního čísla v kartézské soustavě souřadnic [zdroj: autor]

Tento proces je také možné udělat zpětně, z polární do kartézské. Což umožňuje zaměnit fázi nebo magnitudu za jinou z jiného zdroje, dokud mají stejné rozměry (příklad ukázán v následující sekci).

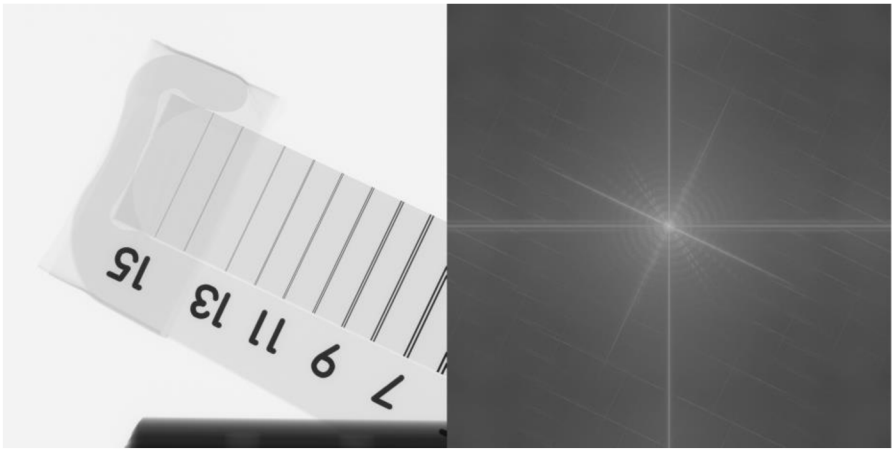
3.1.6 Centralizace pole magnitud a fáze

Obraz fáze většinou mívá formu vizuálně připomínající šum a vizuální inspekcí předává málo informací. Proto se většinou vizuální inspekce provádí na obrazu magnitud. Ten má v neupravené verzi podobu, kde vlevo nahoře (pozice [0;0]) má hodnotu, která je součtem vypočítaných funkcí, a od toho bodu se obraz dále rozvíjí.

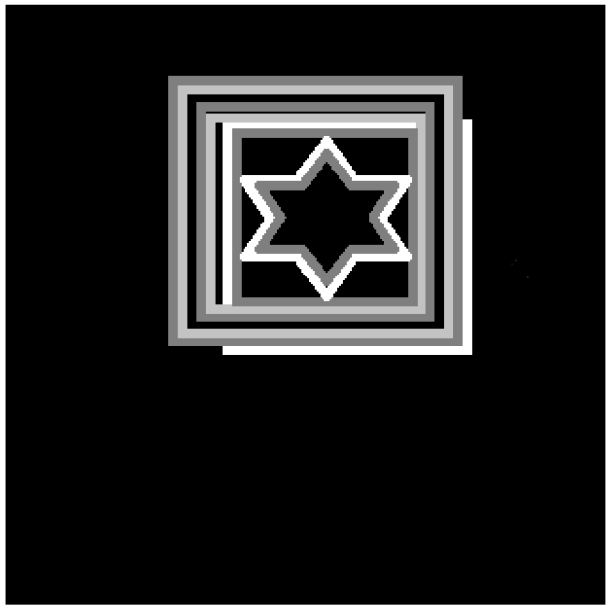
Toto je ale méně přehledné, než může být. Pro přehlednější podobu se většinou obraz centralizuje. Dvě jednoduché možnosti pro centralizaci jsou manuální přesun kvadrantů tak, aby pozice [0;0] byla ve středu obrazu. [9]Druhá možnost je před výpočtem přenásobit původní obraz $(-1)^{xy}$. Tímto je možné získat přehledný centrovaný obraz magnitud.



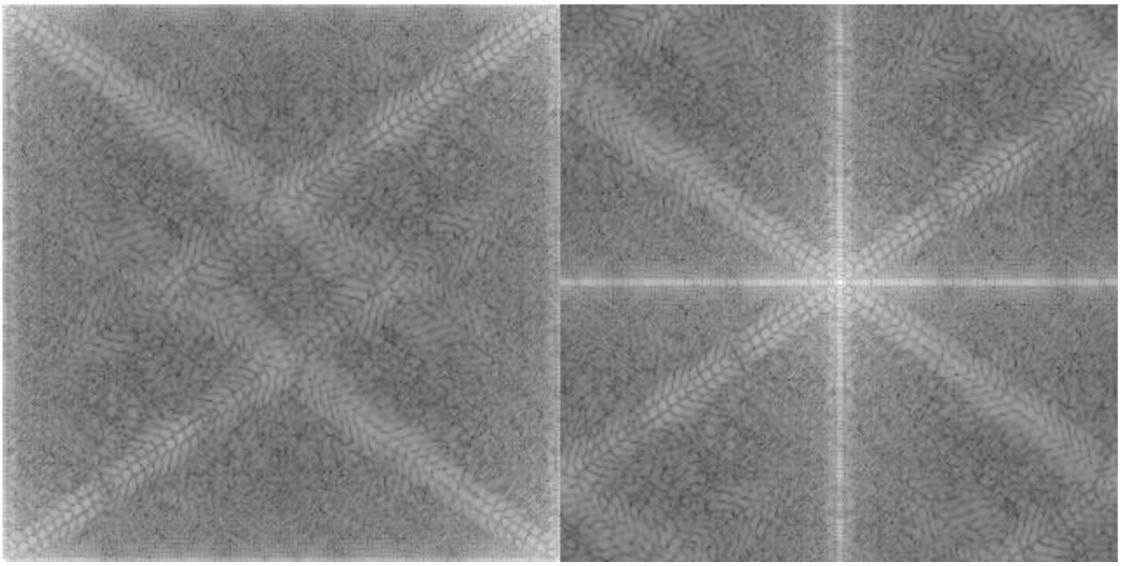
Obrázek 6 - příklad magnitudy na CT skenu motherboardy [zdroj: autor]



Obrázek 7 - příklad magnitudy na CT skenu dvoudrátkové měrky [zdroj: autor]



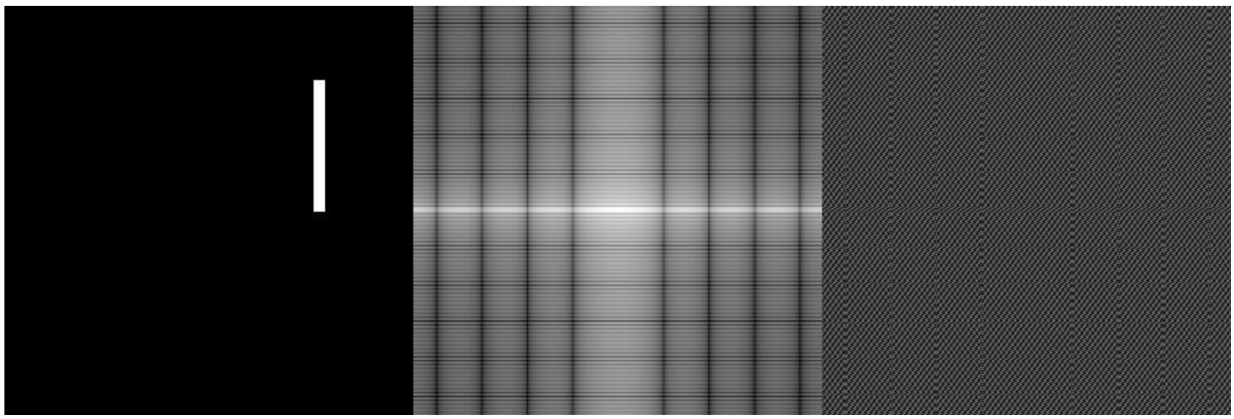
Obrázek 8 - Původní obrázek [zdroj: autor]



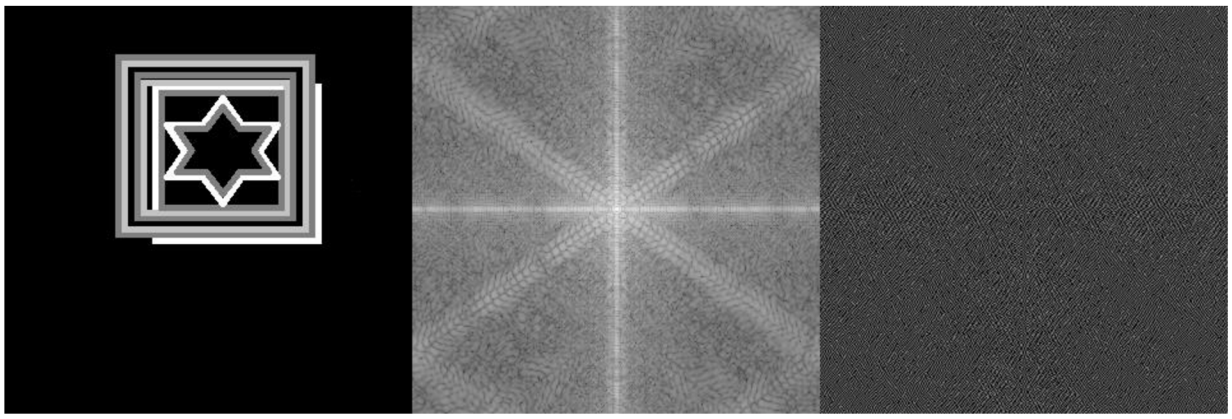
Obrázek 9 - Necentralizovaný obraz magnitud vlevo, Centralizovaný obraz magnitud vpravo [zdroj: autor]

Z centralizovaného obrazu magnitud je možné poznat několik vlastností. Jednou z nich je symetrie podél diagonály. Druhou jsou vyšší hodnoty frekvencí kolmé na hrany v původním obraze. V případě, ve kterém by se by se nacházel periodický typ šumu, je možné tento šum vidět jako zvýšený výskyt určitých frekvencí. Tento šum je následně možné utlumit snížením hodnot těchto frekvencí. Zasahování do Fourierova spektra je ale globální operace a může vést k ovlivnění i důležitých informací s podobnou charakteristikou jako tlumený šum.

3.1.7 Vliv magnitudy a fáze v obraze

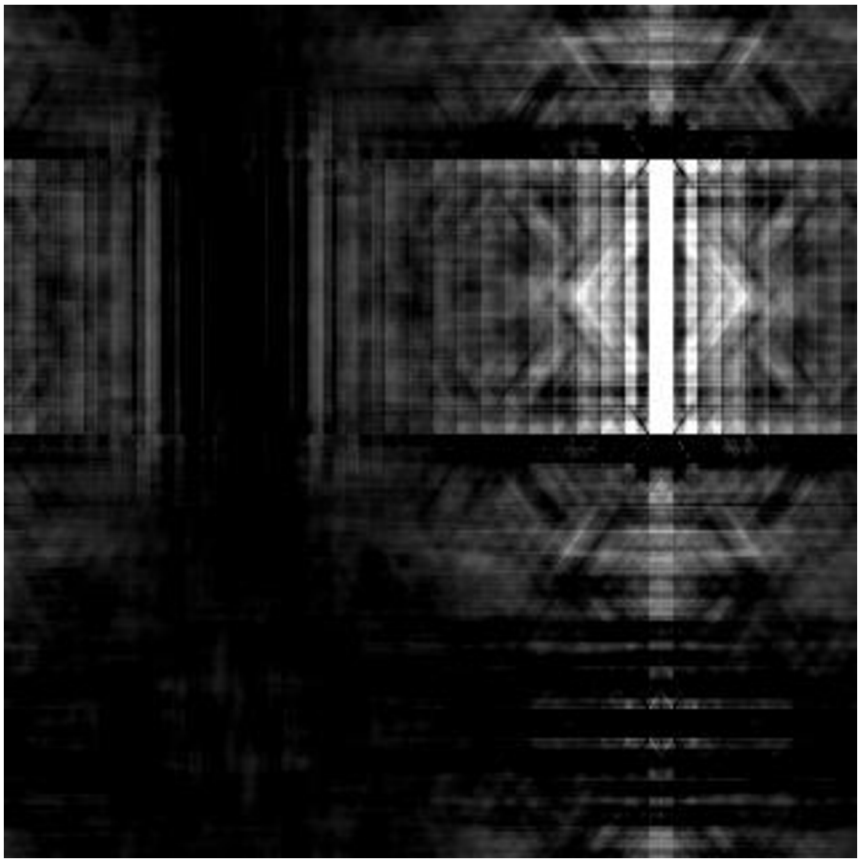


Obrázek 10 - původní obrázek vlevo, vizualizace magnitud uprostřed, vizualizace fáze vpravo [zdroj: autor]



