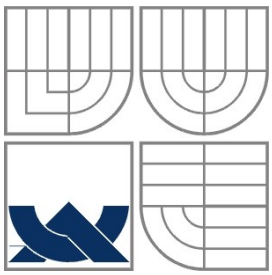
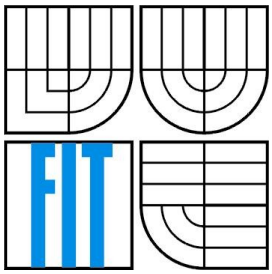


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GEOMETRICKÉ TRANSFORMACE OBRAZU

GEOMETRICAL IMAGE TRANSFORMS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

JAN HAVELKA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. MICHAL ŠPANĚL

BRNO 2008

Abstrakt

Cílem bakalářské práce je nastudování základů zpracování obrazu, především pak lineárních transformací obrazu a interpolací obrazových bodů. Za účelem bližšího seznámení je vhodné provést implementaci základních geometrických transformací a alespoň několika druhů různých interpolačních metod.

Klíčová slova

transformace, interpolace, obraz, rastr, rastrová grafika, rotace, velikost, posunutí, bilineární, bikubická, spline, sinc

Abstract

This bachelor's thesis is about introducing to basics of image processing, mostly with linear image transformations and interpolation. For the purpose of the closer understanding is properly to implement some basic geometrical image transformations and some different interpolation methods.

Keywords

Transformation, interpolation, image, raster, raster graphics, rotation, scale, translation, bilinear, bicubic, spline, sinc

Citace

Jan Havelka: Geometrické transformace obrazu, bakalářská práce, Brno, FIT VUT v Brně, 2008

Geometrické transformace obrazu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Michala Španěla
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jméno Příjmení
Datum

Obsah

1 Úvod

S geometrickými transformacemi se setkáváme téměř na každém kroku našeho běžného života. Jsme doslova obklopeni grafickými motivy všech možných druhů, ať už jde o reklamu na billboardech, letáky v obchodech nebo jen pořady v televizi. Je velmi pravděpodobné, že každý obraz, který vidíme, prošel rukama nějakého grafika a je také velmi pravděpodobné, že na něm byla použita právě ona geometrická transformace.

Jakmile se zamyslíme nad tím, kde všude se geometrické transformace používají, rychle nám dojde, že se právem řadí do skupiny nejpoužívanějších operací v počítačové grafice. Dělí se na lineární a nelineární. Do skupiny lineárních patří například otočení obrazu, změna velikosti, posunutí, atd. Nelineární transformace se využívají například k modelování různých složitějších objektů nebo k warppingu.

Cílem bakalářské práce je seznámení se se základy zpracování obrazu, především pak s lineárními transformacemi obrazu a interpolacemi obrazových bodů. Za účelem bližšího seznámení je vhodné provést implementaci základních geometrických transformací a alespoň několika druhů různých interpolačních metod. Pomocí následného testování konečné aplikace a porovnávání dílčích výsledků si budeme schopni vytvořit objektivní názor na problematiku geometrických transformací, což by mělo být podstatou zadání bakalářské práce.

V následující kapitole bych rád uvedl veškeré teoretické podklady, které jsou důležité k důkladnému pochopení zadané problematiky nebo jsou klíčové k vypracování aplikace. Návrh aplikace je popsán v kapitole třetí. Ve čtvrté je pak konkrétní implementace programu. V kapitole páté a předposlední je souhrn nejzajímavějších provedených testů a jejich výsledků. Poslední částí práce je zhodnocení provedených testů a závěry z nich vyplývající.

2 Teoretický základ

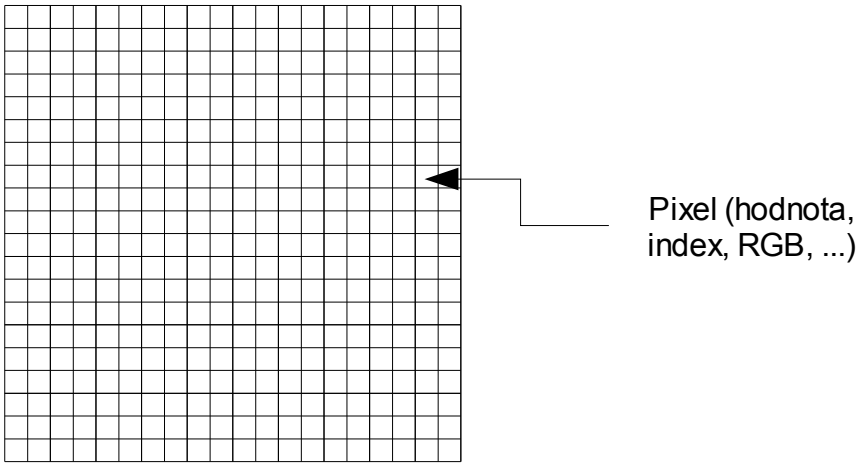
2.1 Počítačová grafika - úvod

Jde o zcela samostatnou kategorii grafiky. Jedná se o celý obor informatiky. Ten si klade za úkol s pomocí počítače analyzovat nebo vytvářet grafické obrazové informace. Analýza těchto informací se nazývá Počítačové vidění. Vytváření grafické informace se většinou děje například na základě nějakého matematického modelu, který daný obraz popisuje. Počítačovou grafiku dělíme podle několika hledisek. Například podle dimenze (2D, 3D) nebo podle způsobu zobrazení výstupních dat (rastrová, vektorová).

2.1.1 Rastrová grafika

Je zřejmé, že u rastrové grafiky se bude zacházet s jakýmsi „rastrem“. Pod tímto pojmem rozumíme 2D nebo 3D matici, do které provádíme zobrazení. Jednu buňku matice nazýváme pixelem.

Rastrová grafika se zabývá vlastně zobrazením grafických informací do této námi vytvořené matice. Například přímka nebude zadána dvěma koncovými body, ale každým bodem, kterým prochází. V matematice toto není možné kvůli nekonečně malému zobrazení jednoho bodu. V počítačové grafice se vychází z pevné velikosti bodu resp. buňky matice. Jedná se o jakýsi průmět zobrazovaného objektu do rastrové matice. Vzhledem k tomu, že monitor, jako zobrazovací zařízení, je pouze rastr o daném rozlišení, tak všechny informace zobrazované na monitoru jsou dílem rastrové grafiky.



Obrázek 1: Rastrová matice

2.1.2 Barva

„Barva je vjem, který vytváří viditelné světlo dopadající na sítnici lidského oka. Barevné vidění lidského oka zprostředkují receptory zvané čípky trojího druhu – citlivé na tři základní barvy: červenou, zelenou a modrou.“ [1]

V počítačové grafice je barva většinou znázorněna vektorem právě těchto tří barev (RGB). Každé barvě se přiřadí hodnota od 0 do 255, což představuje 8 bitů. Složením těchto tří barev dostaneme 24 bitů. Tento vektor nazýváme barevnou hloubkou. Pokud každé složce vektoru přiřadíme menší barevné rozmezí, můžeme samozřejmě získat i menší barevnou hloubku. Rozlišujeme základní barevné módy. Prvním z nich je indexový mód, který je spojen s používáním tzv. barevné palety. Hodnota pixelu v tomto módu nenese tedy přímo barvu, ale ukazatel do tabulky barev – palety. Tímto způsobem se reprezentuje tzv. pseudocolor, kdy paleta barev slouží jako převodní tabulka pro zobrazení nižšího počtu barev než 2^{24}. Dalším barevným módem je obraz truecolor. Ten udává, že každý pixel v obraze si nese v sobě informace o každé barvě sám. Nepoužívá žádný odkaz do barevné palety. Posledním módem je tzv. direct color, který pro jeden pixel definuje tři ukazatele, každý do samostatné barvy. Tento naposled jmenovaný mód je výhodný zejména při barevných korekcích obrazu, protože nevyžaduje změnu samotné hodnoty pixelu, ale pouze přepsání barevné palety.

