Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Aplikace Radiálních bázových funkcí

Plzeň, 2007

Jiří Zapletal

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne _____

podpis

Poděkování

Na tomto místě bych chtěl poděkovat svému školiteli Prof. Ing. Václavu Skalovi, CSc. za jeho úsilí a podněty, kterými mě zahrnoval, a také Ing. Petrovi Vaněčkovi, Ph.D. za jeho cenné rady, návrhy a připomínky.
V neposlední řadě patří mé poděkování především rodině, která mi vždy vycházela vstříc a všemi způsoby mě podporovala. Dále přátelům, kolegům ale i všem ostatním, kteří mě jakkoliv inspirovali či motivovali v mém úsilí. Bez jejich přispění by tato práce nikdy nevznikla.

Abstract

Application of Radial basis functions

This diploma thesis is concerned with application of radial basis function interpolation. This type of interpolation has become very popular nowadays especially because of its ability to interpolate scattered or non-uniformly sampled data. In addition it could either interpolate or approximate this data and one can achieve very interesting results even in very complex cases where other methods fail.
This thesis is mainly focused on the reconstruction of damaged images while the stress is put on the finding of method giving the best possible visual results. We will propose a new algorithm and compare it with other existing methods (especially with method proposed by Ing. Karel Uhlíř). Investigated methods were tested on non-trivial real data damaged by various defect characters.
The implementation was written in C# language under MVE-2 environment.

Abstrakt

Tato diplomová práce se zabývá aplikací interpolace pomocí radiálních bázových funkcí. Tento způsob interpolace se poslední dobou stává velice oblíbeným, zvláště kvůli své schopnosti interpolovat rozptýlená či nerovnoměrně nasnímaná data. Navíc data umí nejen interpolovat, ale také aproximovat a lze dosáhnout velice zajímavých výsledků dokonce i ve velmi složitých případech, kde ostatní metody selhávají.
Práce je zaměřena především na rekonstrukci poškozených obrazů, přičemž důraz je kladen na nalezení metody dávající co nejkvalitnější vizuální výsledky. Představíme si nový algoritmus a srovnáme ho s jinými existujícími metodami (především s metodou Ing. Karla Uhlíře). Zkoumané metody byly otestovány na netriviálních reálných datech, poškozených defekty různého charakteru.
Implementace byla provedena v jazyce C# do prostředí MVE-2.

Práce podporována projektem:
VIRTUÁLNÍ VĚDECKO-PEDAGOGICKÉ CENTRUM POČÍTAČOVÉ GRAFIKY A VIZUALIZACE DAT, projekt MSMT CR 2C 06002

Použité nástroje:
SSIM index measurements were done with MSU Video Quality Measurement Tool.

Obsah

Značení a zkratky

$\boldsymbol{A}$	…matice (Times New Roman, velké písmeno, tučně, kurzíva)
$\boldsymbol{b}$	…vektor (Times New Roman, malé písmeno, tučně, kurzíva)
x_1	…složka vektoru či bodu (Times New Roman, malé písmeno, kurzíva)
x	…bod, konstanta, proměnná, důraz (Times New Roman, malé písmeno, kurzíva)
$\|\cdot\|$	…euklidovská norma (vzdálenost)
`Foo()`	…zápis v pseudokódu (Courier New, tučně)

CAD	… Computer Aided Design
CSG	… Constructive Solid Geometry
CSRBF	… Compactly Supported RBF
FMM	… Fast Multipole Method
GMRES	… Generalized Minimal Residual
HDR	… High Dynamic Range
MSE	… Mean Square Error
POU	… Partition Of Unity
PSNR	… Peak Signal to Noise Ratio
RBF	… Radiální Bázové Funkce
SSIM	… Structural SIMilarity index
SVD	… Singular Value Decomposition
TPS	… Thin-Plate Spline

1. Úvod

V této úvodní kapitole si nejprve ukážeme motivační využití interpolace roztroušených dat a přehled nejpoužívanějších metod. Informace jsou čerpány především z [Amid02].

Problém interpolace roztroušených dat znamená vytvoření spojité funkce jedné, dvou, tří či více nezávislých proměnných, která interpoluje data na základě hodnot, známých pouze v několika rozptýlených (tzn. nerovnoměrně rozložených) bodech v rovině či v prostoru. Potřeba interpolace takových dat se poslední dobou – zejména v souvislosti s rozvojem snímací a výpočetní techniky – uplatňuje v mnoha vědních oborech, např. v lékařství (3D scanner, CT či MRI data), meteorologii, geologii, oceánografii, kartografii, CAD systémech a dalších. S vizualizací těchto dat úzce souvisí právě problém jejich interpolace, tedy proložení hladkého povrchu získanými daty. Interpolací tedy hledáme funkci, která prochází všemi body ze vstupní množiny dat. Po nalezení této funkce lze již vypočítat hodnotu v libovolném bodě, tedy i v takovém, jehož hodnotu jsme z měření nezískali.
Např. v meteorologii se hodnoty pro měření počasí získávají z nerovnoměrně rozmístěných meteorologických stanic, v geologii se struktury podloží či obsah vody v oblasti zjišťují z dat, přístupných pouze z několika lokací. Data v těchto několika oblastech musí být interpolována pro umožnění 2D či 3D zobrazení specifickými nástroji k dalším účelům.
Uvedené příklady hledanou funkci převádí $R^2 \to R$ či $R^3 \to R$, může však také nastat situace, kdy je potřeba převodu $R^2 \to R^2$ či $R^3 \to R^3$. Jako příklad k $R^2 \to R^2$ může posloužit potřeba geometrických úprav satelitních snímků, zkreslených díky zakřivení zemského povrchu a úhlu pod kterým byly snímány. V tomto případě vstupní data obsahují identifikační kontrolní body na povrchu (spojnice silnic). Pro každý takový kontrolní bod pak známe nejen jeho zkreslené souřadnice ze snímků, ale i jeho skutečné souřadnice z geografických map. Naším úkolem je tedy zřejmě interpolace mezi těmito několika známými body, abychom získali opravnou geometrickou transformaci $R^2 \to R^2$. Podobné případy nastávají i v zobrazování lékařských dat, kdy je potřeba porovnat snímky pacienta získávané průběžně během léčby či různými snímacími technikami nebo je potřeba jejich porovnání se standardními anatomickými hodnotami
Jiný příklad, kdy hledáme funkci $R^3 \to R^3$, představuje kalibraci barev vstupních či výstupních zařízení (scannerů, tiskáren) vzhledem k barevným prostorům (např. XYZ či L*a*b*). V takových případech je kalibrace zařízení založena na zjištění mapování mezi 3D (na vstupním zařízení závislém) RGB prostoru a zvoleném cílovém 3D prostoru (např. XYZ). To je provedeno pomocí katalogu barev – každý barevný vzorek je předán vstupnímu zařízení k získání RGB hodnot odpovídajícím danému zařízení a poté změřen spektrometrem k získání jeho XYZ hodnot. Tím získáme množinu bodů, které jsou rozptýlené ve vstupním RGB 3D prostoru a ke každému jsou přiřazeny tři hodnoty – souřadnice vzorku v XYZ prostoru. V tomto případě je hodnota v každém z rozptýlených bodů 3D veličina, takže musíme řešit tři případy problému hledání funkce jedné proměnné – jednou pro každou XYZ složku.

