Dokumentace – Image processor

Ondřej Kříž
28. 05. 2022

Obsah

Anotace

Cílem softwaru je poskytnout uživateli jednoduchý obrázkový editor, nabízející základní editační nástroje a obrazové filtry. Jedná se například o převod do odstínů šedi, negativ barev nebo některé efekty založené na principu *konvolunčního jádra*, jako například *Gaussovské rozmazání*, či efekt Sobel. Dále nabízí základní transformace jako překlopení či převod do textové podoby jako ASCII art. Aplikace také u řady filtrů nabízí nastavení některých parametrů a výsledná data umožňuje uložit v podobě formátu *.jpg*.

1 Práce v softwaru

Při spuštění aplikace se nejprve vypíší výtisky signalizující načítání obrázků z předdefinovaného adresáře v **config.yaml**. Je rovněž možné, že pokud některý z obrázků je v podporovaném formátu(dle přípony souboru), ale není validní, tak na standard error vypíše, knihovní chybu a následovně cestu k souboru, který chybu vyvolal.[1]

Poté co jsou všechna data načtená aplikace vypíše uživatelskou nabídku.

```
Initializing data...
-> Not a JPEG file: starts with 0x89 0x50 !!!
/mnt/c/Users/kriz/CLionProjects/ImageProcessor/Resources/g_editor.jpg
Done!
-- Main menu --
a) Go to image library
b) Go to filters menu
c) End
```

Obrázek 1: Příklad stavu aplikace po spuštění

1.1 Image library

První nabídka(na obrázku 1 volba a)) uživateli umožňuje spravovat databázi(knihovnu) načtených obrázku, se kterými chce pracovat. Po této volbě je uživatel přesměrován do následujícího menu.

```
a
-- Library manager --
a) List all images
b) Add image to library
c) Add all images from dir
d) Go to main menu
```

Obrázek 2: Příklad stavu aplikace po přechodu do menu knihovny

Položky z obrázku výše mají následující funkcionality:

a) Vypíše cesty a indexy ke všem načteným souborům. Pokud některý soubor byl načtený a před výpisem přestane existovat, knihovna zaručí automatickou aktualizaci a daný soubor nevypíše.(stejně tak i při přidání, kde daný soubor naopak, za předpokladu, že byl přidán úspěšně, ve výpisu přibude)

b) Přidání jednoho obrázku do knihovny. Po vybrání je uživatel veden k vložení celé cesty k danému obrázku.

c) Přidání všech obrázků v daném adresáři. Pracuje pouze s danou složkou.(tedy není rekurzivní)

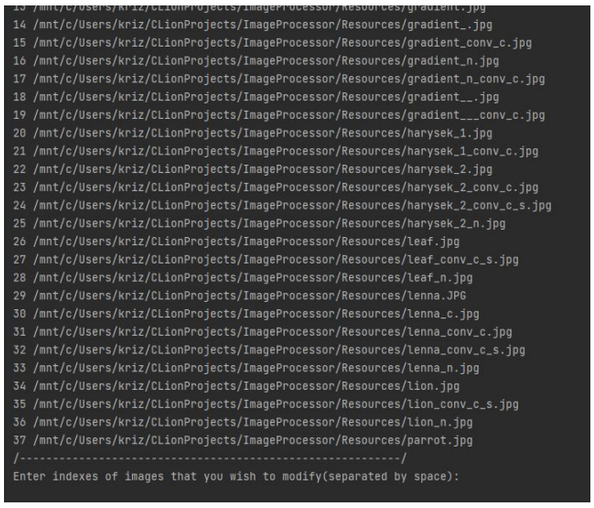
[1] Bohužel, i přesto, že dané soubory jdou např. otevřít v jiných editorech, použitá knihovna libjpeg je bohužel nedokáže rozpoznat.

d) Přesměruje uživatele zpět do hlavní nabídky z 1.