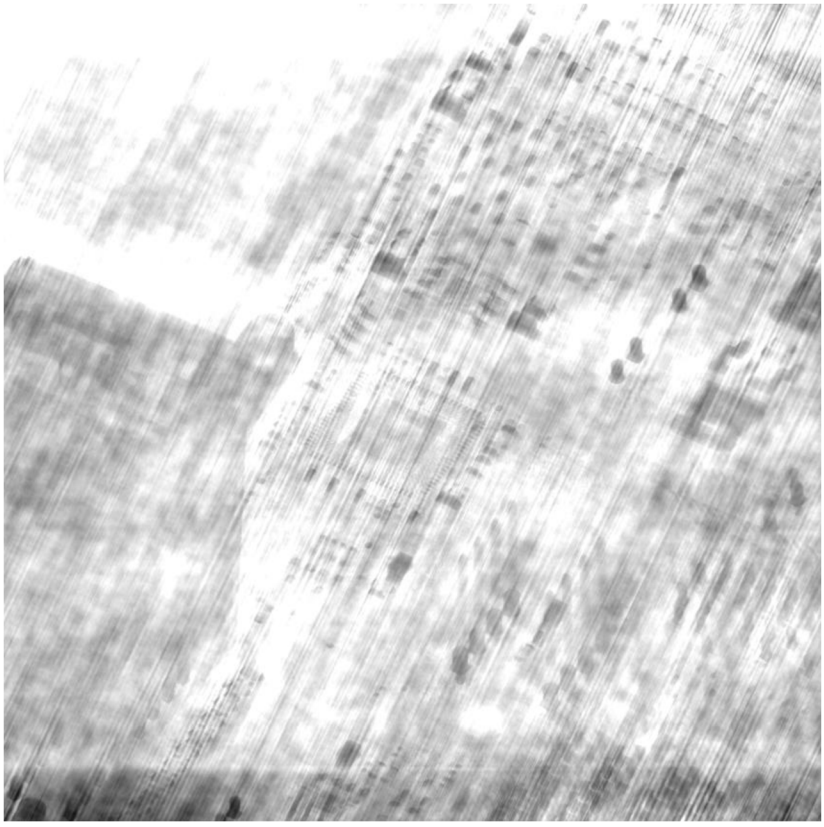
Obrázek 11 - původní obrázek vlevo, vizualizace magnitud uprostřed, vizualizace fáze vpravo [zdroj: autor]

Z rozkladů těchto dvou obrázků na magnitudu a fázi je možné pochopit, že pro lidské pochopení je jednodušší zobrazení magnitud. Na druhou stranu obraz fáze často vypadá jako šum. Z následujících obrázků, u kterých je vzájemně prohozena fáze, je ale prokázáno že důležité informace, jako jsou tvary a hrany, jsou uleženy spíše ve fázi.

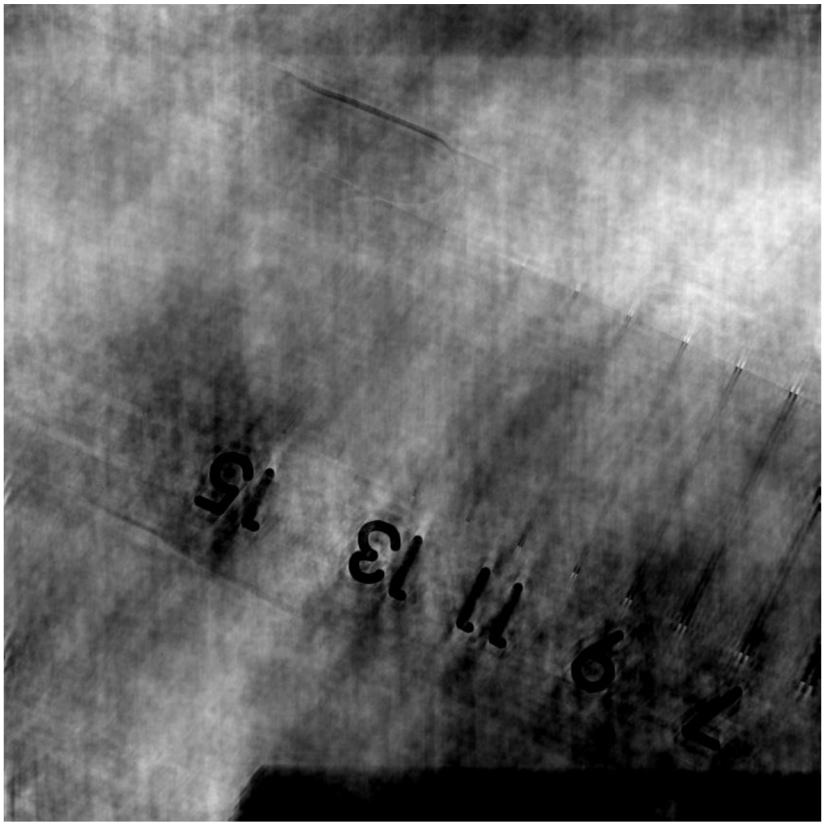


Obrázek 12 - Fáze jedné čáry s magnitudou hvězdy v boxu [zdroj: autor]

Obrázek 13 - Fáze hvězdy v boxu s magnitudou čáry [zdroj: autor]



Obrázek 14 - Fáze motherboardy s magnitudou měrky [zdroj: autor]



Obrázek 15 – Fáze měrky s magnitudou motherboardy [zdroj: autor]

3.2 Kernel

3.2.1 Úvod

Kernel neboli konvoluční matice je součástí konvolučního procesu. Tento pojem může být někdy zaměnitelný s pojmem PSF (rozptylová funkce). Ten se více využívá s otázkou optiky nebo odhadování při slepé dekonvoluci. Z pohledu teorie je to pouze druhá funkce, se kterou se provede operace zvaná konvoluce. Následující sekce se zaměřují na využití v oboru zpracování obrazu.

3.2.2 Typické vlastnosti

Změnou obsahu kernelu je možné dosáhnout mnoho různých výsledků. V pozdější sekci jsou některé obecně známé kernely ukázány s výsledky, ale většinou mají několik typických vlastností.

Jednou z prvních je, že suma hodnot v kernelu se rovná jedné. Tímto se udržuje relativně konstantní průměrná hodnota obrázku. Toto často vede k obecně nízkým hodnotám ve velkých kernelech, například pro box blur o počtu prvků x by každý prvek matice měl hodnotu 1/x. V kernelech o rozměrech 4096x4096 by tedy každá hodnota byla 1/16 777 216.

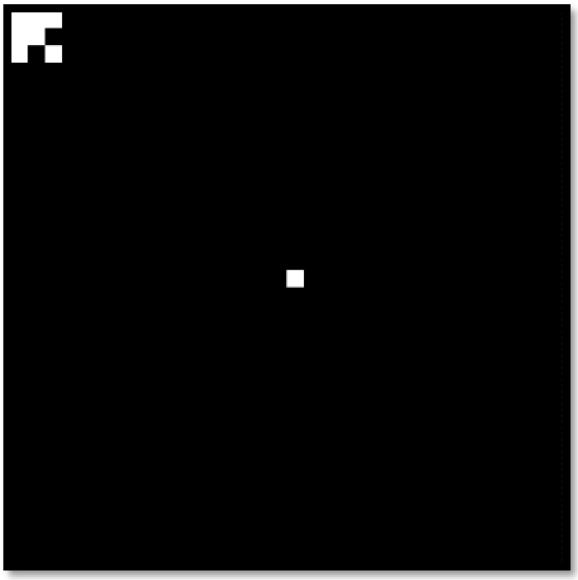
Další vlastností je počátek. V případě symetrických kernelů to často bývá střed.

3.2.3 Vliv počátku kernelu

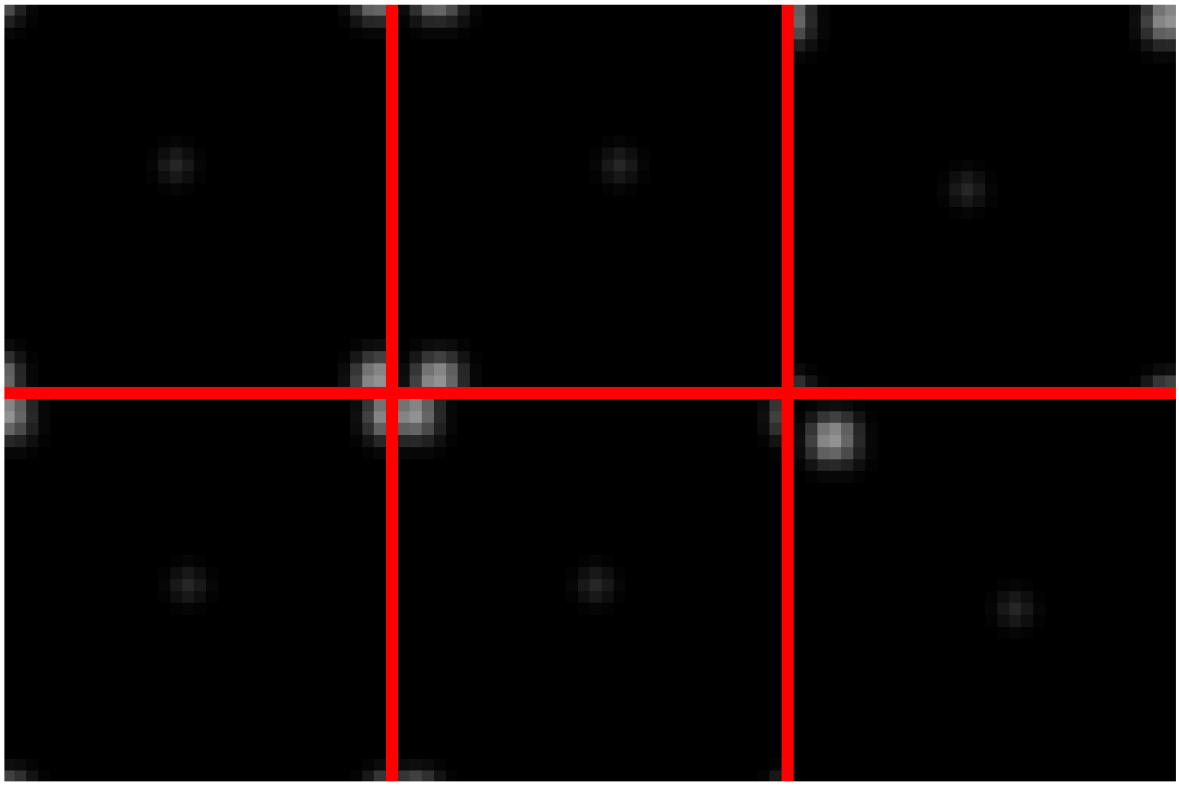
$$\frac{1}{256} * \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Obrázek 16 - Aproximace Gauss kernelu s rozměry 5x5 [zdroj: autor]

Na následujících obrázcích je ukázáno, jaký vliv má pozice počátku na výsledek konvoluce při konvoluci výše znázorněným Gaussovským kernelem. Číslování pozic začíná v levém horním rohu nultou pozicí. Jedná se o prostý výpočet konvoluce, při kterém je výpočet mimo obrázek řešen wrapováním (při překročení řádku na konci se hodnota bere z jeho začátku).