Další možností je reprezentace obrazu pomocí stupnice šedé barvy (grayscale). Převod do stupňů šedi se provádí buďto opět změnou barevné palety v případě, kdy máme obraz reprezentován ukazateli nebo převodem každého pixelu obrazu pomocí vzorce pro intenzitu barvy (vzorec 1). Je to umožněno díky nedokonalosti lidského oka, které je různě citlivé na jednotlivé hlavní barvy. Nejvíce jsme citliví na zelenou barvu, potom na červenou a nejméně na modrou.

$$I = 0.299R + 0.587G + 0.114B$$

Vzorec 1: Intenzita barvy

Pomocí vzorce získáme intenzitu barvy v rozmezí osmi bitů (256 stupňů šedi), kterou použijeme pro každý barevný kanál. Výsledná barva bude znázorněna jako vektor tří stejných čísel. Pro různé, například lékařské účely, může být stupnice šedi i vícebitová, ale pro konečné zobrazení musí být stejně převedena na intenzitu 0 – 255.

1 http://cs.wikipedia.org/wiki/Barva

2.2 Transformace ve 2D

Pomocí transformací se provádí, jednoduše řečeno, geometrické úpravy obrazu. Může se jednat například o otočení obrazu dle zadaného úhlu. Nebo zvětšování resp. zmenšování obrazu dle měřítka. Dále například posun obrazu o daný počet pixelů všemi směry.

Tyto úpravy se provádějí pomocí transformačních matic. Každý pixel obrazu se musí vynásobit určitou maticí. Tím se zjistí jeho nové souřadnice, jeho poloha v novém obrazu.

2.2.1 Homogenní souřadnice

Umožňují nám pracovat se všemi transformacemi pomocí maticového zápisu.

Podle definice jsou homogenní souřadnice bodu s kartézskými souřadnicemi $[x, y]$, uspořádanou trojicí $[X, Y, w]$. Kde w je tzv. váha bodu. Platí, že $x = X / w$ a $y = Y / w$. V našem případě je $w = 1$, protože se nám jedná o lineární transformace.

2.2.2 Rotace

Rotace provádí otočení jednoho bodu $P(x, y, 1)$ v rovině o úhel α. Středem otáčení (osou) je v tuto chvíli počátek souřadnic. Výsledný bod, již otočený, bude mít souřadnice $P'(x', y', 1)$.

Zápis pomocí vzorce: $$P'(x', y', 1) = P(x, y, 1) * T$$

Maticový zápis transformace: $$[x', y', 1] = [x, y, 1] * \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Jednotlivé transformační matice:

$$R = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Vzorec 2: Rotační matice

$$R^{-1} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Vzorec 3: Rotační inverzní matice

2.2.3 Měřítko

Změna měřítka (angl. scale) je vlastně „zoom“ obrázku. Provádí se opět pomocí transformačních matic tak, jak uvádí následně vypsané vzorce. Bodem P se rozumí souřadnice pixelu z původního obrázku. Bodem P' potom souřadnice pixelu v novém obraze. S_x a S_y jsou koeficienty zvětšení nebo zmenšení. Pokud $S_{xy} \in (0, 1)$, pak se jedná o zmenšení. Pokud $S_{xy} \in (1, \infty)$, pak o zvětšení.

Zápis pomocí vzorce: $$P'(x', y', 1) = P(x, y, 1) * T$$

Maticový zápis transformace: $$[x', y', 1] = [x, y, 1] * \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Jednotlivé transformační matice:

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Vzorec 5: Matice měřítka

$$S^{-1} = \begin{bmatrix} 1/S_x & 0 & 0 \\ 0 & 1/S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Vzorec 4: Inverzní matice měřítka

2.2.4 Posun

Posunutí (angl. translation) je jednou z nejjednodušších transformací vůbec, nebo alespoň podle zápisu svojí matice. Jedná se o posunutí každého pixelu v obraze o předem zvolené množství pixelů v obou osách. Bodem P se rozumí souřadnice pixelu z původního obrázku. Bodem P' potom souřadnice pixelu v novém obraze. Proměnné d_x a d_y určují o kolik bodů se má obraz posunout.

Zápis pomocí vzorce: $$P'(x', y', 1) = P(x, y, 1) * T$$

Maticový zápis transformace: $$[x', y', 1] = [x, y, 1] * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ d_x & d_y & 1 \end{bmatrix}$$

Jednotlivé transformační matice:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ d_x & d_y & 1 \end{bmatrix}$$

Vzorec 6: Matice pro posunutí

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -d_x & -d_y & 1 \end{bmatrix}$$

Vzorec 7: Inverzní matice pro posunutí

2.2.5 Skládání transformací

Skládáním se rozumí použití několika transformací v jednom kroku. K tomuto faktu nám výrazně pomáhá maticový zápis. Díky němu stačí původní pixel vynásobit potřebným počtem matic a získáme pixel výsledný.

$$P' = P * T$$

Kde T je součin všech matic, které chceme skládat.

Při skládání transformací záleží na pořadí, protože násobení matic není komutativní. To znamená, že záleží na tom, jestli provedeme nejprve rotaci a potom posunutí nebo naopak.

2.3 Interpolace

Interpolace obrazu je metoda, kdy se při zobrazování dat dopočítávají chybějící body obrazu. Ty vznikají například při zvětšování obrazu. Když máme obraz dvojnásobně zvětšit, tak se musí teoreticky dvojnásobně zvětšit každý pixel obrazu. Problém je, že velikost rastru zůstává stále stejná. V situaci jednoho pixelu a dvojnásobného zvětšení by se tedy musel roztáhnout na dva pixely v každé ose, dohromady tedy čtyři body. Konkrétní barvu ale známe jenom u toho původního. Další tři body musíme tedy dopočítat. K tomu nám slouží právě interpolace.

Výpočet barvy konkrétního pixelu se vždy musí provádět podle jeho okolních pixelů. Jak je okolí velké, záleží vždy na dané interpolaci. Každému pixelu v této oblasti přiřadíme jeho vlastní váhu. Váhou rozumíme číslo, které určuje nakolik si budeme všímat jednoho konkrétního pixelu.

Finální barva se potom spočte jako součet barev všech pixelů v daném okolí vynásobených vždy svou vlastní váhou.

2.3.1 Metoda nejbližšího souseda

Metoda nejbližšího souseda je interpolací nultého řádu. To znamená, že hodnota daného, námi vyžadovaného, bodu se s zkopíruje od pixelu, který je nejblíže.

Váha bodu se spočítá metodou „buď a nebo“. Bod který je blíž dostane váhu jedna a bod, který je dál váhu nula.

2.3.2 Bilineární interpolace

Bilineární interpolace je druhou nejjednodušší metodou používanou k odhadování konečné barvy hledaného bodu. Jestliže metoda nejbližšího souseda byla nultého řádu, pak bilineární interpolace je řádu prvního. Využívá se v ní lineární funkce, která je pro představu reprezentována přímkou.