1.2 Aplikace filtrů

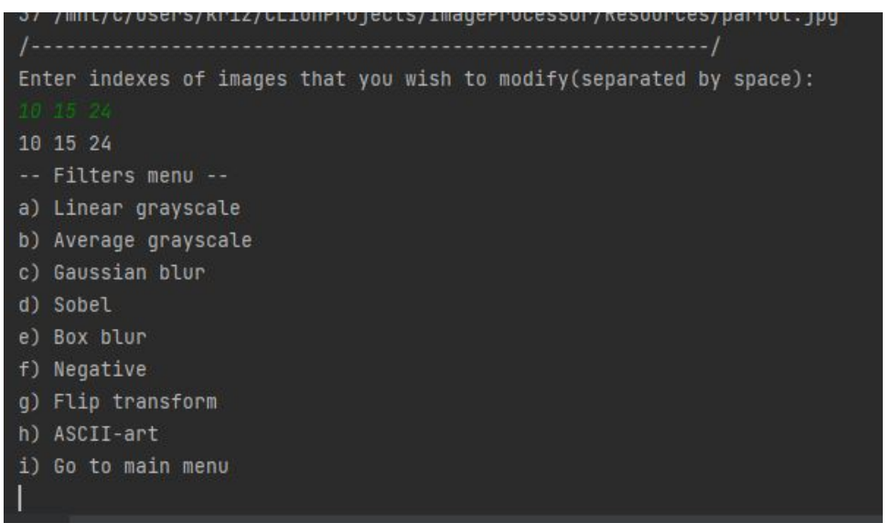
Při vložení volby b) z obrázku 1 je uživatel přesměrován do nabídky aplikace filtrů. Jako první se uživateli zobrazí seznam všech dostupných obrázků, které je možné editovat. Přitom je uživatel vyzván, aby vložil indexy(neboli čísla u jednotlivých adres obrázků) na které chce uživatel aplikovat některý z filtrů.

- Pokud uživatel zadá index, který není v seznamu uvedený, nebo je jakkoliv neplatný, daný index bude ignorován.
- Pokud uživatel zadá na vstupu jiné znaky než číslice 0-9 a prázdné znaky bude vypsán **Error** a uživatel bude přesměrován do hlavní nabídky z obrázku 1.
- Jediný další symbol, který je na vstupu v tomto stavu povolený je ”-1”(bez uvozovek). Tato volba automaticky vybere všechny obrázky z nabídky. (Je nutné aby vstup obsahoval pouze -1, případně prázdné znaky, jinak bude vstup brán jako neplatný)
- Prázdný vstup je brán jako chybný.



Obrázek 3: Příklad stavu aplikace po přechodu do menu filtrů

Poté co uživatel zadá validní vstup, je následně přesměrován do nabídky výběru filtru, který chce na zvolené obrázky aplikovat.



Obrázek 4: Příklad stavu aplikace výběru filtru

Následující sekce ukazují příklady výstupů jednotlivých filtrů na testovacím obrázku níže.

Obrázek 5: Testovací obrázek – originál

Každý filtr také přidává k původnímu názvu speciální apendix, který je uvedený pod každým obrázkem.

1.2.1 Linear grayscale

Obrázek 6: Testovací obrázek – linear grayscale

Apendix: *_grayscale_l*

1.2.2 Average grayscale

Obrázek 7: Testovací obrázek – average grayscale

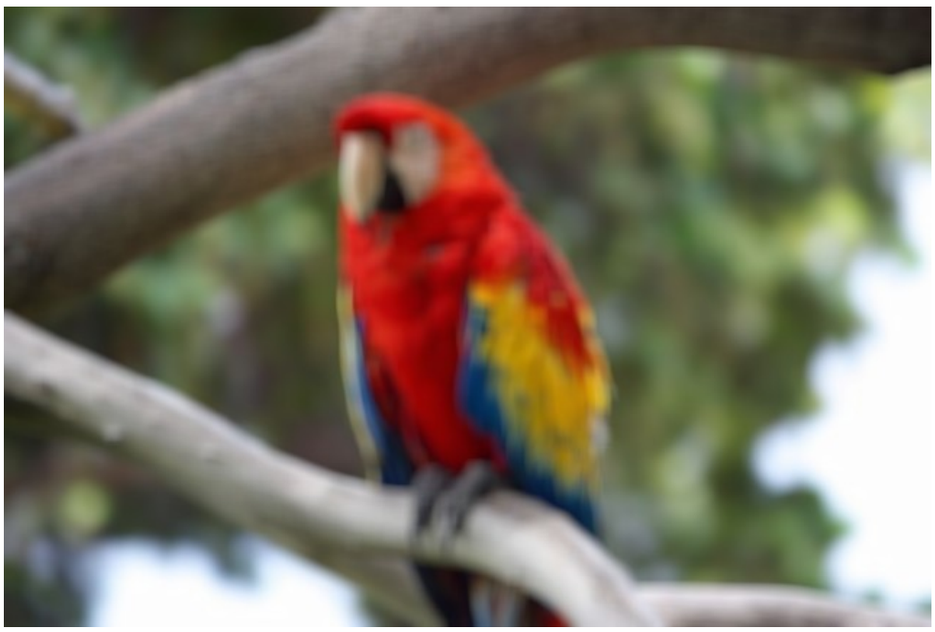
Apendix: *_grayscale_avg*

1.2.3 Gaussian blur

Uživatel může navolit následující parametry:

1. Velikost jádra konolunční matice.
2. Směrodatná odchylka Gaussovské funkce.

Veškeré výše uvedené vstupy mají své defaultní hodnoty. Pokud uživatel zadá neplatný vstup, je informován a je použita defaultní hodnota.
Na obrázku níže je zvolena velikost konvolunčního jádra 11 a směrodatná odchylka 5.