Obrázek 17 - originální obrázek pro testování počátku s přidaným tmavým ohraničením [zdroj: autor]



Obrázek 18 – sada šesti transformací, které se liší pouze počátkem [zdroj: autor]

Na výše znázorněném obrázku je možné vidět jaký vliv počátek na transformaci. Pozice jak v obrázcích, tak v kernelu se začíná počítat od levého horního rohu. Obrázky byly počítány s počátky 0, 4, 10, 11, 12, 24. Obrázky jsou srovnány vzestupně s nejnižším (nultá pozice počátku) v levém horním a postupně zvyšovanou pozicí směrem doprava.

Tyto specifické hodnoty byly vybrány pro pochopení chování kernelu v krajních případech, kdy se počátek nachází u krajů kernelu.

V případě nulté pozice počátku (levý horní obrázek) se počátek nachází v levém horním rohu kernelu. Toto vede k tomu, že intenzita pixelu po transformaci se počítá z intenzity pixelů v originálním obrázku směrem doprava a směrem dolu. Ve výsledku tato pozice způsobí nejen Gaussovské rozmazání ale také posun směrem doleva nahoru.

Čtvrtá pozice (horní uprostřed) se nachází v pravém horním rohu kernelu Toto ovlivňuje transformaci podobně jako v případu nulté pozice až na to, že posun jde směrem doprava nahoru.

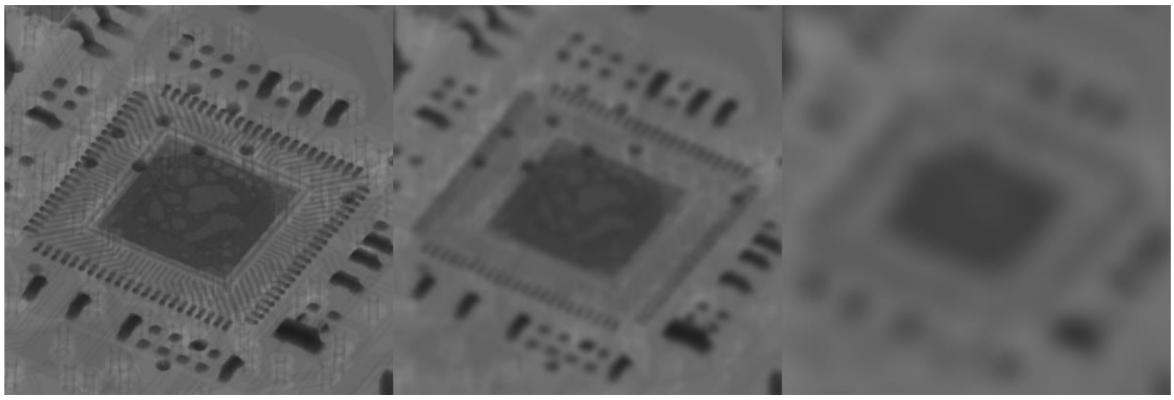
Desátá pozice (horní vpravo) odpovídá začátku třetího (prostředního) řádku kernelu. Tento případ způsobuje pouze posun doleva.

Jedenáctá pozice (spodní vlevo) je tu hlavně jako přechod mezi desátou a dvanáctou pozicí. Vede k posunu o jeden pixel doleva.

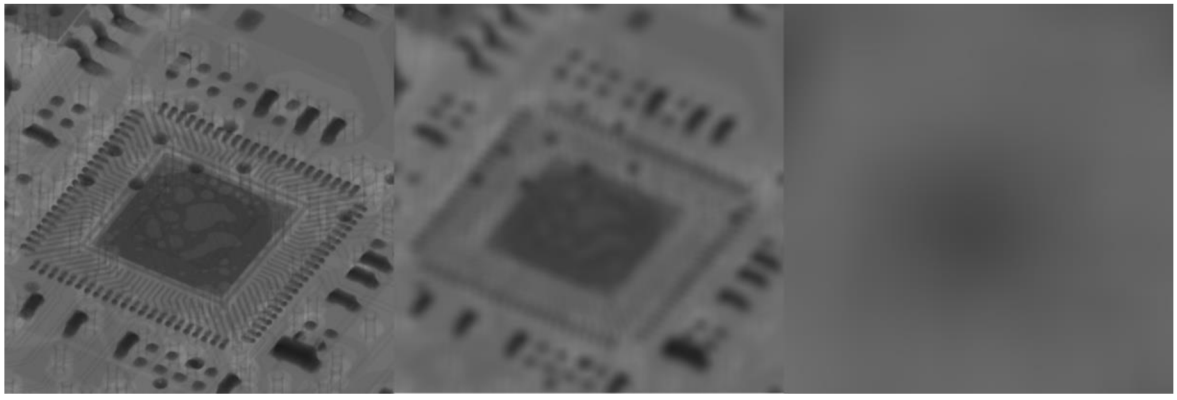
Dvanáctá pozice (spodní uprostřed) odpovídá očekávánému výsledku rozmazání Gaussovským kernelem, kde je kernel symetrický a počátek se nachází uprostřed kernelu.

Čtyřiadvacátá pozice (spodní vpravo) je opakem nulté pozice. Počátek se nachází v pravém spodním rohu kernelu. Vede k posunutí doprava dolu.

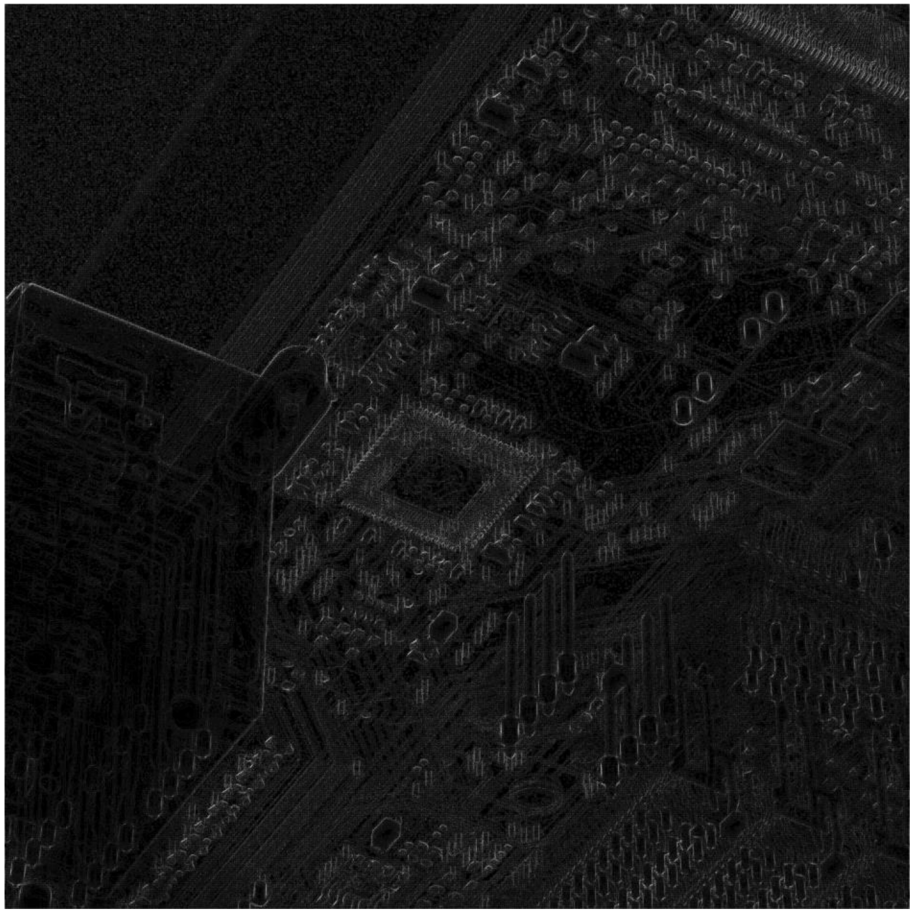
3.2.4 Příklady známých kernelů s efektem



Obrázek 19 - Rozmazání Gaussovským kernelem o velikostech 7x7, 27x27 a 101x101 [zdroj: autor]



Obrázek 20 – Rozmazání kernelem box bluru o velikostech 3x3, 33x33, 333x333 [zdroj: autor]



Obrázek 21 - detekce hran pomocí kernelu upravena zesílením hodnot nálezů [zdroj: autor]

3.3 Konvoluce

3.3.1 Základy konvoluce

[10]Konvoluce dvou funkcí je matematická operace, kde se jedna z funkcí otočí a obě funkce se postupně posouvají přes sebe. Hodnota tohoto překryvu ve specifickém bodě odpovídá integrálu ze součinu těchto dvou funkcí. V tématice zpracování obrazu je obraz brán jako první funkce a kernel jako konvoluční jádro (druhá funkce). Pro konvoluci funkcí spojitých slouží následující vzorec:

$$(f \star h)(t) = \int_{-\infty}^{\infty} f(\tau)h(t-\tau)d\tau$$

$\star$ – Konvoluce

f – Funkce f

h – Funkce h

h(t - τ) – Otočená funkce h

Pokud se na tento vzorec aplikuje Fourierova transformace, je možné získat rovnici, která tvoří jednu důležitou vlastnost pro aplikování výpočtů konvoluce.

$$\vartheta\{(f \star h)(t)\} = \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} f(\tau)h(t-\tau)\right]e^{-j2\pi\mu\tau}d\tau$$

$$\vartheta\{(f \star h)(t)\} = \int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} h(t-\tau)e^{-j2\pi\mu\tau}dt\right]d\tau$$

$$\vartheta\{(f \star h)(t)\} = \int_{-\infty}^{\infty} f(\tau)\left[H(\mu)e^{-j2\pi\mu\tau}\right]d\tau$$

$$\vartheta\{(f \star h)(t)\} = H(\mu)\int_{-\infty}^{\infty} f(\tau)e^{-j2\pi\mu\tau}d\tau$$

$$\vartheta\{(f \star h)(t)\} = H(\mu)F(\mu)$$

$$\vartheta\{(f \star h)(t)\} = (H * F)(\mu)$$

$$(f \star h)(t) = \vartheta^{-1}\{(H * F)(\mu)\}$$

F(μ) – Fourierova transformace funkce f

H(μ) – Fourierova transformace funkce h

ϑ – Symbol zastupující Fourierovu transformaci

Této rovnici se říká konvoluční teorém a popisuje vztah mezi konvolucí funkcí a element-wise násobením jejich podob po Fourierově transformaci. Toto umožní snížit časovou náročnost z $O(n^2)$ blíže k O(n*log(n)) při použití Fast Fourier Algoritmu. Tento rozdíl se nejvíce projevuje ve výpočtu s většími kernely o vyšších dimenzích, jako je zpracování obrazu.

3.4 Dekonvoluce

3.4.1 Úvod

[1]Téma dekonvoluce se zabývá získáním původní funkce, která byla poškozená konvolucí. Pokud tato funkce vznikla v uzavřeném prostředí a je známa přesná podoba kernelu, je možné provést inverzi. Neznáme-li kernel, nebo se k funkci přidal během procesu šum, je potřeba využít komplikovanější metodu. Pro řešení problému šumu je možné využít Wienerovu dekonvoluci nebo Lucy-Richardson dekonvoluci. Pokud ale není kernel znám, je potřeba jej buďto odhadnout z okolností kdy je funkce pořízena nebo využít metodu slepé dekonvoluce.