Problém interpolace rozptýlených dat je i přes množství způsobů jeho řešení stále obtížný a výpočetně náročný. V této práci se budeme zabývat metodou, která se v posledních letech stala středem zájmu mnoha pracovišť – interpolace pomocí radiálních bázových funkcí. Hledaná interpolační funkce je v tomto případě lineární kombinací radiálně symetrických bázových funkcí, kde neznámé hodnoty získáme řešením soustavy (lineárních) rovnic.

1.1. Vybrané metody interpolace roztroušených dat

Zde uvádíme přehled nejčastěji používaných metod interpolace rozptýlených dat včetně základního popisu principu, na kterém fungují.

1.1.1. Metody založené na triangulaci či tetrahedronizaci dat

Metody patřící do této kategorie pracují ve dvou krocích. Nejprve je množina rozptýlených bodů triangulována (ve 2D) či tetrahedronizována (ve 3D) a poté je interpolace použita pro každý trojúhelník či tetrahedron. Z toho plyne, že jsou tyto metody vždy lokální.

1.1.2. Metody vážené inverzní vzdálenosti

Jedná se o jednu z nejběžnějších technik interpolace rozptýlených dat. Metody vážené inverzní vzdálenosti jsou také známy jako Shepardovy metody – podle zakladatele metod fungujících na tomto principu. Tyto techniky jsou globální, tzn. že využívají všechny body ze vstupní množiny dat k výpočtu každé interpolované hodnoty.
Jejich základním předpokladem je, že interpolované hodnoty budou více ovlivněny blízkými body a méně těmi vzdálenými. Interpolovaná hodnota v každém novém bodě P je váženým průměrem hodnot všech rozptýlených bodů a váha přiřazená každému rozptýlenému bodu klesá se vzdáleností od právě interpolovaného bodu P.
Máme-li množinu nerovnoměrně rozložených bodů P_i v prostoru, jejichž souřadnice a hodnota jsou (x_i, y_i) a z_i, hledáme interpolující funkci $f(x_i, y_i) = z_i$ pro všechna i, kde vliv bodu P_i klesá se zvyšující se vzdáleností mezi (x_i, y_i) a (x, y). Shepardův postup lze zapsat jako vážený průměr hodnot z_i:

$$f(x,y) = \sum_{i=1}^{n} w_i(x,y)\, z_i = \sum_{i=1}^{n} \frac{h_i(x,y)}{\sum_{i=1}^{n} h_i(x,y)}\, z_i$$

s vahami

$$w_i(x,y) = \frac{h_i(x,y)}{\sum_{i=1}^{n} h_i(x,y)} \qquad 0 \le w_i(x,y) \le 1 \qquad \sum_{i=1}^{n} w_i(x,y) = 1,$$

kde

$$h_i(x,y) = \frac{1}{\|x-y\|^k} \qquad k \ge 1 \quad \text{je zvolená hodnota exponentu}$$

1.1.3. Metody radiálních bázových funkcí

Jedná se opět o globální interpolační metody. Jako první s nimi přišel Hardy koncem 60. let 20. století. Metody lze charakterizovat následujícím způsobem:

Pro každý bod (x_i, y_i) vstupní množiny dat zvolíme funkci $\phi_i(x, y)$ a spočítáme koeficienty λ_i, takže $f(x, y) = \sum_{i=1}^{n} \lambda_i \phi_i(x, y)$ interpoluje data. Vhodná volba funkce $\phi_i(x, y)$ není snadná. Například polynomiální funkce dávají velmi špatně interpolované povrchy s množstvím přesahů a zvlnění mezi zadanými body. Dokonce i funkce, o kterých je známo, že pracují dobře, nelze vždy snadno matematicky ověřit. Nejlepší funkce $\phi_i(x, y)$ jsou tzv. radiální funkce jedné proměnné $\phi(r)$ takové, že bázová funkce přiřazená každému datovému bodu (x_i, y_i) má tvar $\phi_i(x, y) = \phi(d_i)$, kde d_i je vzdálenost mezi (x, y) a (x_i, y_i). Odtud název Radiální Bázové Funkce (RBF).
Tato diplomová práce se zabývá právě touto interpolační metodou a jejím využitím pro rekonstrukci poškozených obrazů a hledání povrchů ze zadaných bodů.