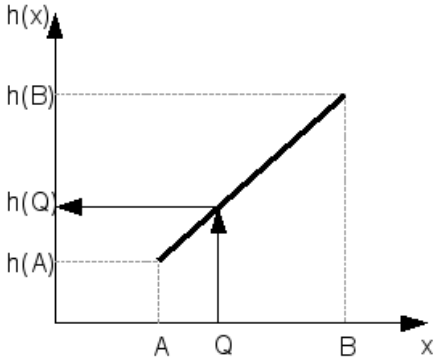
Proč BI – lineární? Předpona BI je zde z jednoduchého důvodu. Pracujeme ve 2D prostoru, máme dvě osy a pro každou z nich můžeme použít jednoduchou lineární interpolaci.

Pokud hledáme hodnotu bodu Q, pak je zřejmé, že okolo něj musí být právě dva body A a B. Funkční hodnoty, neboli v našem případě konkrétní barva pixelu, jsou znázorněny v grafu jako h(A), h(B). Námi hledaná hodnota bodu Q bude ležet někde na úsečce mezi krajními body A a B, jak názorně ukazuje obrázek 2. Váha pro oba krajní body se spočte podle vzorce 8, kde w je váha a x je desetinná část souřadnice podle vybrané dimenze.

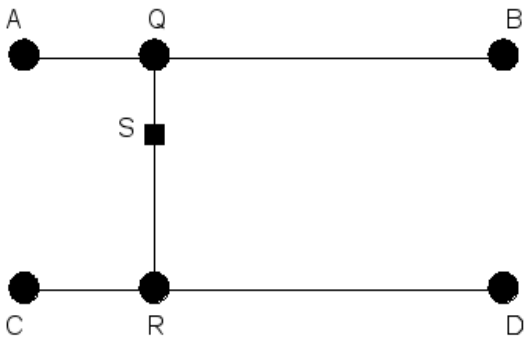
$$w = 1 - x$$

Vzorec 8: Váha

Bilineární interpolací se rozumí použití lineární metody ve dvou osách. Postup pro výpočet konečného bodu je znázorněn na obrázku 3. Písmena použitá pro označení bodů na obrázku značí jejich funkční hodnoty, resp. jejich barvy. Nejprve se pomocí lineární interpolace spočtou body Q a R, každý podle svých sousedů ve vodorovné ose. Námi od počátku hledaný bod S se pak spočte stejným způsobem pomocí lineární funkce, kde krajními body úsečky budou právě nalezené body Q a R.



Obrázek 2: 1D lineární interpolace



Obrázek 3: 2D bilineární interpolace

2.3.3 Bikubická interpolace

Jedná se o polynomiální interpolaci třetího stupně. Výpočet se provádí pomocí kubického polynomu a je k němu tudíž potřeba čtyř bodů pro každou počítanou osu. Ve 2D prostoru jde o dvě osy, takže celkem šestnáct pixelů.

Největším rozdílem oproti předchozí metodě je, že nebudeme v grafu interpolace používat přímku, ale kubickou křivku (polynom třetího stupně) zadanou právě čtyřmi body v jedné dimenzi. Pro nejpřesnější výpočet je nutné, aby hledaný bod ležel mezi dvěma prostředními pixely. Toto schéma nám přiblíží obrázek 4.

Postup řešení je velice podobný jako u bilineární interpolace. I zde použijeme výpočetní metodu nejprve na řádky a později na sloupec dílčích výsledků. Postupně každým řádkem proložíme kubickou funkci a vypočteme hodnotu v bodě, který hledáme. Přičemž pro určení jeho polohy nás zajímá pouze vodorovná souřadnice, protože se nacházíme v jedné dimenzi. Čtyři výsledné body Q1-4 poslouží jako řídící body pro křivku v druhé dimenzi. Toto ilustruje obrázek 5.

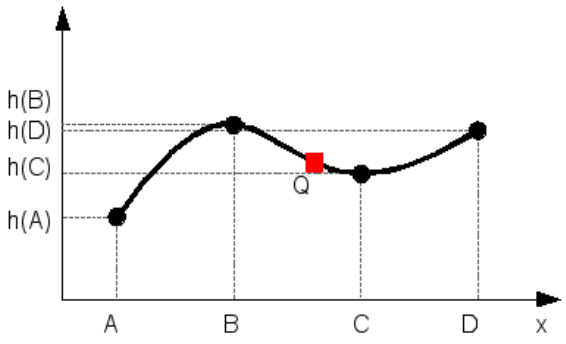
Spočtení váhy již není jednoduché. Celkem budeme v jedné dimenzi počítat váhu pro čtyři pixely. Vezmeme první z nich, v našem případě bod A, a zjistíme jak daleko je od interpolovaného bodu. Tato vzdálenost by měla být od nuly do dvou, protože interpolovaný bod leží mezi dvěma prostředními pixely. Pokud vzdálenost padne mezi nulu a jedničku použijeme pro výpočet váhy vzorec 9. Analogicky pro vzdálenost od jedné do dvou použijeme vzorec 10.

$$w = ((A+2.0)x - (A+3.0))x^2 + 1.0, \qquad 0<x<1$$

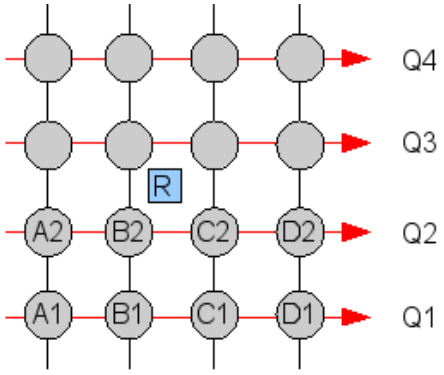
Vzorec 9: Váha pro x od 0 do 1

$$w = ((Ax-5.0A)x + 8.0A)x-4.0A, \qquad 1<x<2$$

Vzorec 10: Váha pro x od 1 do 2



Obrázek 4: 1D kubická interpolace



Obrázek 5: 2D Bikubická interpolace

2.3.4 Spline interpolace

Další alternativou pro interpolaci je použití funkce spline. Díky ní se dají rozložit složité polynomiální funkce na jednodušší polynomy nižších řádů. To se nejvíce projeví u funkcí vyšších řádů.

Spline je podle definice „polynom po částech“. To znamená, že definiční obor složitého polynomu se rozdělí na intervaly a na každém z nich se definuje polynom nižšího řádu. Definice splinu uvádí jedinou podmínku a tou je hladká návaznost jednoduchých polynomů. Tedy nulovou první derivaci v krajních bodech intervalů.

Obvykle se na dílčích intervalech počítá s přirozeným kubickým splinem. Jeho obecný tvar ukazuje vzorec 11. Obecně se dá použít pro libovolně složitý polynom. S vyšším řádem původního polynomu roste také oblast bodů, se kterou se pracuje. Nejčastěji se provádí interpolace pomocí splinu na oblasti šestnácti nebo třiceti šesti pixelů.

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

Vzorec 11: Přirozený kubický spline

$$w = \left(\left(x - \frac{9}{5}\right)x - \frac{1}{5}\right)x + 1.0, \qquad 0 < x < 1$$

$$w = \left[\left(-\frac{1}{3}(x-1) + \frac{4}{5}\right)(x-1) - \frac{7}{15}\right](x-1), \qquad 1 < x < 2$$

Vzorec 12: Váha pro spline16

$$w = \left[\left(\frac{13}{11}x - \frac{453}{209}\right)x - \frac{3}{209}\right]x + 1, \qquad 0 < x < 1$$

$$w = \left[\left(-\frac{6}{11}(x-1) + \frac{270}{209}\right)(x-1) - \frac{156}{209}\right](x-1), \qquad 1 < x < 2$$

$$w = \left[\left(\frac{1}{11}(x-2) - \frac{45}{209}\right)(x-2) + \frac{26}{209}\right](x-2), \qquad 2 < x < 3$$

Vzorec 13: Váha pro spline36

2.3.5 Sinc interpolace

V této interpolaci se na rozdíl od předchozích nevyužívá polynomiálních funkcí. Hlavní výpočet je reprezentován pomocí Lanczosovy funkce definované ve vzorci 14. Název interpolace je odvozen od funkce sinc(x), kterou Lanczos používá pro svojí metodu. Funkce sinc je definovaná ve vzorci 15.