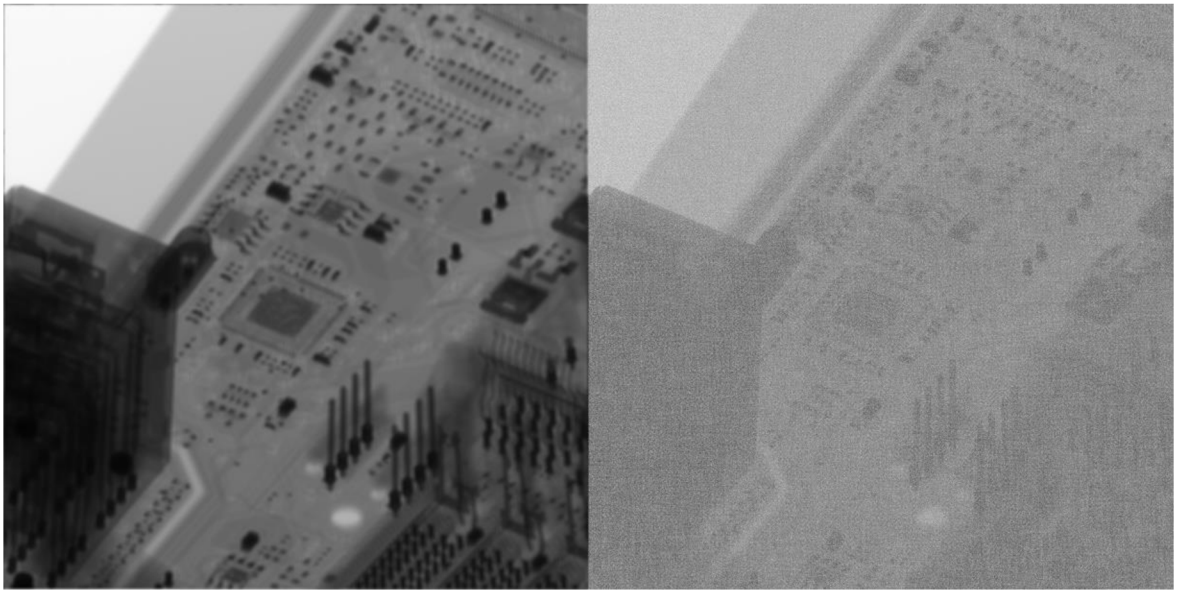
3.4.2 Inverze

Metoda Inverze vychází z jednoduchého pochopení konvolučního teorému. Pokud konvoluce odpovídá násobení ve Fourierově spektru, nejjednodušší řešení je zpětně vydělit. Dokud je znám přesný kernel a funkce je bez šumu, je tato metoda teoreticky přesná. Při konvoluci ve velkých měřítkách mohou nastávat problémy v implementaci z důvodu nepřesnosti plovoucí desetinné čárky na počítači.

$$F(u,v) = \frac{G(u,v)}{H(u,v)}$$

F(u, v) – Fourierova transformace původní funkce
G(u, v) – Fourierova transformace funkce po konvoluci
H(u, v) – Fourierova transformace funkce kernelu

Tato metoda přestává být použitelná při existenci šumu. Dělením se silně zesílí efekt šumu a funkce přestává být čitelná.



Obrázek 22 – Nalevo obraz o rozmezí hodnot 0-255 rozmazán box filtrem s aditivním šumem o rozmezí 0-0.01, napravo jeho inverze [zdroj: autor]

3.4.3 Wiener

Wienerova metoda dekonvoluce je vhodná pro případy, kde je potřeba řešit šum. V případě, kdy je známý šum, je možné použít SNR (signal to noise ratio), které je možno vypočítat podle následujícího vzorce:

$$SNR = \frac{P_{signal}}{P_{noise}}$$

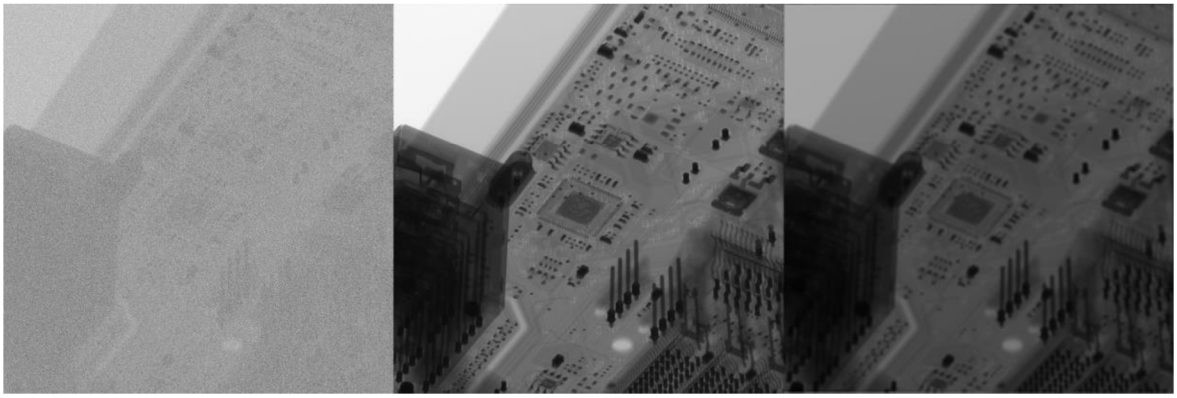
V případě, kdy nejsou informace o šumu k dispozici, je možné použít konstantu. Konstanta se liší podle velikosti a typu šumu. Jedna z možností, jak se k této konstantě dostat, je provedení dekonvoluce na dostatečném množství vzorků s postupnou inkrementací této konstanty.

Narozdíl od inverze se Wienerova dekonvoluce snaží srazit efekt šumu přenásobením výsledku inverze zlomkem, kde ve jmenovateli působí již zmíněná konstanta.

$$F(u,v) = \left[\frac{1}{H(u,v)} * \frac{|H(u,v)|^2}{|H(u,v)|^2 + K}\right] G(u,v)$$

K – Buďto konstanta nebo SNR

Při dosazení K=0 do tohoto vzorce vyjde vzorec totožný s Inverzí.



Obrázek 23 – nalevo nízká konstanta, uprostřed ideální konstanta, napravo vysoká konstanta [zdroj: autor]

Odhadování konstanty většinou vede ke třem typům výsledků. V případě, kdy je konstanta moc malá, je obraz silně ovlivněn šumem. Dále se šum čím dál více snižuje a obraz získává správný vzhled. A v poslední fázi při vysokých hodnotách se snižuje průměrná hodnota v obraze a obraz začíná tmavnout.

3.4.4 Lucy-Richardson

[11]Lucy-Richardson algoritmus nezávisle popsal William Richardson a Leon B. Lucy v letech 1972 (W.R.) a 1974 (L.B.L.). Na rozdíl od předchozích dvou metod je tento algoritmus náročnější na výpočet a jelikož je iterativní, tak je potřeba ho počítat víckrát za sebou. Iterace se postupně konvergují k jednomu výsledku.

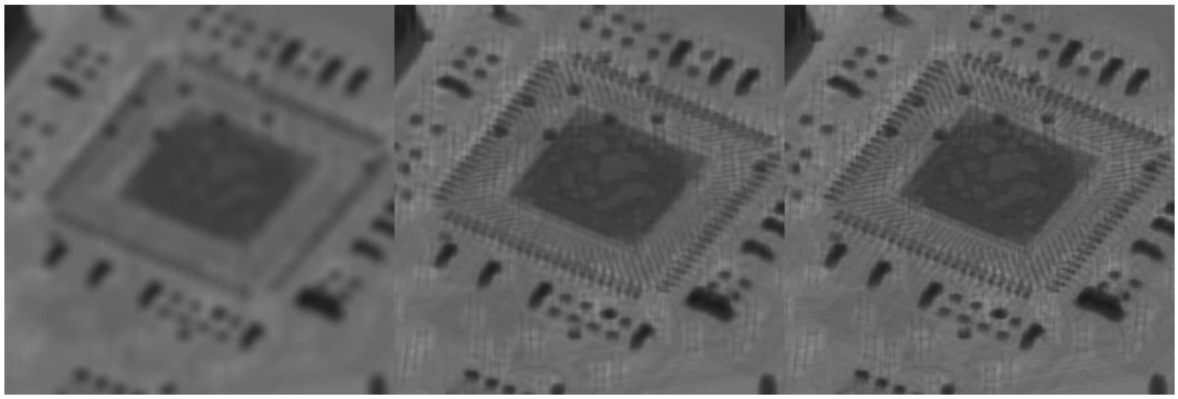
$$u^{(t+1)} = u^{(t)} * \left(\frac{d}{u^{(t)} \star P} \star P^* \right)$$

u – Iterace s identifikátorem na místě mocniny

d – Degradovaný obraz

P – Kernel

P* - Otočený kernel, v tomto kontextu má operace efekt vzájemné korelace



Obrázek 24 – 1. iterace nalevo, 200. iterace uprostřed, 500. iterace napravo [zdroj: autor]

Největší změny se často projevují v prvních 100-200 iterací a jelikož jsou tyto výpočty náročné, je vhodné se rozhodnout, jaký počet iterací je pro určité užití vhodný.

4 Praktická část

4.1 CUDA

4.1.1 Úvod

[12]CUDA je akronym pro Compute Inified Device Architecture. [13]Jedná se o API umožňující práci s grafickými kartami od značky NVIDIA. Využití výpočtů na grafické kartě umožňuje vysokou úroveň paralelizace, obzvlášť v případě element-wise operací, které jsou v této práci použity. [14]CUDA je dělána pro jazyky C, C++, Fortran, Python a MATLAB, v případě této práce je využita verze pro c++.

4.1.2 Použití

Pro použití CUDA API je potřeba nejdříve stáhnout a nainstalovat CUDA Toolkit, který je dostupný na https://developer.nvidia.com/cuda-downloads. Součástí Toolkitu je také subpackage cuFFT rozebraný později.

CUDA obsahuje celkově mnoho funkcí a možností, ale v této sekci se budou probírat jenom v rozsahu základů použitých v této práci.

K paralelizaci dochází spouštěním kernelů (nemají nic společného s kernely z tématiky zpracování obrazu) na jednotlivých vláken. Tato vlákna se dále rozdělují na bloky vláken a v případě potřeby také na clustery bloků. Velikost bloku a jejich počet se většinou rozhodují dvěma způsoby.

První způsob se používá v případě, kdy je situace jednodušší a při výpočtu nezáleží na dimenzích počítaných hodnot. Toto je případ této práce, kdy všechny operace je možné počítat element-wise. Počet vláken v bloku by měl být optimálně rozhodnut podle specifikace grafické karty, ale jedno z obecných pravidel je, že by tato hodnota měla být násobek 32. V této práci je použito 512 vláken na blok a počet bloků je dopočítán podle potřebné velikosti.

Druhým způsobem je využít typu dim3 (jedná se o vektor celých čísel) a nastavit velikosti dimenzí. Tyto dvě metody se liší hlavně možností využít identifikátory pro vlákna a bloky uvnitř kernelu. Hlavní výhodou užití více dimenzí je schopnost identifikovat nejen po ose x ale také y a z.

```
template<typename T>
__global__ void InverseNormalization(const uint64_t size, T *arr) {
    uint64_t i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < size) {
        arr[i] = arr[i]/T(size);
    }
}
```

Obrázek 25 - Kernel zpětné normalizace [zdroj: autor]

Typická struktura kernelu se skládá ze specifikátoru __global__ (v případě potřeby je možné použít ještě __device__ nebo __host__, do dostatečné hloubky, aby těchto specifikátorů bylo využito, ale tato práce nezachází), typické deklarace funkce a těla funkce s několika faktory typickými pro kernel.

Na začátku kernelu většinou bývá výpočet pro index vlákna, ten se později využívá k výpočtu (v tomto případě pro manipulaci se správným místem v paměti). Při využití bloků o více dimenzích je možné pracovat také s blockDim.y/z, blockIdx.y/z, threadIdx.y/z.

Další částí kódu bývá podmínka pro ošetření indexu vlákna. Jelikož se kernely pouští po určitých várkách, je možné, zasáhnout mimo alokovanou paměť. Proto se tento problém nejjednodušeji ošetřuje podmínkou.