Obrázek 8: Testovací obrázek – Gaussian blur

Apendix: *_gauss_blur*

1.2.4 Box blur

Neboť bývá pouze jeden průchod boxblur velmi málo efektní(spíše slouží pro vyhlazení hrubých nedokonalostí), uživatel může zadat počet opakování, kolikrát se má výpočet na daný obrázek aplikovat. Při neplatném vstupu se provede právě 1 průchod.
Na obrázku níže je vidět box blur po 5 průchodech.

Obrázek 9: Testovací obrázek – Box blur

Apendix: *_box_blur*

1.2.5 Sobel

Sobel nabízí celkem čtyři různé směry – varianty jak může být na daný obrázek aplikován.

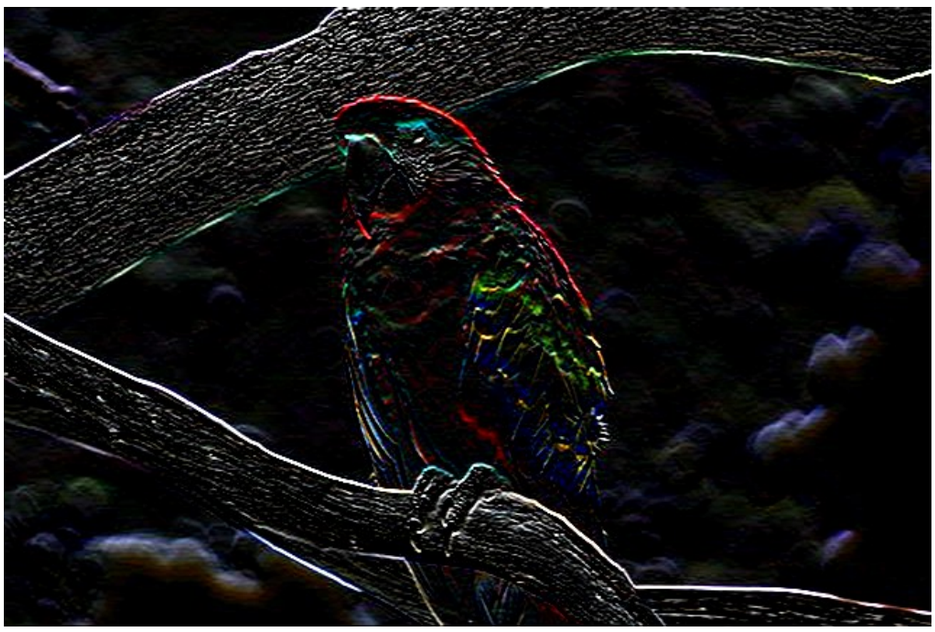
t – Ze shora (top)

b – Ze spodu (bottom)

l – Ze spodu (left)

r – Ze spodu (right)

Dále uživatel může zvolit, zdali si přeje provést Sobel v černobílé(v takovém případě je obrázek nejprve převeden automaticky do grayscale a následně je aplikován Sobel) podobě, nebo s původním RGB.



Obrázek 10: Testovací obrázek – Sobel colored bottom (název: parrot_sobel_colored_b)

Obrázek 11: Testovací obrázek – Sobel bottom (název: parrot_sobel_l)

Apendix: _*sobel*[pokud uživatel zvolil barevný Sobel tak se zde přidá ještě slovo '_color_'; jinak '_'][znak směru který uživatel zvolil]

1.2.6 Negativ

Obrázek 12: Testovací obrázek – Negativ

Apendix: *_negative*

1.2.7 Flip

Umožňuje překlopení obrázku podle svislé a horizontální osy. Uživatel při aplikaci filtru volí, zda použít horizontální – h nebo vertikální – v transformaci. Jako jediný filtr nemá nastavenou defaultní hodnotu. Pokud tedy uživatel zadá neplatnou volbu (tedy jiný vstup než h/v) operace se zruší a uživatel je přesměrován do startovního menu.