Ze vzorců by se mohlo zdát, že výpočet této metody je jednodušší než v předchozích případech, bohužel, opak je pravdou. Aby byla interpolace Sinc opravdu účinná, provádí se na rastru o velikosti 16x16, tzn. 256 pixelů. Lehce se dá spočítat kolik bodů celkem se musí takto vypočítat například u malého obrázku o velikosti 200x200 pixelů. Je pravdou, že tato metoda je nejlepší a nejvěrnější interpolací vůbec, ale zároveň budeme nuceni si na ni chvíli počkat. Rychlost dokončení interpolace je mnohonásobně delší než například u interpolace bikubické.

$$
\begin{aligned}
L(x) &= sinc(x)\,sinc\left(\frac{x}{a}\right), && -a<x<a, \quad x\neq 0\\
L(x) &= 1, && x=0\\
L(x) &= 0, && jinde
\end{aligned}
$$

Vzorec 14: Definice Lanczosovy funkce

$$sinc(x) = \frac{\sin(\pi x)}{\pi x}$$

Vzorec 15: Definice funkce sinc(x)

$$w = \left[\frac{\sin(\pi x)}{(\pi x)}\right] * \left[\frac{\sin\left(\frac{\pi x}{8}\right)}{\left(\frac{\pi x}{8}\right)}\right]$$

Vzorec 16: Váha pro interpolaci sinc

3 Návrh

V této kapitole bych se rád zabýval návrhem již konkrétní aplikace, která byla námětem bakalářské práce. V zadání stojí geometrické transformace obrazu. Bude tedy potřeba navrhnout mechanizmus, pomocí kterého budeme transformovat vstupní obraz. Tento bude reprezentován knihovnou v jazyce C++. Měl by umožňovat alespoň tři základní transformace a to: otáčení (rotation), posunutí (translation) a změnu měřítka (scale).

Abychom mohli transformace realizovat, je nutné si vybrat nějaký nástroj, s jehož pomocí budeme s obrazem manipulovat. Nástroj by měl obsahovat funkce pro načtení rastrového obrazu ze souboru, převedení barev na stupně šedi, vykreslení obrazu na obrazovku nebo uložení do souboru alespoň v některém grafickém formátu.

Dále bude nutné navrhnout knihovnu, která bude obsahovat všechny dostupné druhy interpolačních metod.

Provedení celé transformace bude mít za úkol jednoduchý program. Ten nejprve načte obraz pomocí specifického nástroje a předá ho ke zpracování knihovně pro transformace obrazu. Spolu s obrazem jí předá také informace o tom, jakou chce uživatel použít transformaci a jakou interpolační metodu. Transformační knihovna bude úzce spolupracovat s knihovnou pro interpolační metody, ze které vybere vždy tu správnou. Do programu se po provedení všech uživatelem definovaných akcí vrátí nově vytvořený obraz. Posledním úkolem programu bude tento nový obraz dostat k uživateli na obrazovku nebo ho uložit do souboru.

3.1 Návrh knihovny pro transformace

Knihovna bude reprezentována pomocí jazyka C++, což bylo součástí zadání bakalářské práce. Bude obsahovat tři základní transformace obrazu. Tyto transformace budou definovány pomocí maticového zápisu, především kvůli snadnému skládání jednotlivých transformací do sebe.

Srdcem celé knihovny bude jedna hlavní třída, která se bude o vše starat. Bude obsahovat metody pro jednotlivé geometrické transformace, ale také metody pro jejich skládání. Dále musí obsahovat rozhraní pro komunikaci s okolními třídami, především pak se třídou pro interpolace.

Jak bude třída konkrétně vypadat bude detailně popsáno v kapitole implementace.

3.1.1 Princip transformací

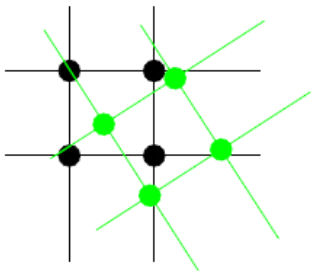
Jak už bylo zmíněno výše, budeme používat maticový tvar zápisu transformací. Teorie k tomuto návrhu je popsána v kapitole 2.2.

Je nutné si uvědomit jakým způsobem budeme samotný výpočet realizovat. V nejčastěji dostupné literatuře se uvádí vždy jednoduchý princip transformace pomocí součinu vektoru původního pixelu a příslušné transformační matice. Toto řešení je po teoretické stránce naprosto v pořádku. V praxi se však dopouští velkého defektu ve výstupním obraze. Pokud bychom postupovali přesně tak, jak říká teorie, tj. vezmeme každý vektor pixelu a vynásobíme ho danou maticí, vždy budeme mít naprosto stejný počet pixelů ve výstupním obrázku jako v obraze vstupním. To je velký problém, protože ve chvíli, kdy budeme obraz například zvětšovat, tak se pixely sice přenesou přesně jak mají být, ale mezi nimi budou mezery vyplněné barvou pozadí obrazu nového. Jako názorný příklad se dají použít obyčejné puzzle. Každý dílek skládačky by se dal považovat za jeden pixel. Jenže dílů skládačky je omezený počet, proto bychom pro její zvětšování museli jednotlivé kostičky dát dál od sebe. V tu chvíli by vznikly mezi jednotlivými dílky mezery a obraz by neodpovídal předloze. Tohoto principu se využívá především ve vektorové grafice, kdy se přenášejí pouze konkrétní body daných objektů.

Správné řešení pro rastrovou grafiku využívá pro svoji funkčnost tzv. interpolaci (viz kapitola o teorii interpolací 2.3). Princip funguje vlastně obráceně než výše zmíněný. Vezmeme postupně všechny body nového obrázku, a ty vynásobíme konkrétní inverzní maticí. V podstatě se bude jednat o dotazovací princip činnosti. Vezmeme první pixel nového obrazu a zeptáme se, kam patřil v obraze původním. Vyjdou nám dvě desetinná čísla určující přesnou polohu daného bodu v původním obrazci. V tuto chvíli nastoupí interpolace, která dokáže zjistit přesnou barvu i na desetinných souřadnicích původního obrazu. Tuto hodnotu barvy potom vrátíme prvnímu pixelu v obrazu novém a vezmeme na řadu další pixel. Toto provádíme pro všechny pixely nového plátna.

3.1.2 Rotace

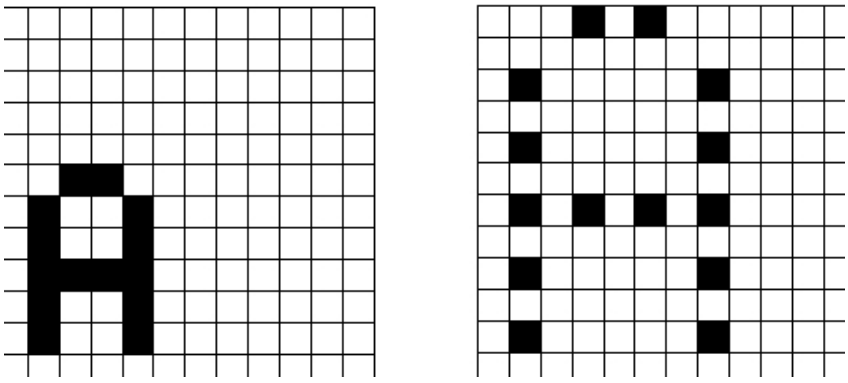
Princip provedení zůstává stejný jako je popsáno v předchozí kapitole. Je použita konkrétní inverzní matice pro rotaci (viz vzorec 3). Při této transformaci dochází k nejnázornějšímu využití interpolace jako takové. Při otočení obrazu totiž pixely téměř nikdy „nezapadnou“ do původního rastru. Tuto situaci demonstruje obrázek 6. Zelená mřížka ukazuje rastr nového obrazce. Pozice jeho pixelů se právě neshodují se základním rastrem obrazu (černá mřížka). O to, aby pixely „zapadly“ tam, kam mají, se postará vybraná metoda interpolace.