V těle podmínky se nachází výpočetní kód. Pro řádnou optimalizaci by tento kód měl být co nejkratší, ideálně jedna operace. Příklad inverze je ještě obohacený o template s typem T. Toto je obyčejné použití templatů v jazyce C++ a slouží k neopakování kódu. Jediný zvláštní vliv, co má template na kód CUDA, je problematika psaní templatu mimo header v jazyce C++.

Pro spuštění kernelu je potřeba využít execution configuration syntax, značen <<<…>>>. Tento syntax má maximálně čtyři vstupní proměnné <<< numberOfBlocks, threadsPerBlock, sharedMemoryPerBlock, Stream >>>. V této práci není sdílená paměť použita, a tak je vždy 0.

Tento kód je potřeba psát v souborech pro kód CUDA (typicky .cuh pro header a .cu pro zdrojový kód).

4.2 CUFFT

[15]CuFFT je součástí CUDA Toolkitu a zabývá se výpočtem rychlé Fourierovy transformace. API cuFFT se skládá ze dvou knihoven, z nichž je projektem použita pouze knihovna cuFFT. Použitý algoritmus je optimalizován pro vstupy o velikostech zapsatelných jako $2^a \times 3^b \times 5^c \times 7^d$.

Pro použití knihovny k Fourierově transformaci je potřeba nejprve vytvořit plán. Tomuto plánu se nastaví, s kolika dimenzemi se počítá (max 3). Dále se nastaví typ operace (reálná do komplexní, komplexní do komplexní, komplexní do reálné). A nakonec se tento plán předá spolu s pointery do paměti funkci na vykonání Fourierovy transformace(cufftExec*typ*).

Pro 2D transformaci využitou v této práci je potřeba vědět, že při transformaci z reálných hodnot do komplexních se počet hodnot změní z $N_1 * N_2$ na $N_1 * [(N_2/2) + 1]$. Tato změna vychází z Nyquistova limitu, kde v praxi dochází k zrcadlení hodnot s opačným znaménkem u komplexní části. Zkrácené hodnoty jsou vráceny při zpětné

transformaci. Dále je ještě potřeba provést při zpětné transformaci normalizaci (viz obrázek 25.).

4.3 Implementace teorie

4.3.1 Kernel

Kernel má dvě formy implementace. V první formě je to možnost počítat konvoluci normálním způsobem v reálných hodnotách, kdy při přetečení přes okraj se do výpočtu vezme hodnota z druhé strany stejného řádku obrázku.

V druhé formě se kernel nejdříve transformuje do Fourierova spektra a výpočty se tak dělají v komplexní formě. Jelikož je ale většina operací element-wise, je potřeba zařídit, aby kernel měl stejnou velikost jako pole, se kterým pracuje. Právě k tomuto slouží padding.

0	1	2
3	4	5
6	7	8

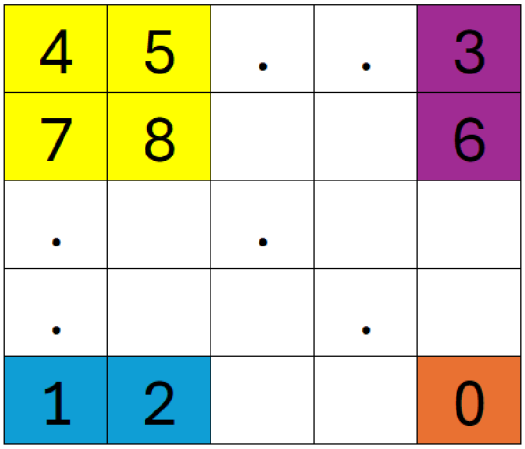
Obrázek 26 - Pozice 3x3 kernelu s vyznačeným středem [zdroj: autor]

Nejdříve je potřeba najít kořen kernelu

0	1	2
3	4	5
6	7	8

Obrázek 27 - 3x3 kernel rozdělen na oblasti podle kořene [zdroj: autor]

Od kořenu (v tomto případě pozice 4) se všechny pozice napravo a dolu označí jako stejná oblast. Všechny hodnoty nad touto (žlutou) oblastí se zařadí do jiné oblasti. Další oblast je nalevo od žluté (ne výš než kořen). Poslední oblast odpovídá tvarem žlutou ale podle diagonály zrcadlenou.



Obrázek 28 - Kernel s paddingem [zdroj: autor]

Posledním krokem je tyto oblasti přemístit, tak jak je znázorněno na obrázku 28. Všechny hodnoty na místě paddingu jsou nulové. Pokud se tímto procesem zvětší Kernel na stejnou velikost jako jiné pole, bude možné provádět element-wise operace ve Fourierově spektru.

4.3.2 Wiener

Wienerův filtr má v této práci 3 formy implementace. První forma slouží k výpočtu na procesoru a je implementována jako template pro použití s různými datovými typy. V průběhu tohoto algoritmu je 2D Fourierova transformace počítána jako dvě sady 1D transformací. Vzorec je počítán totožně, jako byl zapsán v teoretické části.

Další dvě formy jsou mezi sebou totožné až na datový typ, obě jsou počítány na grafické kartě, s důrazem na paralelizaci. Jelikož datový typ mění nastavení cuFFT plánu, je potřeba mít dvě různé varianty.

4.3.3 Lucy-Richardson

Stejně jako u Wienera existují verze pro výpočet na CPU a dvě verze pro GPU. V průběhu výpočtu existuje clamp funkce, která zabraňuje uskočení hodnot mimo očekávaný rozsah

Jedním z největších problémů pro výpočetní rychlost jsou operace s pamětí, je možné část operací paralelizovat asynchronním spuštěním, ale stále je to veliký faktor v rychlosti.

4.3.4 Problematika implementace

Při přechodu z teoretické představy na implementaci nastává několik problémů. Jedním z hlavních, které se mohou prokázat je problém s desetinnou čárou. Reprezentace desetinných čísel na počítači není přesná a s velkým počtem operací se odchylka od správné hodnoty může lišit. Na druhou stranu je tento jev poznat až při výpočtech s vysokým rozlišením pro kernel i obraz. Navíc jde o porovnání mezi získaným výsledkem a originálem, který nemusí být k dispozici.

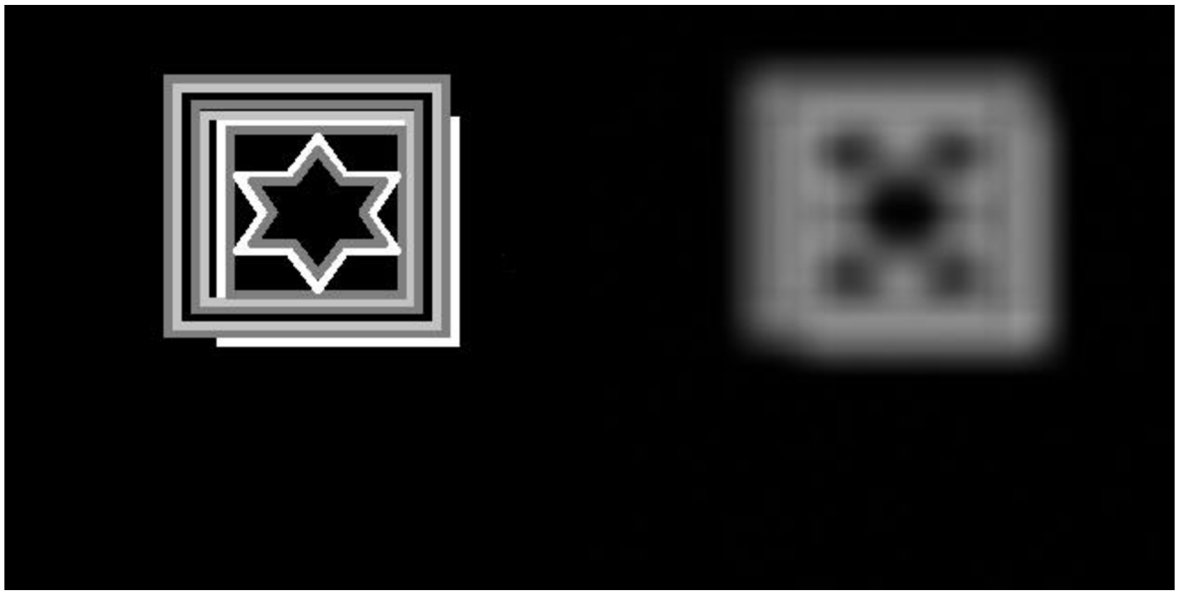
Jedním z řešení tohoto problému je použít datový typ s větší přesností jako je double, tím se ale může zvýšit doba pro výpočty. Další metodou, která pro profesionálnější nasazení je skoro automatická, je využít optimalizovaný algoritmus jako cuFFT tam kde je ho možno použít.

Z vlastních testů se rozdíl u float i double hodnot inkrementálně zvyšuje s počtem operací, ale při použití cuFFT se tento problém začne výrazně prokazovat o mnoho operací později.

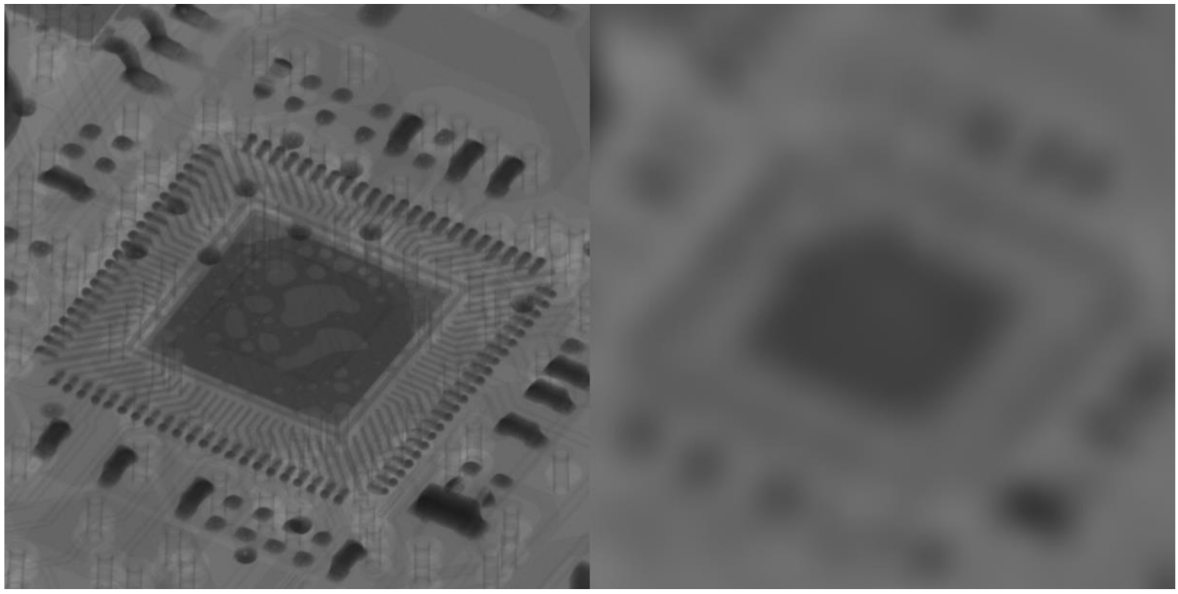
Dalším problémem, který je potřeba řešit, je dělení nulou. Z charakteristiky desetinných hodnot na počítači může být problém ošetřování pro případ přímo nuly. Z testování bylo dosaženo hodnoty 0.000024, která slouží jako hranice pro platné hodnoty. Tato hodnota se z tohoto důvodu používá skrze kód pro ošetření dělení nulou.