1.1.4. Metody interpolace přirozeným sousedem

Tato technika je lokální a vychází z Voronoiova diagramu zadané množiny rozptýlených bodů, která je geometrickým duálem k Delaunayově triangulaci / tetrahedronizaci. Voronoiův diagram rozdělí plochu či prostor na úplnou disjunktní množinu buněk, kdy každá buňka T_i obsahuje právě jeden bod x_i z dané množiny bodů. Buňka T_i je definovaná jako plocha (či objem), která je nejblíže k bodu x_i ze všech ostatních bodů množiny rozptýlených dat. Matematický zápis vypadá takto:

$$T_i = \left\{ x \in R^2, \|x_i - x\| \leq \|x_j - x\|, \forall j = 1, \ldots, n \right\},$$

kde $\|x - y\|$ znamená euklidovskou vzdálenost mezi body x a y. Bod x_i z naší množiny bodů je tzv. přirozený soused bodu x_j tehdy, když jejich buňky mají společnou hranu nebo vrchol. Počet přirozených sousedů bodu x_i (který není na okraji konvexního obalu celé množiny) je nejméně $N+1$, kde N je dimenze prostoru a maximálně $n-1$, kde n je počet bodů množiny. Tento počet přirozených sousedů se liší bod od bodu. Ve 2D případě průměrný počet přirozených sousedů (počet hran buňky) nepřevyšuje 6.
Před začátkem interpolace je potřeba provést preprocessing v podobě vytvoření Voronoiova diagramu zadané množiny rozptýlených bodů. Jakmile je Voronoiův diagram vytvořen, tak k výpočtu interpolované hodnoty v novém bodě x potřebujeme přidat tento nový bod x do množiny rozptýlených bodů a vytvořit novou buňku z ploch (prostorů) okolních buněk.

2. Teoretická část

V této kapitole se budeme zabývat historií vzniku RBF metody, dále uvedeme potřebný matematický aparát, také představíme často používané bázové funkce a nakonec popíšeme vybrané matematické metody řešení problému a možnosti jak je urychlit. Čerpáno bylo převážně z [Uhlir05] a [Wright03].

2.1. Úvod

Na úvod rovnou představíme hlavní výhody, proč při výběru interpolační metody zvolit právě radiální bázové funkce (dále jen RBF) [Zhang02]:

- kompaktní popis objektu jednou funkcí
- schopnost interpolovat i aproximovat data
- schopnost interpolovat řídká, nerovnoměrně rozložená data
- možnost výpočtu kdekoliv na povrchu pro výpočet povrchového modelu v požadovaném rozlišení
- isoplochy získané pomocí RBF jsou manifold (tzn. neprotínají se)

Dále pro představu nezasvěceného čtenáře uvedeme i příklady praktického využití RBF interpolace:

- rekonstrukce povrchů
- vyplňování děr (např. vytváření implantátů a protéz v lékařství)
- rekonstrukce poškozených obrazů
- vyhlazování povrchů
- animace (interpolace mezi dvěma snímky či modely)
- generování modelů
- snižování počtu polygonů modelů

2.2. Historie a základní matematický aparát

Základem interpolační metody pomocí RBF je použití posunující se bázové funkce $\phi(r)$. V nejobecnějším případě je tato funkce závislá na euklidovské vzdálenosti od svého středu. Tato funkce je radiálně symetrická kolem svého středu, odtud její název *radiální bázová funkce* a název metody je *RBF metoda*. Jejich objevitelem se stal L.R. Hardy [Hardy71], jenž je i pojmenoval.

Hardy se zabýval kartografickým problémem, kdy se pokoušel o automatizaci generování vrstevnicových map nalezením spojité funkce, interpolující naměřená data. Tento postup se do té doby musel dělat ručně na základě zkušeností topografa, který tvar terénu z naměřených dat pouze odhadoval. Tento postup mohl evidentně vést ze stejných dat k různým výsledkům od různých topografů. Zpočátku se zdálo, že se využijí Fourierovy řady, ty však pro interpolaci vykazovaly sklony ke kmitání výstupních hodnot mezi vzorovými daty. Použití metody nejmenších čtverců nebylo také úspěšné, neboť výsledky neodpovídaly dostatečně přesně realitě (tato metoda totiž neinterpoluje, ale aproximuje, tzn. výsledek nemusí procházet naměřenými vstupními daty). Hardy nebyl spokojen se žádnou známou metodou, rozhodl se tedy hledat novou. Zjistil, že tvar křivky lze zapsat jako po částech lineární interpolační funkci. Pro množinu n vstupních zdrojových bodů $\{x_i\}_{i=1}^{n}$ a jim odpovídajícím naměřeným hodnotám $\{f_i\}_{i=1}^{n}$ odvodil následující interpolační vztah:

$$s(x) = \sum_{i=1}^{n} \lambda_i \cdot |x - x_i|, \qquad \textbf{(2.2.1)}$$

kdy λ_i určíme rozkladem a $s(x_i) = f_i$ pro $i = 1, 2, ..., n$. Geometricky vzato, jsou data interpolována lineární kombinací n posunutí absolutní hodnoty bázové funkce $|x|$ a vrchol každé bázové funkce je středem některého vstupního bodu.
Nevýhodou se ukázal být skokově se měnící průběh této funkce, bázová funkce byla tedy nahrazena absolutní hodnotou spojitě diferencovatelné funkce $\sqrt{c^2 + x^2}$, kde c je nenulová konstanta. Nyní interpolační vztah vypadal

$$s(x) = \sum_{i=1}^{n} \lambda_i \cdot \sqrt{c^2 + (x - x_i)^2}\ . \qquad \textbf{(2.2.2)}$$

Jeho výhodou bylo také použití i ve vyšší dimenzi (2D) – pro množinu n vstupních zdrojových bodů $\{(x_i, y_i)\}_{i=1}^{n}$ a odpovídajícím měřením $\{f_i\}_{i=1}^{n}$ se funkce vypočítá podle vztahu

$$s(x, y) = \sum_{i=1}^{n} \lambda_i \cdot \sqrt{c^2 + (x - x_i)^2 + (y - y_i)^2}\ . \qquad \textbf{(2.2.3)}$$

Geometricky funkce odpovídá interpolaci dat lineární kombinací n posunutí kužele a tedy radiálně symetrickou funkcí $\phi(r) = r$, kde $r = \sqrt{x^2 + y^2}$. Vrchol každého kužele byl v jednom ze vstupních bodů. Funkce je však aproximační.
Tato Hardyho metoda se označuje jako tzv. *multiquadric* metoda. Je schopná interpolovat ve 2D, ale lze ji rozšířit i do vyšších dimenzí.