Obrázek 13: Testovací obrázek – Flipped Vertically parrot$_{flipped_v}$

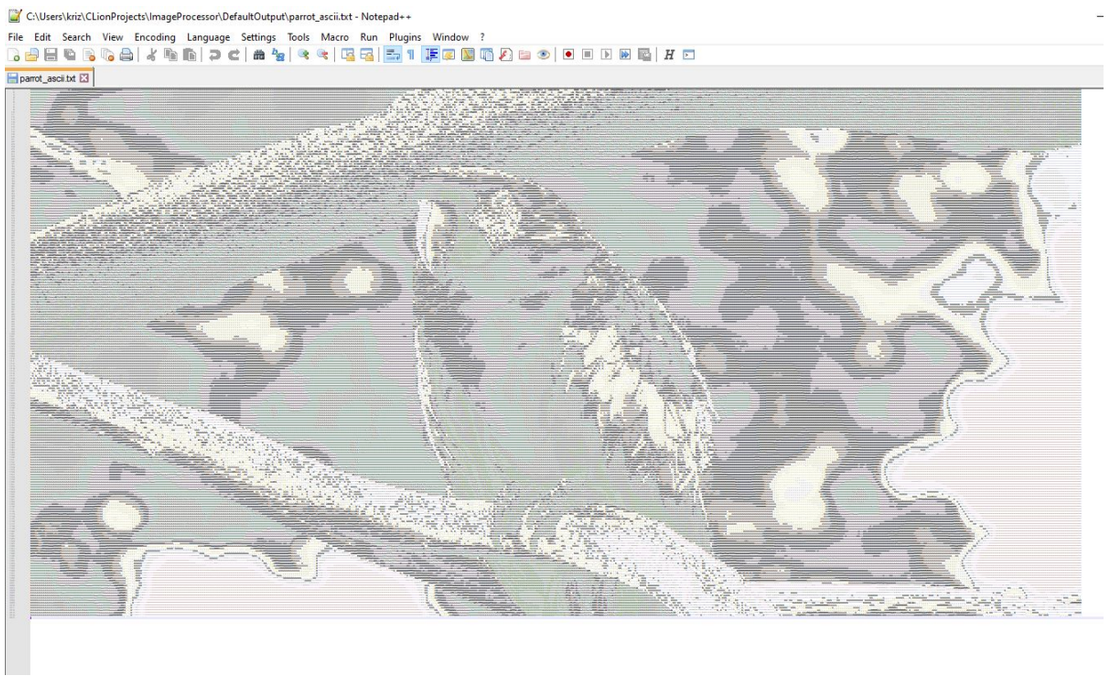
Appendix: *_flipped_*[směr překlopení]

1.2.8 Ascii art

Po zvolení převodu na ASCII art uživatel může zadat postupně následující parametry:

1. Počet znaků na řádku. (Počet znaků nemusí ve výsledku odpovídat viz. algoritmus ASCII).
2. Scaling factor (desetinné číslo)
3. Výstupní adresář.

Pokud uživatel libovolný z výše zmíněných hodnot nezvolí, bude použita defaultní hodnota, na kterou je uživatel upozorňen tiskem.



Obrázek 14: Příklad ASCII art

Při zobrazování na různých platformách a v různých textových editorech je nutné případně nastavit správné encodování (na windows na ANSI).