4.4 Porovnání výsledků

4.4.1 Původní obraz a jeho konvoluce



Obrázek 29 - Původní obraz s velikostí 320x320 a jeho konvoluce s gaussovským kernelem o velikosti 33x33 [zdroj: autor]



Obrázek 30 – Výběr o velikosti 1024x1024 z obrazu o velikosti 4096x4096 a jeho konvoluce s Gaussovským kernelem o velikosti 101x101 [zdroj: autor]

4.4.2 Lucy-Richardson

4.4.2.1 Časové porovnání

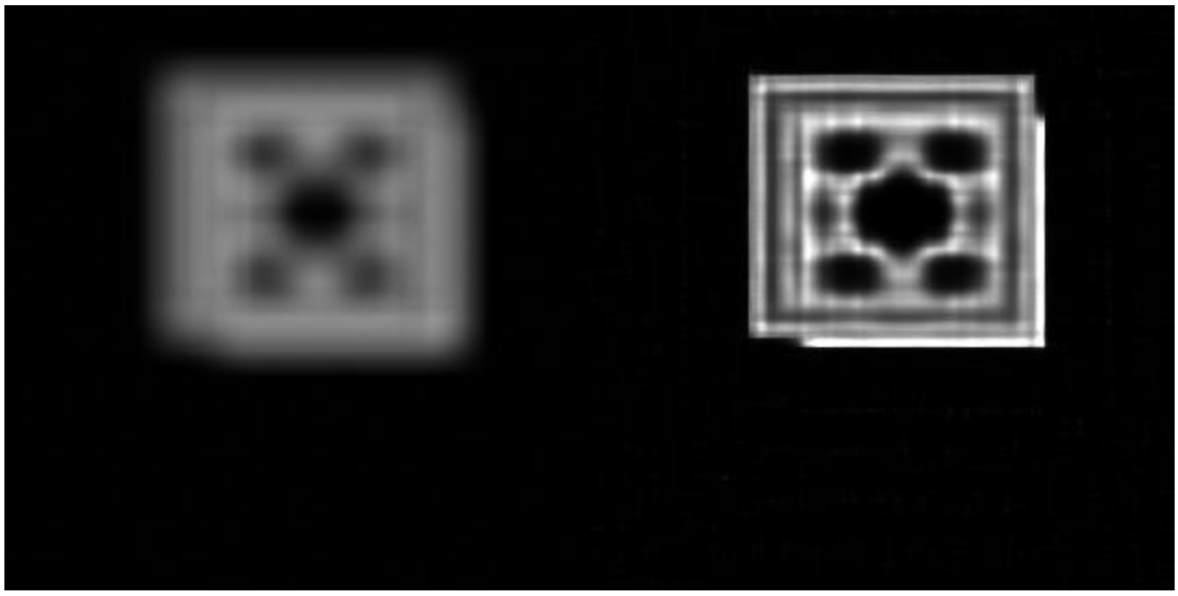
Tabulka 1 – Testování Lucy-Richardson algoritmu [zdroj: autor]

Rozlišení obrazu	Datový typ	Typ Výpočtu	Počet iterací	~Příprava	~Výpočet	~Celkem
320x320	Float	GPU	300	3 [ms]	60 [ms]	64 [ms]
320x320	Double	GPU	300	4 [ms]	190 [ms]	195 [ms]
4096x4096	Float	GPU	300	84 [ms]	2.85[s]	3[s]
4096x4096	Double	GPU	300	270 [ms]	19[s]	20[s]
320x320	Float	CPU	10	-	-	52[s]
320x320	Double	CPU	10	-	-	56[s]

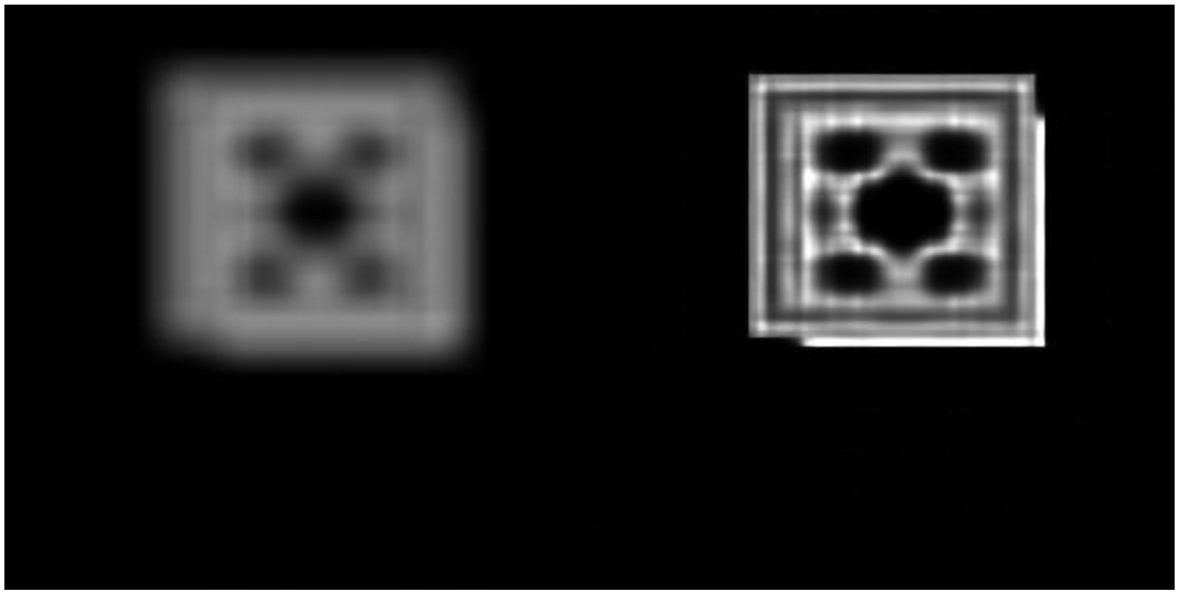
Tato tabulka ukazuje výsledky časové náročnosti podle optimalizace, rozlišení a typu plovoucí čárky. Měření CPU implementace pro rozlišení 4096x4096 chybí z důvodu časové náročnosti. CPU implementaci chybí příprava a výpočet, protože není potřeba nic připravovat, a tak je výpočet totožný s celkovým časem. Na původní obrázky byl aplikován Gauss blur a šum s normální rozdělením na intervalu 0-1. Velikost kernelu je pro 320x320 obrázek 33x33 a pro 4096x4096 je to 101x101. Velikost kernelu má časově minimální vliv, jelikož je tak či onak potřeba počítat padding.

4.4.2.2 Vizuální porovnání

Na levé straně leží původní obrázek po konvoluci, jako je zobrazený na začátku sekce o porovnání výsledků. Na pravé straně leží výsledek po dekonvoluci.



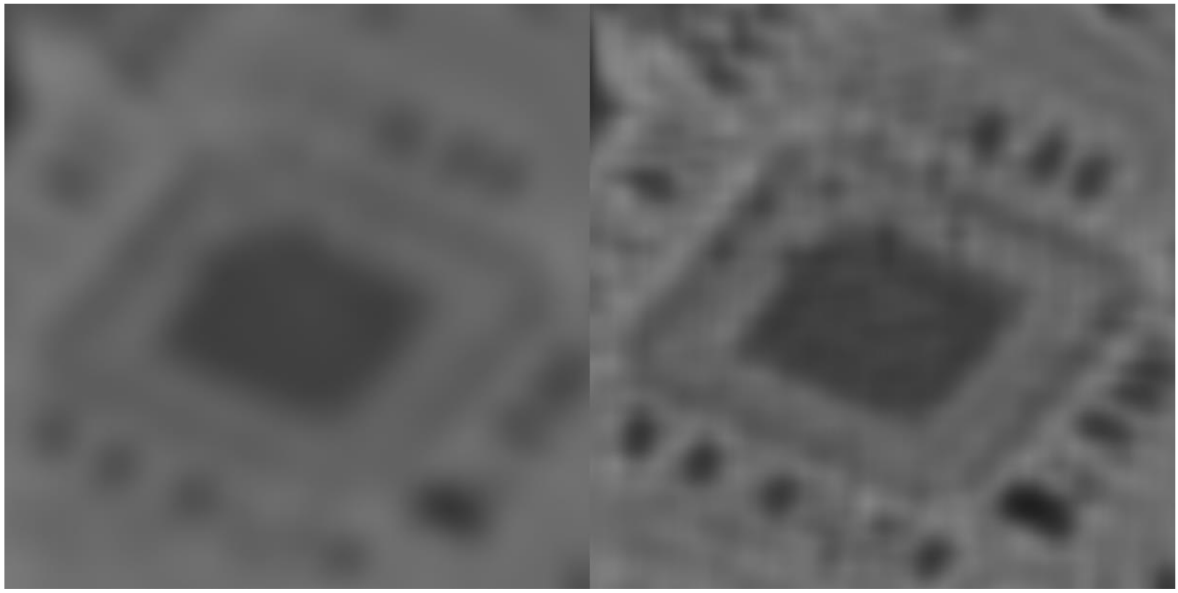
Obrázek 31 – Porovnání Lucy-Richardson dekonvoluce 320x320 obrazu na grafické kartě s typem Float [zdroj: autor]



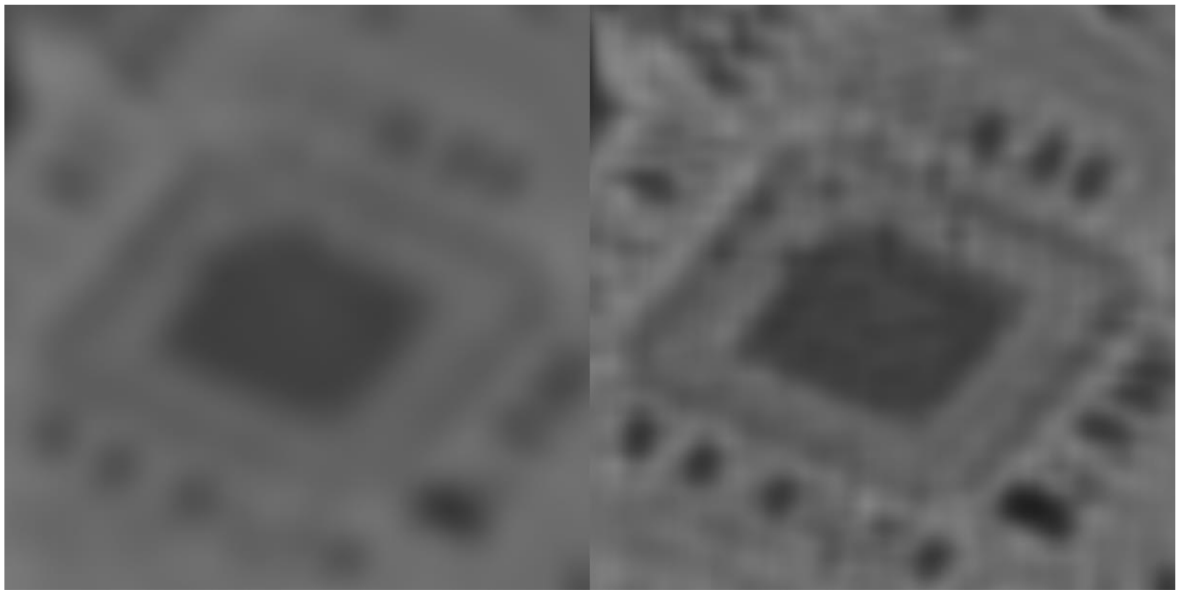
Obrázek 32 - Porovnání Lucy-Richardson dekonvoluce 320x320 obrazu na grafické kartě s typem Double [zdroj: autor]

Obrázek 33 - Porovnání Lucy-Richardson dekonvoluce 320x320 obrazu na procesoru s typem Float [zdroj: autor]

Obrázek 34 - Porovnání Lucy-Richardson dekonvoluce 320x320 obrazu na procesoru s typem Double [zdroj: autor]



Obrázek 35 - Porovnání Lucy-Richardson dekonvoluce 4096x4096 obrazu na grafické kartě s typem Float [zdroj: autor]