Obrázek 6: Situace při rotaci

3.1.3 Změna měřítka

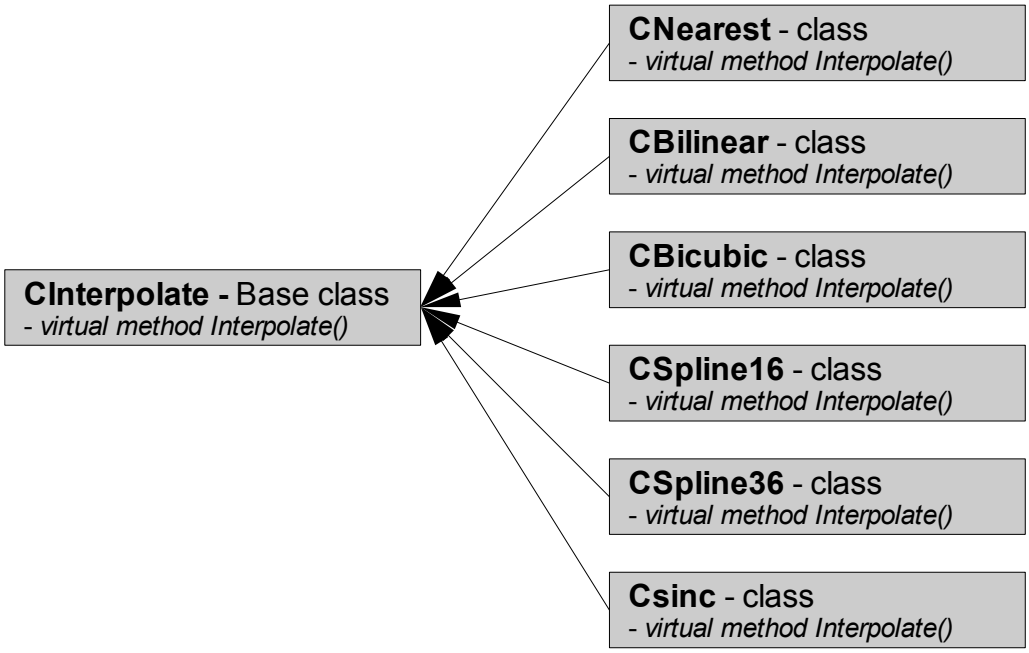
Pro změnu měřítka resp. velikosti obrazu se využívá inverzní matice definované ve vzorci 4. Ukázku použití normální transformační matice místo inverzní ukazuje obrázek 7. Mřížka představuje zvětšený rastr a černé čtverce – pixely. Pokud bychom nepoužili inverzní matici, náš výsledný obraz by měl stejný počet pixelů, jako původní.



Obrázek 7: Ukázka použití normální (neinverzní) matice

3.2 Návrh knihovny pro interpolace

Hlavním úkolem této knihovny bude reprezentace všech dostupných interpolačních funkcí. Bude tvořit jednotný celek, který bude schopen pracovat se všemi metodami. V této podkapitole se pokusím vysvětlit samotný princip interpolací a následně všechny jednotlivé metody v praxi.



Obrázek 8: Schéma dědičnosti

Obrázek 8 znázorňuje vnitřní strukturu knihovny. Dědičnost je v tuto chvíli na místě, protože definice dědičnosti vyjadřuje vztah „je nějaká“. Tento vztah přesně odpovídá našemu případu. CBilinear je nějaká interpolace.

3.2.1 Princip interpolací

Principem činnosti interpolací se má na mysli, jak budou ve své podstatě pracovat. Interpolace jako taková by měla pracovat jako jedna jednoduchá metoda, která bude přijímat zdrojový obrázek, název metody a dvě desetinné souřadnice.

Ve chvíli, kdy dostane interpolace tyto čtyři hodnoty, zavolá správnou metodu a předá jí zdrojový obrázek a souřadnice. Konkrétní metoda provede výpočet a vrátí transformaci výslednou barvu vyžadovanou na daných souřadnicích.

Výpočet se provádí vždy na nejbližším okolí požadovaného bodu. Jak velké „území“ okolo se bere v potaz, záleží jenom na vybrané metodě interpolace. Všem pixelům v daném okolí se přiřadí tzv. váha. Je to číslo, které udává, jak moc si budeme při výpočtu všímat právě daného pixelu a jeho

barvy. Například pro okolí 4x4 s výsledným bodem uprostřed, dostanou nejmenší váhu právě ty pixely, které jsou nejdále. Pixely, které jsou blíže k výslednému, dostanou váhu větší. Jaká váha bude a jak se bude lišit rozhoduje opět konkrétní vybraná interpolace.

3.2.2 Metoda nejbližšího souseda

Tato nejjednodušší metoda závisí na prostém zaokrouhlování. Souřadnice se zaokrouhlí, tím se zjistí nejbližší možný celočíselný sousední pixel, a jeho barva se pošle zpátky funkci, která o interpolaci zažádala.

3.2.3 Bilineární interpolace

Bilineární interpolace už není tak triviální jako její předchůdce. Využívá se již všech barev z okolních čtyř pixelů.

Podle teorie k bilineární interpolaci (viz kapitola 2.3.2) budeme postupovat podle jednotlivých dimenzí. Nejprve spočteme barvu pro řádky a později pro sloupce. Váhy pro jednotlivé pixely se budou počítat vždy pouze v jedné dimenzi. V této interpolaci využíváme lineární funkci. Váha daného pixelu se tedy spočte podle vzorce 8.

3.2.4 Bikubická interpolace

Bikubická interpolace je první zástupce nelineární metody. Jejím grafem již není přímka, jako v předchozím případě, nýbrž křivka definovaná polynomem třetího stupně.

Výpočet se provádí na šestnácti pixelech opět jednotlivě pro obě dimenze. Nejprve pro řádky a později pro sloupec mezivýsledků.

Spočtení váhy již není jednoduché. Celkem budeme v jedné dimenzi počítat váhu pro čtyři pixely. Vezmeme první z nich a zjistíme, jak daleko je od interpolovaného bodu. Tato vzdálenost by měla být od nuly do dvou, protože interpolovaný bod leží mezi dvěma prostředními pixely. Pokud vzdálenost padne mezi nulu a jedničku použijeme pro výpočet váhy vzorec 9. Analogicky pro vzdálenost od jedné do dvou použijeme vzorec 10.

V obou vzorcích je důležitá konstanta velké A, která určuje chování polynomu, a tím do značné míry ovlivňuje průběh interpolace. V mojí aplikaci je konstanta nastavena na hodnotu -0,75. To je stejná hodnota jako využívá pro svoji kubickou interpolaci Adobe Photoshop.