Definice:
Pro množinu n nezávislých zdrojových bodů $\{\underline{x}_i\}_{i=1}^{n}$ v R^d ($\underline{x}_i = (x_1^{(i)}, x_2^{(i)}, ..., x_d^{(i)})$) a jim odpovídajícím naměřeným skalárním hodnotám $\{f_i\}_{i=1}^{n}$ je multiquadric interpolant definován jako:

$$s(\underline{x}) = \sum_{i=1}^{n} \lambda_i \cdot \sqrt{c^2 + \|\underline{x} - \underline{x}_i\|^2}\ , \qquad \textbf{(2.2.4)}$$

kde $\|\cdot\|$ je euklidovská norma. Koeficienty λ_i jsou určeny z interpolační podmínky $s(\underline{x}_i) = f_i$ pro $i = 1, 2, ..., n$. Maticově lze celý lineární systém zapsat následovně:

$$\begin{bmatrix} & \boldsymbol{A} & \end{bmatrix} \begin{bmatrix} \\ \boldsymbol{\lambda} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \boldsymbol{f} \\ \\ \end{bmatrix}, \qquad \textbf{(2.2.5)}$$

kde $\boldsymbol{A} = (a_{i,j}) = \sqrt{c^2 + \|\underline{x}_i - \underline{x}_j\|^2}$.

Metoda byla záhy po zveřejnění využita např. i v hydrologii, geodesii, fotometrii a geologii. Lze ji použít i s jinou bázovou funkcí, stále je však potřeba dodržet závislost funkce na euklidovské vzdálenosti a vždy platí značení $r = \|\underline{x}\|$.

Oblíbenou funkcí, kterou poprvé použil Duchon [Duchon77] se stala $r^2 \log r$ (ve 2D, nazývaná jako *thin-plate spline – TPS*) a r^3 (ve 3D). TPS funkce má následující fyzikální význam: reprezentuje tenkou kovovou desku, která je přinucena změnit svůj rovinný tvar tak, aby se identické body na desce ztotožnily. Jde tedy o nalezení funkce $f(x, y)$, která přesně ztotožní identické body a zároveň minimalizuje velikosti deformačních sil v kovovém plátu. Jinými slovy – pro zadanou vstupní množinu bodů vytvoří co nejméně zakřivený hladký povrch, procházející všemi zadanými body (př.: ve 3D není TPS pro méně než 3 body definována, pro 3 body získáme rovinu a pro více než 3 body zakřivený povrch). Metoda je často používána pro vizuální výsledky a stabilitu pro objemná data. Jinou možností je například použití Gaussovy funkce $e^{-(\varepsilon r)^2}$, kde ε je zvolený parametr [Schagen79].
Trvalo téměř 15 let, než bylo dokázáno, že multiquadric metoda je vždy nesingulární [Micchelli86] a že Hardy nalezl pouze jeden konkrétní případ obecnější metody. Obecná metoda spočívá v použití posunující se bázové funkce $\phi(r)$, závislé pouze na euklidovské vzdálenosti od svého středu. Takto závislá funkce je symetrická kolem svého středu (radiální). Bázovým funkcím je dále věnována samostatná kapitola (**2.5 Bázové funkce**).

2.3. Základní RBF metoda

Definice:

Je dána množina *n* od sebe různých bodů $\{x_i\}_{i=1}^n$ a jim odpovídající hodnoty $\{h_i\}_{i=1}^n$. Interpolační funkce je pak je dána vztahem

$$f(x) = \sum_{i=1}^{n} \lambda_i \cdot \phi\left(\|x - x_i\|\right), \qquad \textbf{(2.3.1)}$$

kde $\phi\left(\|\mathbf{x} - \mathbf{x}_i\|\right)$ je radiální funkce, zkráceně $\phi(r)$ pro $r \geq 0$. Koeficienty λ_i jsou určeny z interpolačních podmínek $f(x_i) = h_i$ pro $i = 1, ..., n$. Maticový zápis vypadá následovně:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix}, \qquad \textbf{(2.3.2)}$$

zkráceně

$$\begin{bmatrix} \\ \boldsymbol{A} \\ \\ \end{bmatrix} \begin{bmatrix} \\ \boldsymbol{\lambda} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \boldsymbol{h} \\ \\ \end{bmatrix}, \qquad \textbf{(2.3.3)}$$

kde $a_{i,j} = \phi\left(\|\mathbf{x}_i - \mathbf{x}_j\|\right), j = 1, ..., n$.

Definice:
Čtvercová matice $\boldsymbol{A}$ $(n \times n)$ se nazývá regulární, právě když $det\, \boldsymbol{A} \neq 0$ (tzn. matice $\boldsymbol{A}$ má hodnost n).

Definice (Wright [Wright03]):
Řekneme, že funkce ψ je ryze monotónní na intervalu $<0,\infty)$ pokud platí, že

1. $\psi \in C<0;\infty)$
2. $\psi \in C^{\infty}<0;\infty)$
3. $(-1)^i \psi^{(i)}(r) \geq 0$ pro $r > 0$ a $i = 0,1,2,...$

Věta (Schoenberg [Schoen38]) :
Jestliže $\psi(r) = \phi(\sqrt{r})$ je ryze monotónní, avšak ne konstantní na $<0;\infty)$, pak pro libovolnou množinu n rozdílných bodů $\{x_i\}_{i=1}^{n}$ je matice $\boldsymbol{A}$ $(n \times n)$ pozitivně definitní a tedy regulární.