Obrázek 36 - Porovnání Lucy-Richardson dekonvoluce 4096x4096 obrazu na grafické kartě s typem Double [zdroj: autor]

4.4.3 Wiener

4.4.3.1 Časové porovnání

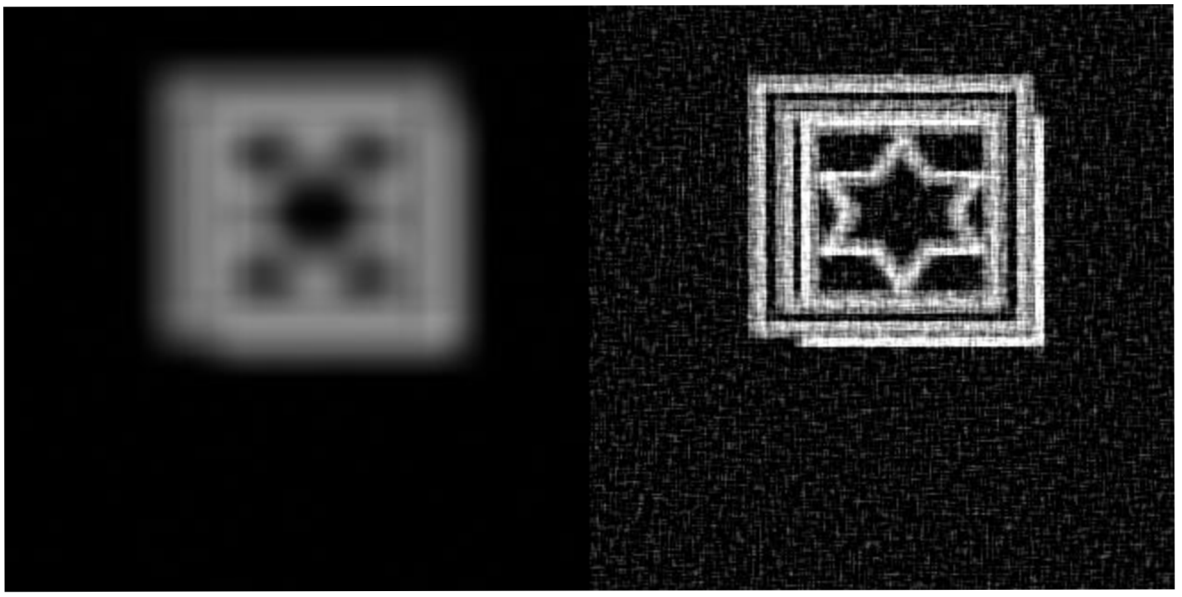
Tabulka 2 - Testování Wienerova algoritmu [zdroj: autor]

Rozlišení obrazu	Datový typ	Typ Výpočtu	~Příprava	~Výpočet	~Celkem
320x320	Float	GPU	4000 [μs]	700 [μs]	5000 [μs]
320x320	Double	GPU	7000 [μs]	1800 [μs]	9000 [μs]
320x320	Float	CPU	-	-	900 [ms]
320x320	Double	CPU	-	-	950 [ms]
4096x4096	Float	GPU	60 [ms]	3.5 [ms]	64 [ms]
4096x4096	Double	GPU	114 [ms]	25 [ms]	140 [ms]

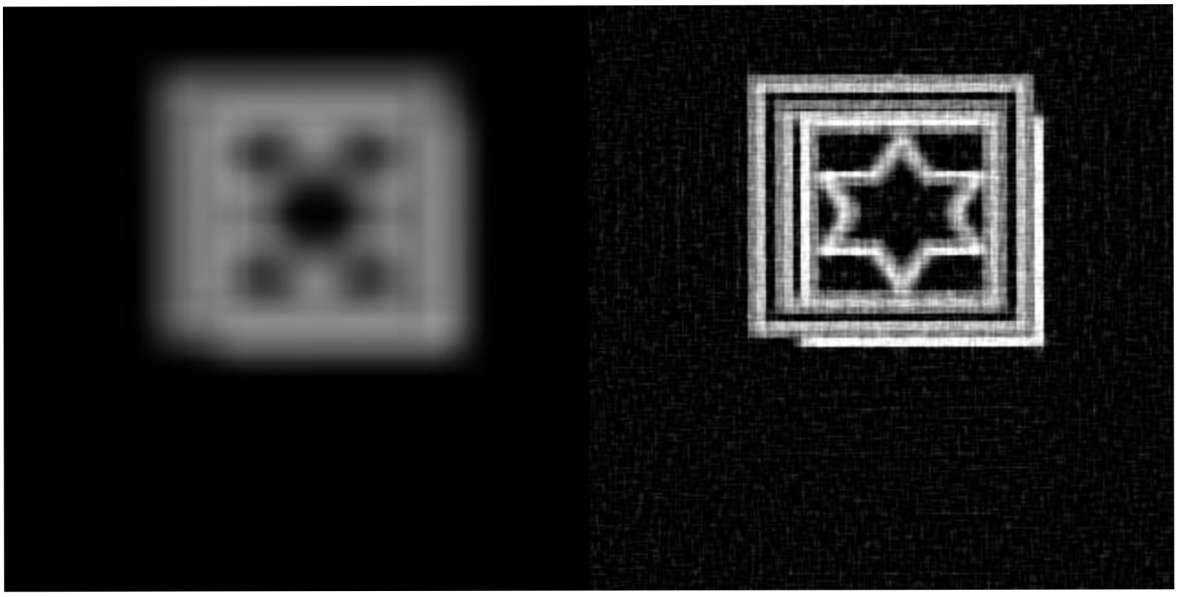
Platí stejná pravidla jako v případě Lucy-Richardson porovnání.

4.4.3.2 Vizuální porovnání

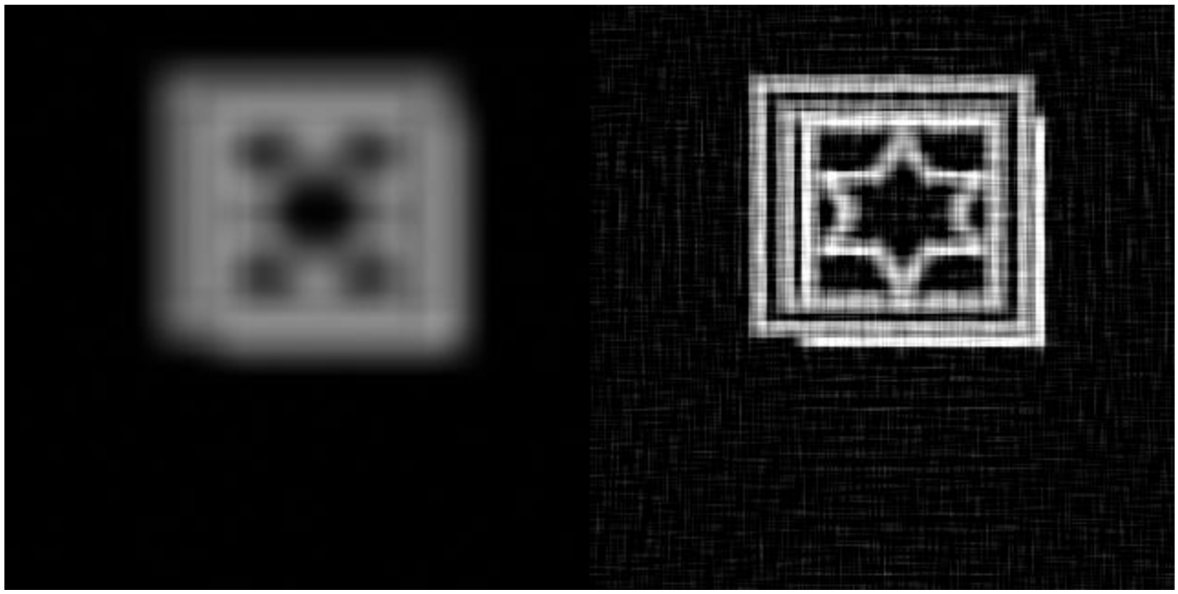
Platí stejné podmínky jako ve vizuálním porovnání Lucy-Richardson.



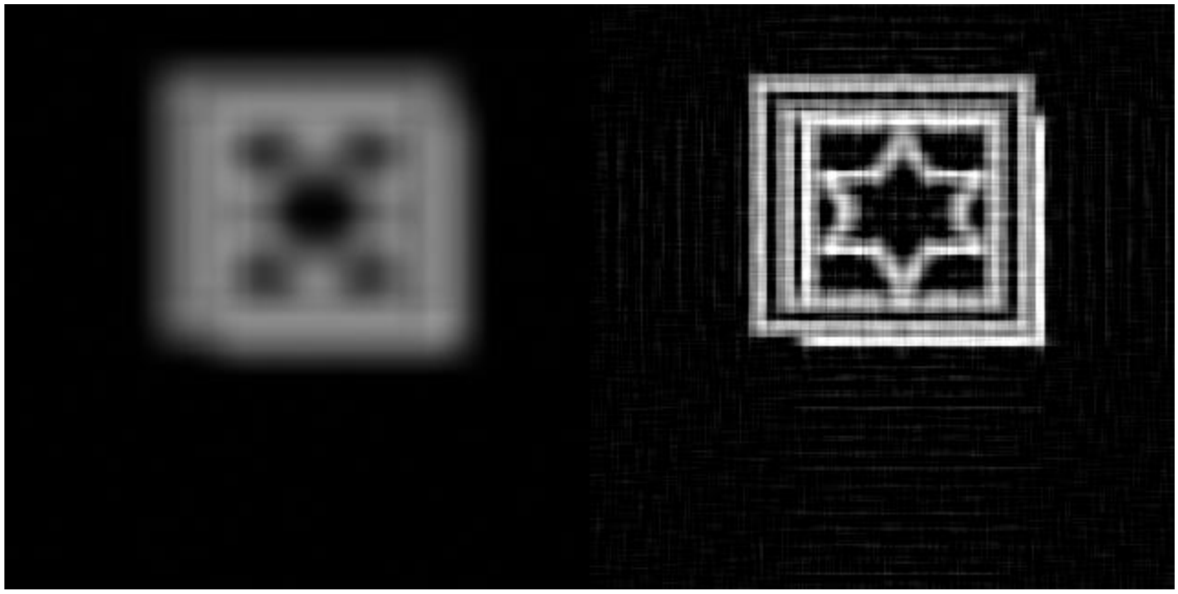
Obrázek 37 - Porovnání Wienerovy dekonvoluce 320x320 obrazu na grafické kartě s typem Float [zdroj: autor]



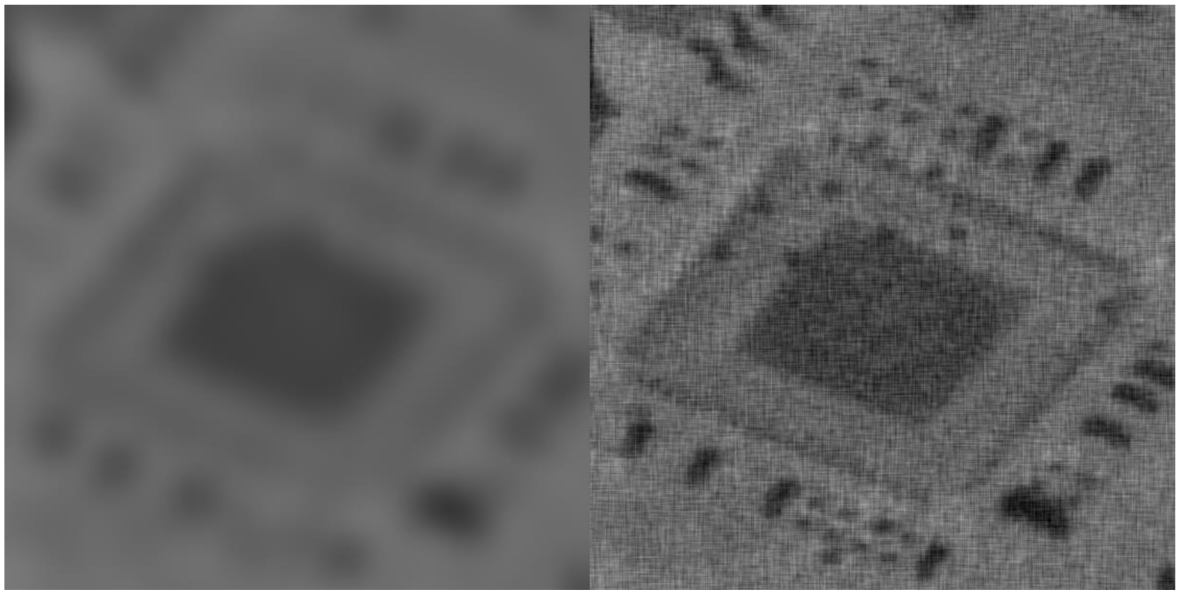
Obrázek 38 - Porovnání Wienerovy dekonvoluce 320x320 obrazu na grafické kartě s typem Double [zdroj: autor]