2 Uložení zpracovaných dat

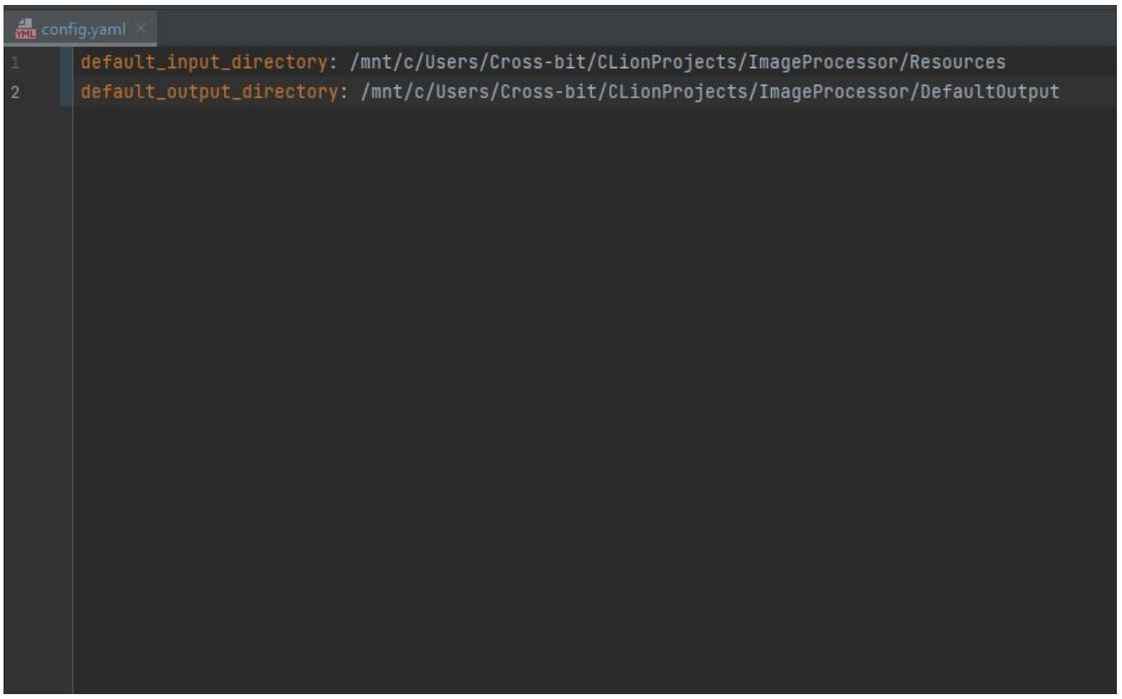
Poté co zvolený filtr zpracuje všechna data je uživatel přesměrován(vyjma filtru ASCII art) k uložení výstupních dat.
Nejprve uživatel zvolí formát ve kterém chce obrazová data uložit. V současné době Image processor nabízí pouze formát .jpg/.jpeg (přičemž při zadání volby .jpeg je i tak použit .jpg). Defaultní fallback je .jpg. (Nabídka je zde tedy spíše pro budoucí rozšíření, jednoduše ji lze nyní přeskočit)
Následně je jako druhý parametr nutné zadat cestu do výstupního adresáře. Je nutné zadat celou cestu. Pokud je cesta neplatná zvolí se fallback directory z config.yaml. Pokud tento adresář není nastavený, operace ukládání selže a zpracovaná data jsou ztracena.

2.1 Počáteční konfigurace

Aplikace umožňuje uživateli prostřednictvím souboru **config.yaml**(v adresáři s executable) před startem nastavit základní aplikace, které budou poté použita. Jedná se o:

- default_input_directory – Nastavení defaultního adresáře odkud se při startu aplikace načtou uživatelská data. (není rekurzivní)
- default_output_directory – Nastavení defaultního adresáře kam budou data ukládána pokud uživatel nespecifikuje při práci v aplikaci jinak.

Všechny cesty k adresářům je nutné psát jako absolutní cestu na lokálním uložišti.



Obrázek 15: Příklad **config.yaml**

3 Algoritmy

3.1 Jednoduché filtry

Nejzákladnější filtry

- linear/average grayscale,
- negative,

jsou založené na lineárním průchodu obrázku *pixel by pixel*, na které se aplikuje jednoduchá matematická operace.

3.1.1 Lineární grayscale

Aby výpočet dával smysl, tak je nutné aby intenzity jednotlivých složek byly přemapovány do lineárního prostoru, z tohoto důvodu je nutné nejprve provést tzv. gama expanzi. Ta z obrázku odstraní gama korekci, která mění mapování intenzity světla z lineárního na exponenciální. Více info např.: https://en.wikipedia.org/wiki/Gamma_correction
Různé barevné prostory používají různé gama křivky. V aplikaci je předpoklad práce s prostorem *sRGB*. Převodní funkce pro exspanzi má zde tvar:

$$C_{linear} = \begin{cases} \frac{C_{srgb}}{12.92}, & C_{srgb} \leq 0.04045 \\ \left(\frac{C_{srgb}+0.055}{1.055}\right)^{2.4}, & \text{jinak.} \end{cases} \tag{1}$$

Kde C_{srgb} představuje libovolnou z složek R_{srgb}, G_{srgb}, B_{srgb} v prostoru s gama korekci.

Hodnotu intenzity výsledného pixelu získáme jako lineární kombinaci:

$$Y_{linear} = 0.2126 \cdot R_{linear} + 0.7152 \cdot G_{linear} + 0.0722 \cdot B_{linear} \tag{2}$$

A tedy v novém obrázku bude každá ze složek, každého pixelu nahrazena Y_{linear}[2].

Na konec je potřeba pro správnou zpětnou interpretaci dat provést tzv. *gama kompresi*, neboli aplikovat gama korekci. Transformační funkce pro prostor sRGB má tvar:

$$Y_{srgb} = \begin{cases} 12.92 \cdot Y_{linear}, & C_{srgb} \leq 0.0031308 \\ (1.055 \cdot C_{srgb})^{1/2.4} - 0.055, & \text{jinak.} \end{cases} \tag{3}$$

Aplikace používá práva výše popsaný algoritmus, přičemž je možné si zde všimnout drobného prostoru pro zlepšení. Konkrétně lze teoreticky až 3x zredukovat velikost výstupních dat neboť nám nyní stačí ukládat pouze informaci o intenzitě daného pixelu namísto složek RGB. K tomuto účelu např. formát *.jpg* nabízí podporu monochromatického prostoru, který lze v použité knihovně libjpeg při ukládání nastavit. Tedy případné přidání této funkcionality by mělo jít snadno.