3.2.5 Spline interpolace

Interpolaci pomocí splinu (v literatuře někdy též „splajnu“) bych rád rozdělil na dvě samostatné metody. Obě fungují podobně, avšak obě pracují s jinak velkým okolím (tj. jiným počtem pixelů) a tudíž obě používají jiné vzorce pro výpočet váhy.

V kubické interpolaci jsme měli jeden polynom třetího stupně rozdělený mezi čtyři pixely. Spline vyžaduje, aby každému intervalu náležel vlastní polynom. Jedinou podmínkou je nulová derivace polynomů v krajních bodech intervalů.

3.2.5.1 Spline16

Postup a princip interpolace je téměř totožný s bikubickou interpolací. Okolí, ze kterého se počítá výsledná barva je také šestnáct pixelů. V jedné dimenzi jde tedy o čtyři pixely a tudíž dva intervaly vzdáleností. Od nuly do jedné a od jedné do dvou. Záporné hodnoty nepřipadají v úvahu, protože se jedná o vzdálenost. Vzorce pro výpočet váhy již nepočítají se žádnou volitelnou konstantou jako v předchozím případě.

3.2.5.2 Spline36

Tato metoda využívá pro výpočet pole pixelů o velikosti 6x6 (celkem 36 pixelů). To znamená, že máme šest bodů na jednu dimenzi. Vzdálenost je vždy kladná, takže tři intervaly vzdáleností od požadovaného bodu, který je vždy uprostřed. Každému intervalu náleží právě jeden polynom definovaný ve vzorcích 13.

3.2.6 Sinc interpolace

Interpolace pomocí funkce sinc je jednou z nejsložitějších interpolací vůbec. Počítá se na vzorku 16x16 pixelů (celkem 256). Naštěstí není potřeba mít pro každý interval vlastní polynom, jako je tomu u splinu. Využijeme zde tzv. Lanczosovi funkce popsané v kapitole o teorii (vzorec 14).

Výpočet váhy je ovšem i tak hodně složitý a především se počítá pro velmi velkou množinu pixelů. Sice jde o nejkvalitnější interpolaci vůbec, ale zároveň se jedná o interpolaci nejdelší. Časová složitost je rozhodně největší právě u této metody. Jaká přesně se dozvíme v kapitole výsledky.

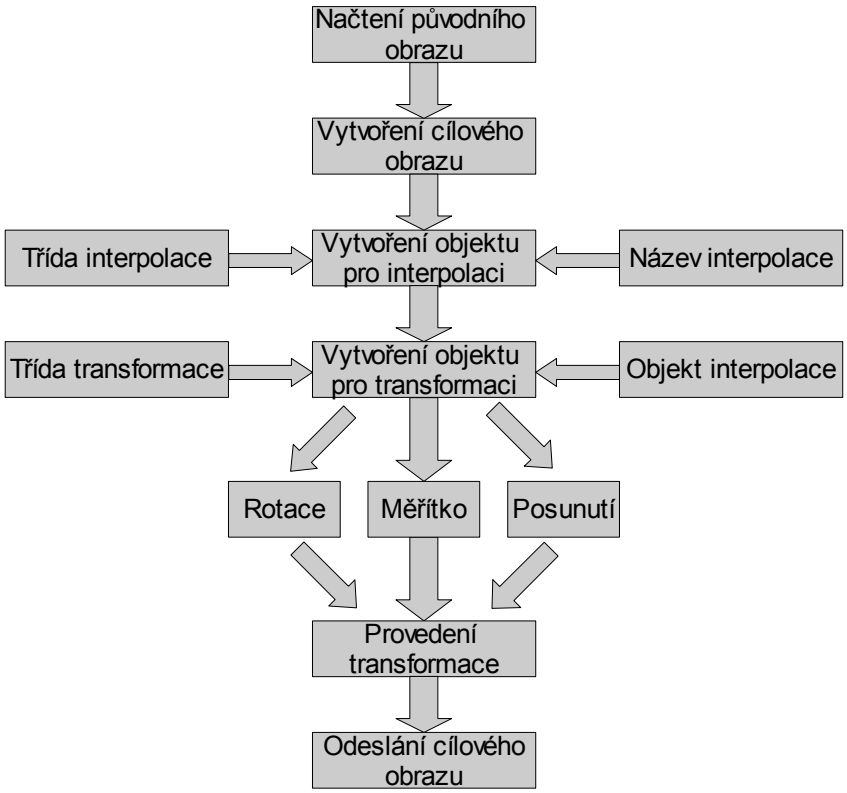
3.3 Návrh programu

Nyní bych rád nastínil, jak bude celá aplikace pracovat dohromady. Je zřejmé, že samotné knihovny, popsané výše, nebudou schopné samostatné činnosti. Je potřeba vytvořit nějaký jednoduchý program, který bude spouštět celý mechanizmus vedoucí od původního obrazu k finálnímu.

Po poradě s vedoucím práce, jsem se rozhodl použít již hotovou knihovnu s názvem MDSTk (Medical Data Segmentation Toolkit). S její pomocí bude mnohem snadnější vytvoření konečné aplikace. MDSTk využívá ke svojí práci tzv. modulů, což jsou malé programy obstarávající vždy konkrétní funkci. Obsahuje například modul pro načtení souboru, pro jeho převod na stupně šedi, pro uložení do souboru a mnohé další.

Mým úkolem bude vytvořit jeden takový modul, který dostane již připravená data od předchozího modulu, provede na nich příslušnou transformaci a předá je dalšímu modulu.

Schéma programu je zobrazeno na obrázku 9. Nejprve musí program načíst vstupní data (obrázek). Rovnou si uvolníme místo v paměti na cílový obraz. Vytvoříme instanci třídy interpolace podle zadaného názvu interpolace. Vytvoříme instanci třídy transformace, které předáme v konstruktoru objekt interpolace. Složíme do sebe všechny zadané transformace a výslednou operaci provedeme. Až ve chvíli provedení řekneme transformaci na jakých obrazech se má provádět. Odešleme finální obraz dalšímu modulu.



Obrázek 9: Návrh možného průběhu programu

4 Implementace

Aplikace je začleněna do prostředí grafického kitu MDSTk, kterého se notně využívá během implementace. Odpadají tím starosti s načítáním obrazu, jeho ukládáním, atd. Vzhledem k této začleněnosti bylo nutné přizpůsobit také implementační jazyk, kterým je C++.

Kvůli zvýšení abstrakce jsou všechny zdrojové texty rozdělené na hlavičkové soubory a implementační.

4.1 mdsSliceTransform

Jedná se o samotný modul, pomocí kterého se spouští všechny transformace. Modul očekává na vstupu datový kanál obsahující původní obraz. Pokud obdrží data ve správném formátu, vytvoříme „chytrý ukazatel“ (angl. smart pointer), který necháme na tyto vstupní data ukazovat. Dále nadeklarujeme dva další chytré ukazatele, jeden na výstupní obrázek a jeden pomocný. Od této chvíle se začne pracovat s knihovnami pro transformace a interpolace. Abychom mohli plně využít možností, které nám nabízí dědičnost implementovaná v knihovně interpolací, nadeklarujeme objekt bázové třídy CInterpolate a necháme ho prázdný. Pomocí switche rozhodneme, kterou metodu interpolace použijeme, a podle výsledku naplníme ukazatel konkrétní zděděnou třídou. Nyní vytvoříme, pomocí konstruktoru, objekt třídy CTransform, kterému předáme v parametru již vytvořený objekt interpolace. Objekt transformace obsahuje veřejné metody pro konkrétní operace s obrazem. Metody mají pouze nastavovací charakter. Pokud zavoláme metodu *rotate()*, pak se pouze nastaví transformační matice do požadovaného tvaru. Ve výsledku to znamená, že těmito metodami (rotate, scale, translate) nastavujeme složení výsledné transformace. Ta se sama o sobě vykoná až ve chvíli, kdy zavoláme metodu *execute()*, které teprve předáme vstupní a cílový obraz. Řešení má největší výhodu v tom, že pokud si nastavíme jednu konkrétní transformaci, vůbec nezáleží na jakém obrazu se bude provádět, a my ji tak můžeme provést na několika různých obrazech úplně stejně.