Z této věty lze odvodit, že matice $\boldsymbol{A}$ je regulární např. pro bázové funkce:

$$\phi(r) = e^{-(\varepsilon r)^2} \qquad \textit{Gaussova funkce}$$

$$\phi(r) = \frac{1}{1+(\varepsilon r)^2} \qquad \textit{inverzní kvadratická funkce}$$

$$\phi(r) = \frac{1}{\sqrt{1+(\varepsilon r)^2}} \qquad \textit{inverzní multiquadric}$$

Pro klasickou multiquadric funkci je však podmínka nedostačující, předchozí věta byla tedy rozšířena:

Věta (Micchelli [Micchelli86]) :
Nechť $\psi(r) = \phi(\sqrt{r}) \in C^0 <0;\infty), \psi(r) > 0$ pro $r > 0$ a $\psi'(r)$ je ryze monotónní na $(0;\infty)$. Pro libovolnou množinu n rozdílných bodů $\{x_i\}_{i=1}^{n}$ je matice $\boldsymbol{A}$ $(n \times n)$ regulární. Kromě toho, pro $n \geq 2$ má matice $n-1$ negativních vlastních čísel a jedno pozitivní.

Tuto větu již splňuje i multiquadric funkce a s její pomocí vytvořená matice $\boldsymbol{A}$ je regulární a tento systém má tedy jednoznačné řešení. Tato věta ovšem stále neplatí např. pro TPS funkci, musela být tedy upravena a rozšířena o další podmínky, které budou popsány dále.

2.4. Rozšířená RBF metoda

Pro některé bázové funkce (Gaussovu, inverzní multiquadric, TPS) je potřeba k získání jednoznačného řešení přidat dodatečné podmínky a základní metodu rozšířit.

Definice:
Nechť $\coprod_m(R^d)$ je prostor všech polynomů d-proměnných které mají stupeň menší nebo roven m. Kromě toho nechť M udává dimenzi $\coprod_m(R^d)$. Potom $M = \binom{m+d}{m}$.

Definice [Wright03, Micchelli86]:
Je dána množina n od sebe různých bodů $\{x_i\}_{i=1}^n$ a jim odpovídajících hodnot $\{h_i\}_{i=1}^n$. Rozšířená interpolační funkce je pak dána vztahem

$$f(x) = \sum_{i=1}^{n} \lambda_i \cdot \phi\left(\|x - x_i\|\right) + \sum_{j=1}^{M} \gamma_j p_j(x),\ x \in R^d, \qquad \textbf{(2.4.1)}$$

kde $\{p_j(x)\}_{j=1}^M$ je báze $\coprod_m(R^d)$ a $\phi(r), r \geq 0$ je radiální funkce. Aby bylo možné použít polynomiální výraz, musí být doplněna omezující podmínka

$$\sum_{i=1}^{n} \lambda_i p_j(x_i) = 0,\ j = 1, 2, ..., M\ . \qquad \textbf{(2.4.2)}$$

Koeficienty λ_i a γ_j jsou určeny z interpolačních podmínek, které vedou k následujícímu symetrickému lineárnímu systému

$$\left[\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{P} \\ \hline \boldsymbol{P}^T & \boldsymbol{0} \end{array}\right]\left[\begin{array}{c} \boldsymbol{\lambda} \\ \hline \boldsymbol{\gamma} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{h} \\ \hline \boldsymbol{0} \end{array}\right]. \qquad \textbf{(2.4.3)}$$

$\boldsymbol{A}$ je stejná jako v základní RBF metodě (vztah **2.3.3**) a $\boldsymbol{P}$ je $n \times M$ matice $p_j(x_i)$ pro $i = 1,...,n$ a $j = 1,...,M$.

Věta (Micchelli [Micchelli86]):
Nechť $\psi(r) = \phi(\sqrt{r}) \in C^0 < 0;\infty), \psi^{m+1}(r)$ je ryze monotónní, ne však konstantní na $(0;\infty)$ pro $m \geq 0$. Pak pro libovolnou množinu n rozdílných bodů $\{x_i\}_{i=1}^n$ splňuje podmínku hodnosti matice $\boldsymbol{P}$, $hod(\boldsymbol{P}) = M$, kde $\boldsymbol{P}$ je $n \times M$ matice. Plná matice $(n+M) \times (n+M)$ je regulární. Mimoto je $\overline{m}$ nejmenší m takové, že $\psi^{m+1}(r)$ je ryze monotónní, potom pro libovolný nenulový vektor $\boldsymbol{\alpha} \in R^n$ splňuje podmínku $\boldsymbol{P^T\alpha = 0}$ a platí následující relace: $(-1)^{\overline{m}+1} \boldsymbol{\alpha^T A \alpha > 0}$, kde $\boldsymbol{A}$ je submatice řádu n.

Teprve tato věta zajišťuje podmínky pro regulárnost rozšířeného lineárního systému. V základní RBF metodě (vztah **2.3.3**) mohlo dojít k situaci, kdy byla matice $\boldsymbol{A}$ singulární. V rozšířené metodě nás však od nejednoznačného řešení ochrání přidání rozšiřujícího elementu (konstanta či mnohočlen), který bude interpolovat data i v případě, kdy je matice $\boldsymbol{A}$ singulární.
Nyní máme zajištěno, že RBF metoda bude mít řešení nejen ve své základní formě pro některé bázové funkce, ale i v rozšířené formě (vztah **2.4.3**) pro ty ostatní (splňující všechny 3 výše uvedené věty).