Filtr lze nalézt v adresáři: *ImageEffects/GrayscaleEffect*

3.1.2 Averaged grayscale

Averaged grayscale, neboli zprůměrované odstíny šedi jsou víceméně přímočaré a nevyžadují gama expanzi. Jedná se o aritmetický průměr všech tří složek RGB. Rovnice intenzity výsledného pixelu je

$$Y = (R + G + B)/3, \tag{4}$$

kde hodnoty R, G, B odpovídají hodnotám jednotlivých barevných kanálů a Y je výsledná monochromatická intenzita.
Tento přístup ač je jednoduše přímočarý, tak bohužel způsobuje ztrátu informace o jasu a tedy výsledné obrázky se poté mohou zdát oproti originálu na některých místech trochu světlejší a na některých místech zase tmavší.

Filtr lze nalézt v adresáři: *ImageEffects/GrayscaleEffect*

[2] Jednotlivé složky mají různé lineární koeficienty. To je proto, že na některé barvy je lidské oko citlivější než na jiné(a tudíž abychom dosáhli tíženého výsledku, tak intenzity některých složek chceme "utlumit"a naopak).

3.1.3 Negative

Obecně lze výpočet jednoduše spočítat(pro obrázek s 8 bit hloubkou) tak, že

$$C_{inv} = 255 - C_{org}, \tag{5}$$

přičemž $C_{inv} \in [0, 255]$ je inverz k $C_{org} \in [0, 255]$, což je složka RGB. Takto lze spočítat hodnotu pro každý pixel.

Zde v implementaci je navíc ještě přidán "intensity koeficient"označme – $k \in$, který dovoluje nastavit poměr původní barvy ku inverzní. Kde pro $k = 0$ dostaneme originální nezměněný obrázek. Při koeficientu $k = 0.5$ se poměry vyváží a celý obrázek bude v barvě šedé a při $k = 1$ dostaneme plnou inverzi.

Filtr lze nalézt v adresáři: *ImageEffects/Negative*

3.2 Filtry založené na konvolučním jádře

Jedná se především o:

- Box blur,
- Gaussian blur,
- Sobel (top, bottom, left, right).

,

Tyto filtry pracují s myšlenkou operace konvolunce, což je matematicky obecnější koncept, který sahá dále mimo pouze počítačovou grafiku. Ležérně můžeme říct, že se jedná o operátor, který vezme na vstupu dvě funkce f, g a vyprodukuje třetí $(f * g)$, která popisuje jak hodnoty jedné ovlivňují hodnoty druhé.$(f * g)(x)$ je poté hodnota konvolunce funkce f s jádrem popsaném funkcí g v bodě x.
Při zpracování obrazu funkce f představuje daný obrázek a funkce g konvolunční jádro(nebo též konvolunční matice). Obecná konvolunce je také definovaná pomocí integrálu. Při zpracování obrazových dat se však zabýváme tzv. diskrétní konvulencí definovanou pomocí sum. Případně více info: https://en.wikipedia.org/wiki/Convolution#Fast_convolution_algorithms

Aplikace implementuje triviální a nejnaivnější algoritmus výpočtu. Ten na vstupu bere obrázek velikosti $n * m$(n – výška, m – šířka) a *konvolunční matici* velikost k postupně pro každý pixel původního obrázku na základně hodnot *konvulenčí matice* určí hodnotu pixelu ve výsledném obrázku. Tento algoritmus je funkční nicméně doba výpočtu je značně neefektivní. Na každém pixelu se limitně stráví k^2 operací a pixelů je celkem $n * m$. Navíc tento výpočet musí paralelně počítat pro všechny tři barevné kanály. Celkem tedy máme odhad $O(n \cdot m \cdot k^2)$. Což je značně neefektivní. Pro malá jádra např. $k = 3$ je výpočet celkem snesitelný, ale při větších jádrech je čas výpočtu znatelně delší.
Nutno podotknout, že (pro většinu matic) existují mnohem rychlejší algoritmy, které lze pro obrázek o počtu pixelů n implementovat v čase $O(n \cdot log(n))$, nebo dokonce $O(n)$ (velikost jádra považujeme za konstantu).