4.2 mdsTransform

Název mdsTransform označuje knihovnu obsahující třídu pro geometrické transformace CTransform. Třída obsahuje vlastní soukromou transformační matici, matici pro změnu velikosti obrazu a potom údaje o konkrétní velikosti. Veřejné metody pro nastavení transformace pracují právě s těmito globálně definovanými maticemi a proměnnými. Každá metoda upraví výslednou matici transformace a zároveň matici pro velikost obrazu. Hlavní metoda *execute()* provede výpočet na převzatých obrázcích pomocí globálních matic pro transformaci a pro velikost. V této metodě se také

provádí například posunutí osy transformace do středu nového obrázku. K chybám by docházelo především u rotace, kdy by se obrázek otáčel kolem prvního bodu. Cyklus pracuje s každým bodem nového obrazu. Nejprve pixel putuje do transformační matice, ta z něj udělá, v případě rotace, desetinné číslo a to se odešle jako parametr objektu interpolace. Z něj se vrátí už konkrétní barva, která náleží pixelu na souřadnicích platných před odesláním do transformační matice.

4.3 mdsInterpolate

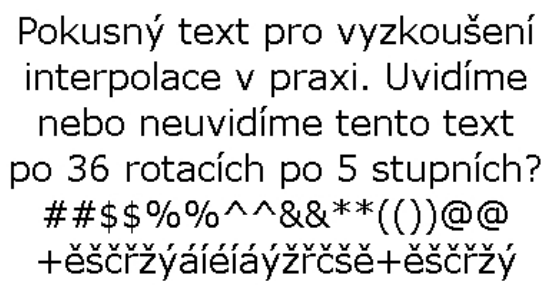
Poslední částí aplikace je knihovna mdsInterpolate, která v sobě ukrývá všechny interpolační metody. První třída Cinterpolate je rodičovskou třídou pro všechny ostatní metody. Sama o sobě obsahuje pouze virtuální metodu *interpolate()*, která zaručí správné použití konkrétní metody. Každá zděděná interpolační třída rovněž obsahuje tuto metodu. Některé složitější potom mají navíc metody na výpočet polynomů pro váhy okolních pixelů.

5 Výsledky

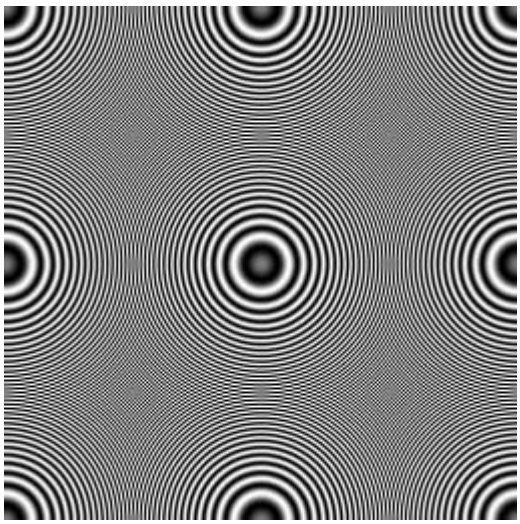
Před zhodnocením všech výsledků je třeba provést velkou sérii testů. všechny testy, které jsem prováděl jsou přiloženy na doprovodném CD-ROM. Do této textové části jsem vybral jen nejatraktivnější položky, abych nepřesáhl požadovanou délku práce.

5.1 Původní obrazy

Pro testy jsem vybral řadu rozličných obrázků, pokud možno tak, aby spolu neměly vůbec nic společného. Vybrané obrazy jsou všechny ve stupních šedi, aby byly jejich úpravy jednodušší a časově méně náročné.



Obrázek 11: Rastrový text



Obrázek 10: Vysokofrekvenční rastr[2]

Obrázek 12: Bitmapa [3]

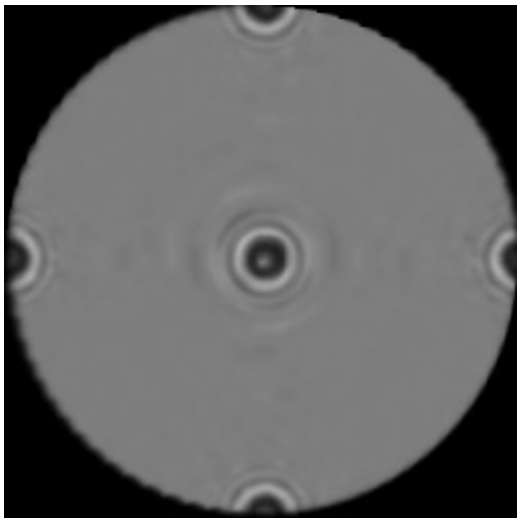
2 http://www.all-in-one.ee/~dersch/interpolator/interpolator.html

3 http://www.deviantart.com/

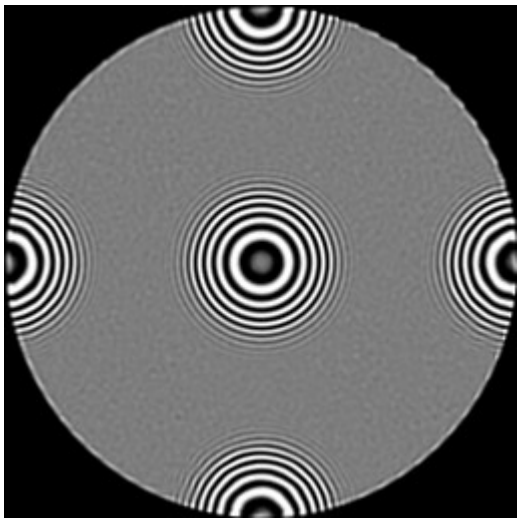
5.2 Testy

5.2.1 Rotace

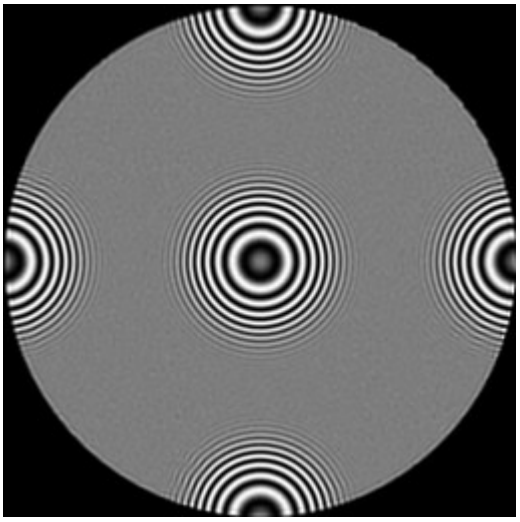
Tento test ukáže jak budou vypadat některé obrázky po provedení 36 rotací za sebou. Každá rotace bude o 5˚ takže celkem 180˚. Rozdíly mezi jednotlivými interpolacemi jsou opravdu značné.