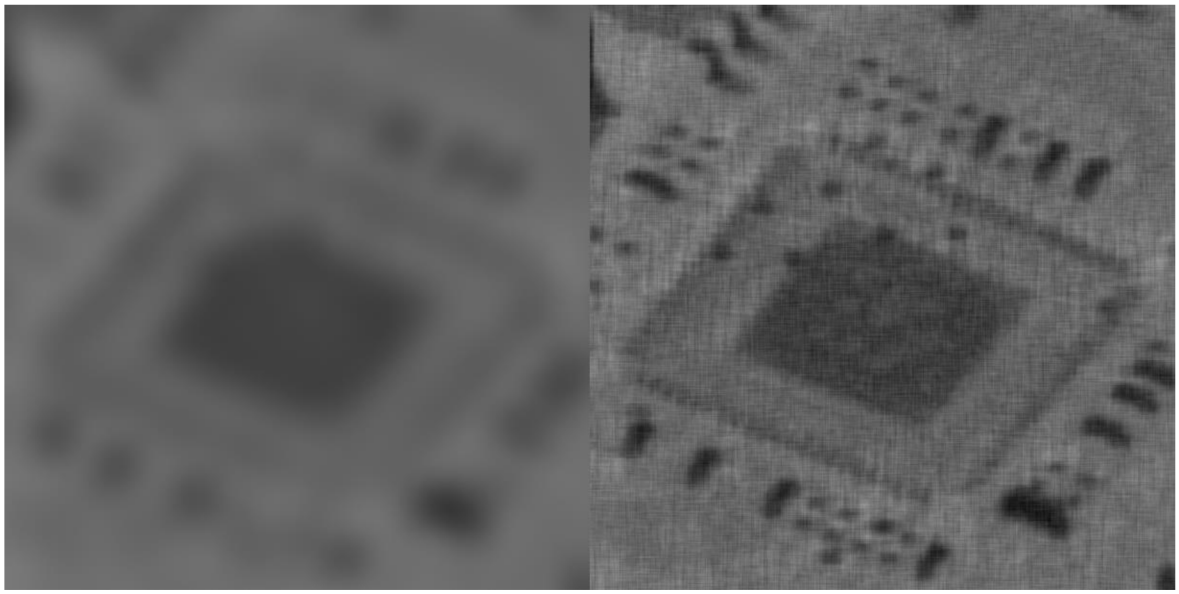
Obrázek 39 - Porovnání Wienerovy dekonvoluce 320x320 obrazu na procesoru s typem Float [zdroj: autor]



Obrázek 40 - Porovnání Wienerovy dekonvoluce 320x320 obrazu na procesoru s typem Double [zdroj: autor]



Obrázek 41 - Porovnání Wienerovy dekonvoluce 4096x4096 obrazu na grafické kartě s typem Float [zdroj: autor]



Obrázek 42 - Porovnání Wienerovy dekonvoluce 4096x4096 obrazu na grafické kartě s typem Double [zdroj: autor]

4.5 Struktura programu

```
│   CMakeLists.txt
│   convolved_with_noise.bmp
│   main.cpp
│
├───BMP
│       BMP.cpp
│       BMP.h
│
├───include
│   ├───Deconvolution
│   │   ├───LucyRich
│   │   │       LucyRichCPU.h
│   │   │       LucyRichDouble.cuh
│   │   │       LucyRichFloat.cuh
│   │   │
│   │   └───Wiener
│   │           WienerCPU.h
│   │           WienerDouble.cuh
│   │           WienerFloat.cuh
│   │           WienerUtils.cuh
│   │
│   └───Utils
│       │   ConvolutionUtils.h
│       │
│       ├───CudaUtils
│       │       CudaPointerHandler.h
│       │       CudaStreamHandler.h
│       │       CudaUtils.cuh
│       │       CufftPlanHandler.h
│       │
│       ├───Fourier
```

```
│   │   Separate2DDFT.h
│   │
│   └──HelperClasses
│       ImageWrapper.h
│       KernelWrapper.h
│       point.h
│       Timer.h
│
└──src
   ├──Deconvolution
   │  ├──LucyRich
   │  │   LucyRichDouble.cu
   │  │   LucyRichFloat.cu
   │  │
   │  └──Wiener
   │      WienerDouble.cu
   │      WienerFloat.cu
   │
   └──Utils
      ├──CudaUtils
      │   CudaStreamHandler.cpp
      │   CufftPlanHandler.cpp
      │
      └──HelperClasses
          Timer.cpp
```

Struktura je od začátku rozdělena na include a src. Pod adresářem include se nachází struktura adresářů s headry a pod src je tato struktura zrcadlena s jejich implementacemi v .cpp souborech. Ne každý header má k sobě do páru zdrojový .cpp, protože template jde za normálních podmínek implementovat pouze v headeru.

Další velké rozdělení je pod adresářem Utils, kde se nachází funkce a třídy potřebné pro výpočet nebo práci v prostředí CUDA.

5 Shrnutí a diskuse výsledků

Práce dosáhla z teoretického pohledu výsledků v souladu s literaturou. Z praktického hlediska ale ukázala potřebu optimalizace takto výpočtově obtížných algoritmů. Práce této se k této optimalizaci dostala hlavně pomocí knihoven cuFFT, které obsahují optimalizovaný FFT algoritmus pro výpočet na grafické kartě.

Psaní kódu v jazyce CUDA má ale pár problémů. Pro správnou funkčnost programu je potřeba ručně spravovat paměť a odchytávat chyby. Práce tohoto dosáhla pomocí utilitních tříd jako jsou CudaPointerHandler, CudaStreamHandler a CufftPlanHandler. Dalším problémem s prací v jazyce CUDA je debugování. Nativně IDE nepodporují debugování paměti grafické karty, a tak je potřeba buď průběžně kopírovat zpět do paměti procesoru nebo použití programů jako je NVIDIA Nsight Debugger.

Práce dosáhla relativně dobrého porovnání dvou dekonvolučních algoritmů. Ukázala, že Lucy-Richardson algoritmus dokáže zlepšit ostrost obrazu a že se iterativně přibližuje k jednomu výsledku. Potřebuje k tomu ale oproti Wienerově mnoho více výpočtů. Výhoda Wienerovy dekonvoluce je, že stačí provést pouze jeden výpočet. Wiener umí velice dobře najít hrany a detaily. Často ale přinese veliké množství šumu do obrazu a je potřeba najít správnou konstantu, se kterou lze najít co nejvíce detailů s co nejméně šumem.

6 Závěry a doporučení

Práce splňuje zadání, téma rekonstrukce obrazu je ale o dost komplikovanější a tyto dva algoritmy jsou spíše vstupní do této tématiky. Práce by se dále mohla pokračovat do několika směrů. Je možné upravit tyto základní algoritmy dalšími filtry, které mohou dosáhnout lepších výsledků. Další možností je řešení problému více z reality, kde kernel nemusí být skrze celý obraz stejný. Je v tomto případě potřeba řešit různé regiony zvlášť. S touto možností se také může spojovat odhadování kernelu podle fyzických charakteristik pořízení. Nakonec by další možností mohlo být také prozkoumávání dalších algoritmů, už pro v práci zmíněných typů nebo slepé dekonvoluce.

7 Seznam použité literatury

1. Image Restoration and Reconstruction. (2018). In R. C. Gonzalez & R. E. Woods, *Digital image processing* (pp. 365–449). Pearson.

2. Makarkin, M., & Bratashov, D. (2021). State-of-the-Art Approaches for Image Deconvolution Problems, including Modern Deep Learning Architectures. *Micromachines*, *12*(12), 1558. https://doi.org/10.3390/mi12121558

3. Preliminary Concepts. (2018). In R. C. Gonzalez & R. E. Woods, *Digital image processing* (pp. 253–261). Pearson.

4. Gonzalez, R. C., & Woods, R. E. (2018). *Digital image processing*. Pearson.

5. A brief history of Fourier series and transform. (2018). In R. C. Gonzalez & R. E. Woods, *Digital image processing* (p. 250). Pearson.

6. Background. (2018). In R. C. Gonzalez & R. E. Woods, *Digital image processing* (pp. 250–252). Pearson.

7. Sampling and the Fourier transform of sampled functions. (2018). In R. C. Gonzalez & R. E. Woods, *Digital image processing* (pp. 261–271). Pearson.

8. The discrete Fourier transform of one variable. (2018). In R. C. Gonzalez & R. E. Woods, *Digital image processing* (pp. 271–275). Pearson.

9. Periodicity. (2018). In R. C. Gonzalez & R. E. Woods, *Digital image processing* (p. 288). Pearson.

10. Convolution. (2018). In R. C. Gonzalez & R. E. Woods, *Digital image processing* (pp. 259–261). Pearson.

11. Lucy, L. B. (1974). An iterative technique for the rectification of observed distributions. *The Astronomical Journal*, *79*, 745. https://doi.org/10.1086/111605

12. *CUDA Toolkit—Free Tools and Training*. (n.d.). NVIDIA Developer. Retrieved 20 April 2024, from https://developer.nvidia.com/cuda-toolkit

13. *CUDA GPUs—Compute Capability*. (n.d.). NVIDIA Developer. Retrieved 20 April 2024, from https://developer.nvidia.com/cuda-gpus

14. *CUDA Zone—Library of Resources*. (n.d.). NVIDIA Developer. Retrieved 20 April 2024, from https://developer.nvidia.com/cuda-zone

15. *1. Introduction—cuFFT 12.4 documentation*. (n.d.). Retrieved 21 April 2024, from https://docs.nvidia.com/cuda/cufft/index.html

8 Seznam obrázků

9 Seznam tabulek

10 Přílohy

1) Zdrojové kódy na CD

11 Zadání práce z IS (eVŠKP)

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Ladislav Zítka**
Studium:	I2100307
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Dekonvoluce jednokanálových snímků pro systém s grafickou kartou podporující CUDA.**
Název bakalářské práce AJ:	Deconvolution of single-channel images for a system with a CUDA-enabled graphics card.

Cíl, metody, literatura, předpoklady:

Cíl práce: Cílem práce je vytvoření knihovny na dekonvoluci jednokanálových snímků pro systém s grafickou kartou podporující technologii CUDA.

Teoretická část: Základy teorie dekonvoluce a popis algoritmů.

Praktická část: Dokumentace vlastního programového řešení (vytvořené knihovny . Ukázka výsledků a porovnání mezi algoritmy (časová náročnost mezi samostatnými algoritmy a implementacemi na GPU a CPU + porovnání kvality výsledků).

GONZALEZ, Rafael C. a Richard E. WOODS. *Digital image processing.* 4th ed. New York, NY: Pearson, [2018].

Další bude upřesněna

Zadávající pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	prof. RNDr. PhDr. Antonín Slabý, CSc.

Datum zadání závěrečné práce: 30.5.2023