Tyto filtry jsou k nalezení v adresáři: *ImageEffects/Convolution*
Přičemž zmíněný výpočet je implementován v:
ImageEffects/Convolution/ConvolutionProcessor.cpp
Jednotlivá jádra jsou poté v: *ImageEffects/Convolution/ConvolutionKernels*

3.2.1 Box blur

Box blur je jeden ze standardních rozmazávacích konvolunčních algoritmů. Má nicméně velmi jednoduchou matici velikost $3x3$:

$$K := \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix} \tag{6}$$

Neboli se dá jednoduše říct, že se jedná o aritmetický průměr pixelů nad kterými se matice při výpočtu nachází.

3.2.2 Gaussain blur

Jeden z hlavních rozdílů mezi Gaussovským blur a Box blur je velikost konvolunční matice n, který může nabývat libovolného lichého přirozeného čísla většího 3. (teoreticky i 1, ale to by byla identity)
Další specificitou, jak již název vypovídá, je že hodnoty matice odpovídají distribuci Gaussovského rozdělení. Konkrétně jsou tyto hodnoty rovnoměrně distribuovány symetricky od prostředního prvku do všech stran[3]. Dá se tedy říct, že čím větší velikost matice je, tím jemnost/kvalita a intenzita výsledného rozmazání bude.
Intenzita výsledného Gaussovského rozmazání však souvisí ještě s druhým parametrem funkce. Tím je σ – směrodatná odchylka(v angličtině standard deviation), která ovlivňuje jak "strmé"Gaussovské rozložení bude, resp. jak rychle budou hodnoty od středu matice klesat k nule. Čím menší σ zvolíme tím strmější funkce bude, a tím menší efekt dané rozmazání bude mít. Naopak čím větší σ bude, tím intenzivnější Gaussian blure bude. Nicméně v jistém okamžiku(zhruba $\sigma \approx n$) již zvyšování této hodnoty přestane mít na intenzitu rozmazání takový vliv. Pokud bychom chtěli docílit většího rozmazání, je nutné zvlolit větší n nebo efekt aplikovat opakovaně.

Gaussovská funkce v dvou rozměrech má tvar

$$f(x,y) = \frac{1}{\pi 2\sigma^2} \cdot \exp\left(\frac{x^2+y^2}{2\sigma^2}\right)^{-1} \tag{7}$$

Přičemž při generování matice jsou poté ještě hodnoty normalizovány, aby jejich suma dávala 1. Jinak by docházelo i k modifikaci jasu obrázku.

3.2.3 Sobel

Sobel se často používá při detekci hran. Konkrétně zaznamenává rychlé změny intenzity/přechody mezi tmavými a světlými místy v obraze. Sobel má matici velikosti $3x3$. Obecně můžeme definovat rovnou 4 varianty tohoto filtru, které jsou:

Left sobel mající matici:

$$S_l := \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \tag{8}$$

Right sobel s maticí:

$$S_r := \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \tag{9}$$

Top sobel:

$$S_t := \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{10}$$

Bottom sobel:

$$S_b := \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{11}$$

[3]Což je mimo jiné i důvod proč musí být velikost matice liché číslo.

3.3 Generování ASCII art

Existuje několik různých metod, jak převést obrazová data na ASCII art. Při generování se často, stejně jako v tomto algoritmu, vstupní obrazová data nejprve převedou do grayscalu, neboť obsahuje již zprůměrované informace ze všech 3 barevných kanálů o celkové intenzitě jednotlivých pixelů[4]. Zde je k tomuto účelu použit algoritmus pro lineární grayscale.
Dalším vstupním parametrem je vstupní abeceda, jejíž znaky budou použity např.:

$$\Sigma = \$\%8\&WM\#oahkbdpqwmZO0QLCJUYXzcvunxrjft/\|()1[]?-\_+\ <>i!;:,‘’.$$

Přičemž je důležité udržet ”optickou hustotu”jednotlivých znaků, ve smyslu kolik pixelů na monitoru zabere samotný znak. V tomto algoritmu je zvolena konvence, že znaky s nižším indexem jsou opticky hustší a naopak s vyšším indexem jsou opticky ”řidší”.

V neposlední řadě je potřeba parametrizovat to, jak velké oblasti obrázku bude každý znak odpovídat. Možnost by byla zvolit co pixel to jeden znak. Pro velké obrázky by však tento převod by nemusel být vhodný, a proto tyto oblasti definují dva parametry:

- $ColsX \in [1, \{imgWidth\}]$ – udává kolik znaků se může maximálně vejít na řádek[5], neboli udává kolik pixelů na šířku pokreje jeden znak.
- $ScalingFactor \in (0, 1]$ – určuje kolik pixelů na výšku pokreje jeden znak. Při 1 je $\{tileHeight\} == \{tileWidth\}$, neboli 1 znak pokreje právě jeden pixel.