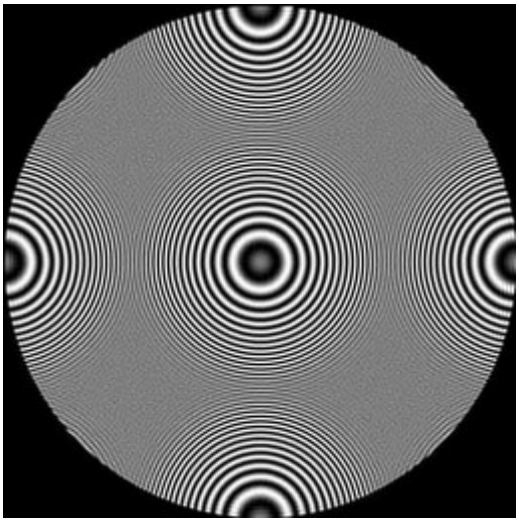
Obrázek 14: Bilineární interpolace



Obrázek 13: Bikubická interpolace



Obrázek 15: Interpolace spline36



Obrázek 16: Interpolace sinc

Obrázek 17: Nejbližší soused

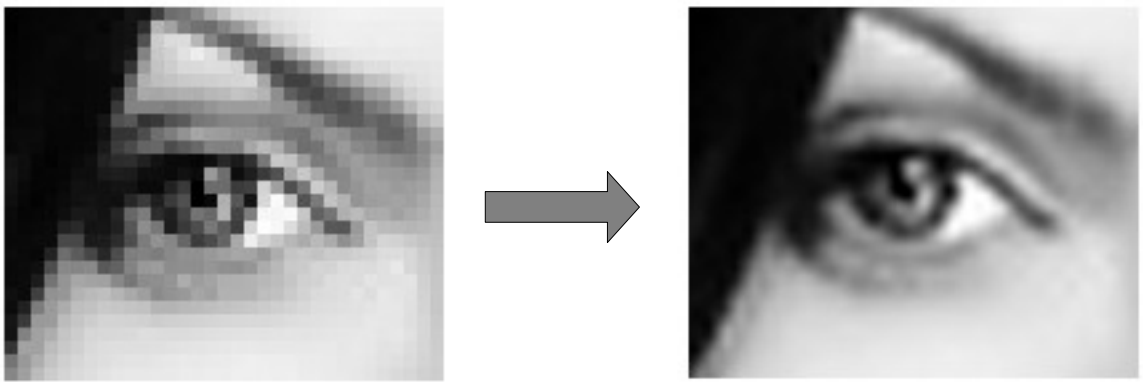
Obrázek 18: Bikubická interpolace

Obrázek 19: Interpolace spline36

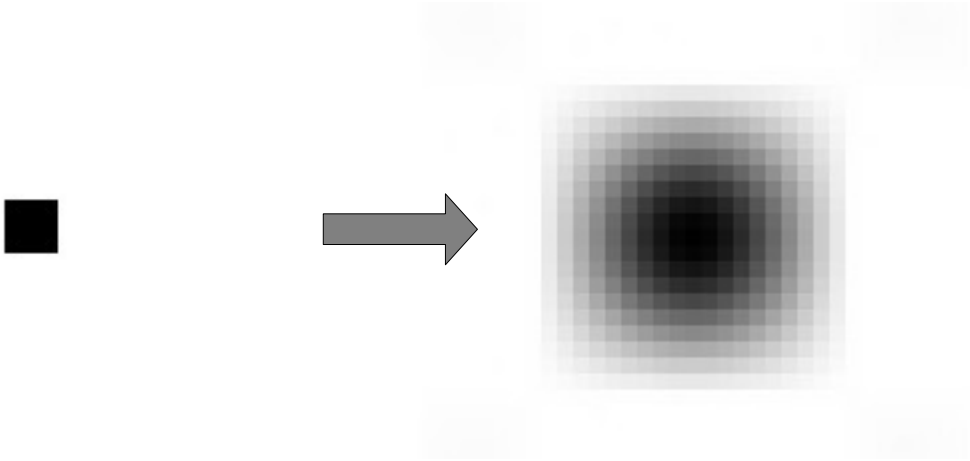
Obrázek 20: Interpolace sinc

5.2.2 Zvětšování

U zvětšování bych rád ukázal především detaily obrázků. Výřezy z bitmapy, takové, aby ukázaly co se v obraze ve skutečnosti děje.



Obrázek 21: Rozdíl mezi obyčejným zvětšením a bikubickou interpolací



Obrázek 22: Zvětšení jednoho pixelu pomocí bikubické interpolace

5.2.3 Časová složitost

Prvním testem bylo 36 rotací po pěti stupních. Druhým pak zvětšování obrazu na 800% jeho původní velikosti. Níže uvedené tabulky ukazují, jak všechny metody dopadly. Jistě bude vše záležet na rychlosti počítače, ale nás budou zajímat především rozdíly mezi jednotlivými výsledky.

Interpolace	Čas [s]
Nejbližší soused	2
Bilineární	2
Bikubická	2
Spline16	2
Spline36	3
Sinc	16

Tabulka 1: Test 1

Interpolace	Čas [s]
Nejbližší soused	5
Bilineární	7
Bikubická	10
Spline16	9
Spline36	18
Sinc	125

Tabulka 2: Test 2

Jak je vidět z tabulek, tak čím složitější metoda je, tím déle trvá její výpočet. Nejvíce záleží na velikosti okolí počítaného pixelu. V obou testech trvá nejdéle právě funkce sinc, která má okolí největší.

6 Závěr

V závěrečné kapitole bych rád uvedl shrnutí dosažených poznatků. Již při získávání potřebných informací pro obhajobu zadání bakalářské práce při semestrálním projektu mi bylo jasné, že nejzajímavějším hlediskem nebudou transformace samotné, ale že to budou právě různé druhy interpolačních metod.

Jednotlivé interpolační funkce mají úplně rozdílný charakter a všechny bezpochyby najdou své uplatnění v praxi. Pokud potřebujeme, například v grafickém editoru, přiblížit a změnit obraz na jednotlivých pixelech, musíme si vybrat metodu nejbližšího souseda, protože ostatní interpolace by jinak neustále pixely vyhlazovaly a my bychom nic neviděli. Složitější interpolace dostanou prostor ve chvíli, kdy nám naopak záleží na kvalitě provedené operace. Existuje mnoho případů, kdy je potřeba provést transformaci pokud možno co nejpřesněji. Jedním z nich jsou například lékařské snímky.

Celkově nejvýhodnější se ukázala interpolace pomocí splinu na 36 pixelech. Podle dosažených výsledků je vidět, že je dostatečně přesná a zároveň překvapivě rychlá. O svůj post se mohla podělit pouze s bikubickou interpolací, která ovšem nedosahuje tak kvalitních výsledných obrazů v poměru k výpočetnímu času.

Z hlediska dalšího vývoje projektu je možné aplikaci obohatit o další druhy transformací počínaje různými zkoseními, zrcadleními, konče různými nelineárními transformacemi jako jsou třeba warpping atd. V kategorii interpolací je další vývoj nepravděpodobný, protože všechny známé a nejčastěji používané metody jsou v této práci popsány a implementovány.

Literatura

[1] Žára, J., Beneš, B., Sochor, J., Felkel, P. *Moderní počítačová grafika*. Brno 2004.

[2] Fajmon, B., Růžičková, I. *Matematika 3*. FEKT VUT Brno 2005

[3] Kršek, P., *Základy počítačové grafiky - Studijní opora*. FIT VUT Brno v0.9

[4] http://cs.wikipedia.org

[5] http://www.all-in-one.ee/~dersch/interpolator/interpolator.html

Seznam příloh

Příloha A. Plakát reprezentující práci.

Příloha B. CD-ROM (zdrojové texty, dokumentace, testy, tato práce v elektronické podobě).