2.4.1. Lineární polynom

Pokud pro interpolaci rozšířenou metodou použijeme jako rozšiřující element lineární polynom, dostáváme pro 2D případ následující interpolant:

$$f(x,y)=\sum_{i=1}^{n}\lambda_i\phi\left(\sqrt{(x-x_i)^2+(y-y_i)^2}\right)+\gamma_0+\gamma_1 x+\gamma_2 y\,. \qquad \textbf{(2.4.1.1)}$$

Lineární systém pak tedy vypadá (pro 2D interpolant):

$$\left[\begin{array}{cccc|ccc} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} & x_1 & y_1 & 1 \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2n} & x_2 & y_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & 1 \\ \phi_{n1} & \phi_{n2} & \cdots & \phi_{nn} & x_n & y_n & 1 \\ \hline x_1 & x_2 & \cdots & x_n & 0 & 0 & 0 \\ y_1 & y_2 & \cdots & y_n & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c}\lambda_1\\ \lambda_2\\ \vdots\\ \lambda_n\\ \hline \gamma_1\\ \gamma_2\\ \gamma_0\end{array}\right]=\left[\begin{array}{c}h_1\\ h_2\\ \vdots\\ h_n\\ \hline 0\\ 0\\ 0\end{array}\right] \qquad \textbf{(2.4.1.2)}$$

zkráceně

$$\boldsymbol{B}\left[\begin{array}{c}\boldsymbol{\lambda}\\ \hline \boldsymbol{\gamma}\end{array}\right]=\left[\begin{array}{c}\boldsymbol{h}\\ \hline \boldsymbol{0}\end{array}\right]. \qquad \textbf{(2.4.1.3)}$$

2.4.2. Kvadratický polynom

V případě, že jako rozšiřující element zvolíme kvadratický polynom, 2D interpolant získáme řešením rovnice

$$f(x,y)=\sum_{i=1}^{n}\lambda_i\phi\left(\sqrt{(x-x_i)^2+(y-y_i)^2}\right)+\gamma_0+\gamma_1 x^2+\gamma_2 y^2+\gamma_3 xy+\gamma_4 x+\gamma_5 y \qquad \textbf{(2.4.2.1)}$$

a lineární systém se rozroste následujícím způsobem:

$$\left[\begin{array}{cccc|ccccccc} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} & x_1^2 & y_1^2 & x_1y_1 & x_1 & y_1 & 1 \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2n} & x_2^2 & y_2^2 & x_2y_2 & x_2 & y_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 1 \\ \phi_{n1} & \phi_{n2} & \cdots & \phi_{nn} & x_n^2 & y_n^2 & x_ny_n & x_n & y_n & 1 \\ \hline x_1^2 & x_2^2 & \cdots & x_n^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ y_1^2 & y_2^2 & \cdots & y_n^2 & 0 & \ddots & & & & 0 \\ x_1y_1 & x_2y_2 & \cdots & x_ny_n & 0 & & \ddots & & & 0 \\ x_1 & x_2 & \cdots & x_n & 0 & & & \ddots & & 0 \\ y_1 & y_2 & \cdots & y_n & 0 & & & & \ddots & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c}\lambda_1\\ \lambda_2\\ \vdots\\ \lambda_n\\ \hline \gamma_1\\ \gamma_2\\ \gamma_3\\ \gamma_4\\ \gamma_5\\ \gamma_0\end{array}\right]=\left[\begin{array}{c}h_1\\ h_2\\ \vdots\\ h_n\\ \hline 0\\ 0\\ 0\\ 0\\ 0\\ 0\end{array}\right] \qquad \textbf{(2.4.2.2)}$$

2.4.3. Konstanta

Lineární systém lze rozšířit nejen polynomem, ale také přidáním rozšiřující konstanty. V takovém případě 2D interpolant vypadá následovně:

$$f(x,y)=\sum_{i=1}^{n}\lambda_i\phi\left(\sqrt{(x-x_i)^2+(y-y_i)^2}\right)+\gamma \qquad \textbf{(2.4.3.1)}$$

a příslušný lineární systém:

$$\left[\begin{array}{cccc|c} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} & 1 \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2n} & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 \\ \phi_{n1} & \phi_{n2} & \cdots & \phi_{nn} & 1 \\ \hline 1 & 1 & 1 & 1 & 0 \end{array}\right]\left[\begin{array}{c}\lambda_1\\ \lambda_2\\ \vdots\\ \lambda_n\\ \hline \gamma\end{array}\right]=\left[\begin{array}{c}h_1\\ h_2\\ \vdots\\ h_n\\ \hline 0\end{array}\right] \qquad \textbf{(2.4.3.2)}$$

2.5. Bázové funkce

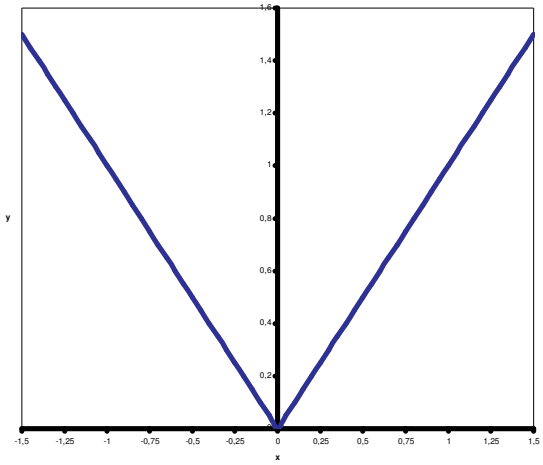
Volba bázové funkce $\phi(r), r \geq 0$ zásadním způsobem ovlivňuje tvar interpolantu. Můžeme je rozdělit do dvou skupin, a to na

- po částech hladké funkce
- nekonečně hladké funkce

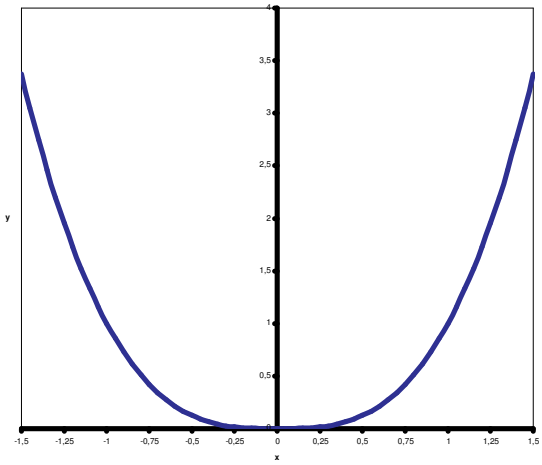
Na obě tyto skupiny se nyní podíváme a uvedeme jejich typické zástupce.