Alternativně by šlo specifikovat $\{tileHeight\}$ a $\{tileWidth\}$ přímo(na což je v třídě algoritmu připravený overload), nicméně se ukázalo, že tento přístup není moc praktický z pohledu uživatele.

Algoritmus poté prochází data obrázku po těchto oblastech a intenzity obsažených pixelů zprůměruje. Následně nalezne mapování na odpovídající znak v Σ a pošle na výstup.

4 Kompozice kódu

Kód je členěn celkem do čtyř sekcí příslušnými adresáři. Jedná se o adresáře:

- ImageEffects – Poskytuje hlavní API pro práci s filtry. Hlavní end point zde představuje *ImageEffectFactory.cpp* poskytující nabídku pro tvorbu jednotlivých filtrů. Při práci využívá rozhraní a modely z ImageLibrary.
- ImageLibrary – Poskytuje rozhraní pro R&W operace s obrázky a jejich reprezentaci v programu.
- Services – Představuje služby především určené pro načítání externích (ne obrázkových) dat jako např. načtení a pars startovní configurace *config.yaml*.
- UserMenu – Vytváří klientské rozhraní pro práci s aplikací.

[4]Stejně tak, lze samozřejmě data generovat i na základě např. hodnoty v konkrétním jednom kanálu.
[5]Tento parametr je lehce zavádějící, protože: $\{tileWidth\} = \{imgWidth\}/ColsX$ a kvůli zaokrouhlovacím chybám se pak počet znaků na řádku může značně lišit.

5 3rd party knihovny

Veškerý 3rd party software použitý v aplikaci je k nalezení v standardním adresáři *External* jako statické knihovny (tedy i společně s příslušnými header files).

5.1 libjpeg

Pro načítání obrázků formátu *.jpg* aplikace používá opensource knihovnu **libjpeg**. Jako poznámku je také dobré zde zmínit, že ač je tato knihovna historicky zažitá, v současné době existuje její novější varianta libjpeg-turbo, která je dokonce o něco rychlejší a poskytuje i uživatelsky příjemnější rozhraní pro práci. *Libjpeg* byla použita, neboť je časem nejvíce ověřená a také protože straň rychlosti je pro tuto aplikaci dostačující.
Více info např.:
https://en.wikipedia.org/wiki/Libjpeg

5.2 yaml-cpp

Pro načtení a parse vstupního konfiguračního souboru byla použita opensource knihovna *yaml-cpp*. Více info např.:
https://github.com/jbeder/yaml-cpp

6 Závěr

Software poskytuje celkem jednoduché a ovladatelné rozhraní pro editaci s obrázky, čili hlavní cíl byl splněn. Nejvíce bych asi vystihl to, že aplikace je víceméně dobře navržená straň dalšího rozšiřování a přidávání nových funkcionalit.

Nicméně i tak je v aplikaci několik míst, které bych změnil či dnes udělal jinak. Myslím, že celé uživatelské rozhraní by bylo dobré udělat znova. Byl to "můj osobní pokus"o vytvoření OOP konzolového menu, jehož architektura by šla do budoucna dále rozvíjet. Ve výsledku tuto myšlenku realizuje a v mnoha směrech splňuje SOLID. Co se mi však nelíbí je rozsáhlost celého systému. S OOP vždy přichází cena za overhead tříd, struktur, apod..., nicméně zde se mi zdá, že rozsáhlost není příliš úměrná tomu, co tento systém ve skutečnosti klientovi nabízí. Zároveň se mi nelíbí, že jsem příliš neuhlídal konzistentnost v samotném vypisování, či čtení příkazů uživatele. Tuto část by chtělo ještě refaktororvat a vytvořit sjednocené API, kde veškerá komunikace s konzolí byla implementována a všechny prvky uživatelského menu by pracovali pouze s tímto jedním API. Nicméně i tak si myslím, že tento prvek přinesl další zbytečnou komplexitu do projektu, kvůli které jsem měl méně času se zabývat jádrem aplikace. Myslím si, že použít nějaký již vytvořený parser příkazů z příkazové řádky by byl naprosto dostačující a v mnoha ohledech lepší a flexibilnější.

Co se rozhraní pro práci s obrázky a filtry týče tak si myslím, že by zde byl rozhodně ještě prostor pro zlepšení a mnoho věcí by šlo udělat lépe. Nicméně ve výsledku si myslím, že současná verze je pro jednoduché základní použití postačující. Ve výsledku jsem s projektem celkem spokojený, ale rozhodně je zde velký prostor pro zlepšení.