K těmto vybraným funkcím je pro představu uveden i jejich průběh ve 2D případě na intervalu $x \in <-1.5;1.5>$.

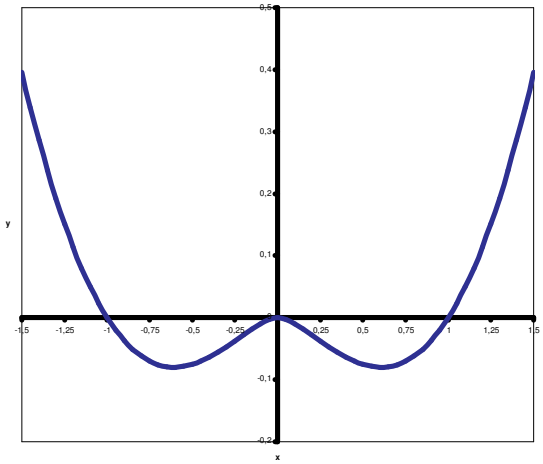
2.5.1. Po částech hladké funkce



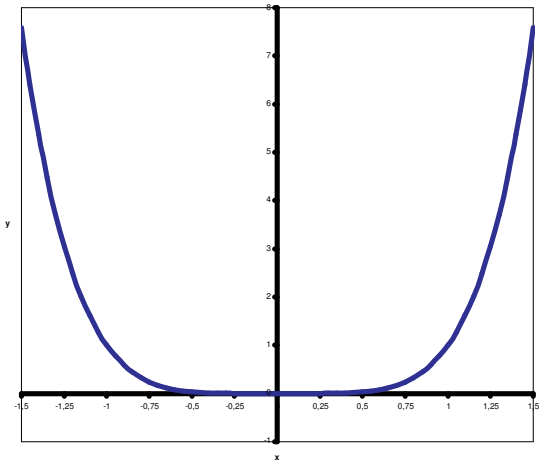
Lineární r



Kubická r^3



Thin-plate spline (TPS) $r^2 \log r$



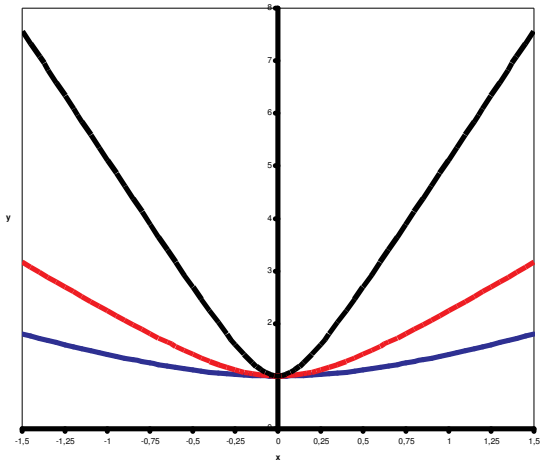
Quintic r^5

Pro po částech hladké funkce platí, že se rostoucím počtem bodů interpolant konverguje k dostatečně hladké funkci. Míra konvergence je přímo úměrná hladkosti bázové funkce a zvětšuje se společně se zvětšující se dimenzí. Hladkost bázové funkce má vliv také na stabilitu rozšířeného lineárního systému. Se zvětšujícím se počtem bodů ve stanovené oblasti se algebraicky zvětšuje i číslo podmíněnosti matice lineárního systému s mírou vztahující se k nepravidelnosti $\phi(r)$ v počátku.

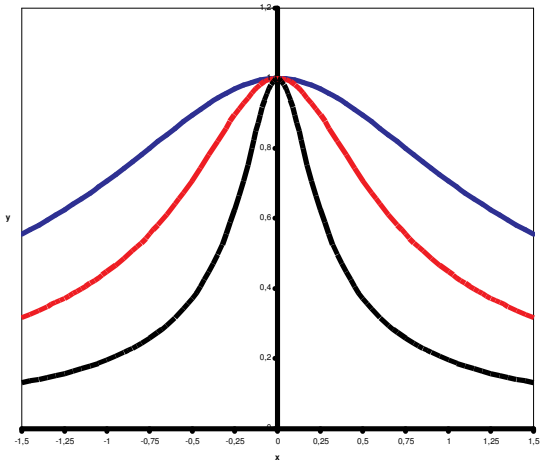
2.5.2. Nekonečně hladké funkce

Grafy zobrazují průběhy funkcí pro následující různé hodnoty parametru ε:

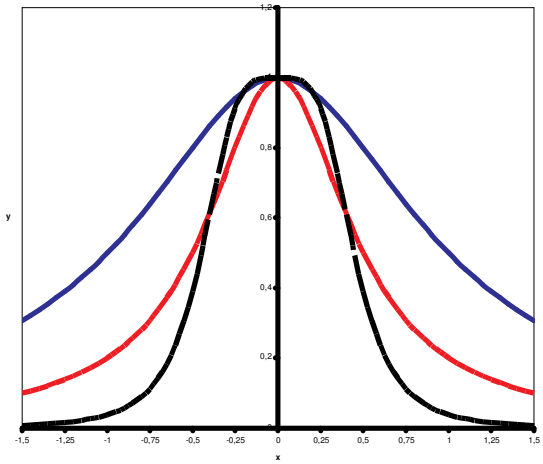
modrá $\varepsilon = 1$
červená $\varepsilon = 2$
černá $\varepsilon = 5$



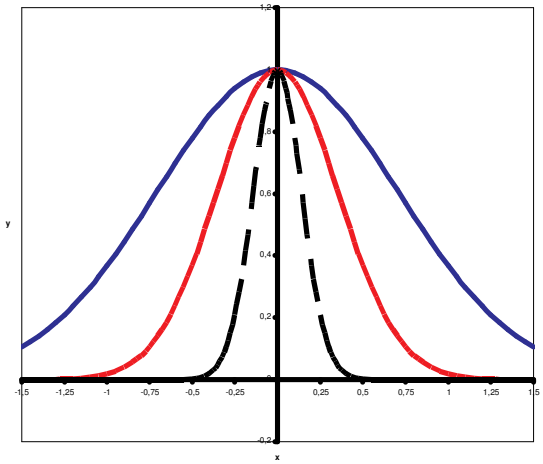
Multiquadric $\sqrt{1+(\varepsilon r)^2}$



Inverzní multiquadric $\dfrac{1}{\sqrt{1+(\varepsilon r)^2}}$



Inverzní kvadratická $\dfrac{1}{1+(\varepsilon r)^2}$



Gaussova $e^{-(\varepsilon r)^2}$

Mezi nejvíce používané funkce patří *thin-plate spline (TPS)* bázová funkce. Tato funkce má fyzikální opodstatnění a je uváděna pro interpolace ve 2D. Ve 3D je jejím ekvivalentem *kubická* bázová funkce. Kvadratické bázové funkce a bázové funkce r^{2k} pro $k=1,\ldots,n$ obecně, se nepoužívají.
Klasická lineární bázová funkce byla vůbec jako první použita pro RBF metodu. Použití této bázové funkce v podstatě znamená interpolaci dat lineární kombinací posunující se absolutní hodnoty bázové funkce, kde vrcholem každé bázové funkce je jeden ze zdrojových bodů. Tato funkce se příliš nehodila pro interpolaci, proto byla rozšířena tak, aby byla bázová funkce spojitě diferencovatelná. Tak byla odvozena multiquadric bázová funkce.

Na rozdíl od po částech hladkých bázových funkcí, přesnost a stabilita pro nekonečně hladké bázové funkce závisí na počtu datových bodů a hodnotě parametru ε. Pro konstantní počet bodů může být přesnost často zlepšena zmenšením parametru ε. Zmenšení parametru ε nebo zvětšení počtu bodů má významný vliv na stabilitu lineárního systému. Pro konstantní ε číslo

podmíněnosti matice lineárního systému roste exponenciálně s počtem přibývajících bodů. Pro konstantní počet datových bodů dochází ke zvětšování s $\varepsilon \to 0$.
Výhodou multiquadric bázové funkce je, že i v základním tvaru RBF metody (vztah **2.3.3**) poskytuje jednoznačná řešení. Multiquadric RBF jsou nekonečně hladké a s volbou parametru ε se chovají tak, že malé ε vede k dobré přesnosti a velké ε zabezpečuje dobrou podmíněnost. Další možnou volbou bázové funkce je funkce Gaussova typu.

2.5.3. Compactly Supported RBF

Velice zajímavé je i vyžití tzv. *Compactly Supported* funkcí (CSRBF), které se v posledních letech staly velmi oblíbenými. Jedná se o lokální bázové funkce, které zaručují, že matice $\boldsymbol{A}$ lineárního systému je *pozitivně definitní*.

Definice:
Symetrická matice $\boldsymbol{A}$ je *pozitivně definitní*, jestliže pro každý nenulový sloupcový vektor $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ platí

$$\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} = \sum_{i,j=1}^{n} x_i a_{ij} x_j > 0 . \qquad \textbf{(2.5.3.1)}$$

CSRBF využívají principu lokality B-splinu, tzn. že změna jednoho bodu vyvolá pouze lokální změnu interpolační funkce. Na základě mocninné funkce jsou odvozeny výpočetně nenáročně CSRBF:

$$\phi(r) = \begin{cases} (1-r)^p P(r) & 0 \le r \le 1 \\ 0 & r > 1 \end{cases}, \qquad \textbf{(2.5.3.2)}$$

kde $P(r)$ je polynom. Funkce mají různé stupně spojitosti C^k a jsou určeny pro různé dimenze d. Následující tabulka představuje některé CSRBF [Wend95]:

$d=1$	$(1-r)_+$	C^0
	$(1-r)_+^3 (3r+1)$	C^2
	$(1-r)_+^5 (8r^2+5r+1)$	C^4
$d=3$	$(1-r)_+^2$	C^0
	$(1-r)_+^4 (4r+1)$	C^2
	$(1-r)_+^6 (35r^2+18r+3)$	C^6
	$(1-r)_+^8 (32r^{3'}+25r^2+8r+1)$	C^6
$d=5$	$(1-r)_+^3$	C^0
	$(1-r)_+^5 (5r+1)$	C^2
	$(1-r)_+^7 (16r^2+7r+1)$	C^4

Tabulka 2.5.3

Uvedené funkce mají poloměr působnosti (*radius of support*) roven 1, tedy na intervalu <-1, 1>. Poloměr bázové funkce však lze upravit na $\phi(r/\alpha)$. Tímto získáme funkce s poloměrem působnosti α.

Definice:
k-d strom je vícerozměrný binární strom, kde x je kořenem s podstromy T_{levy}, T_{pravy} a platí

$$\begin{aligned} \forall y \in T_{levy} &: y^d \le x^d \\ \forall y \in T_{pravy} &: y^d > x^d \end{aligned}, \qquad \textbf{(2.5.3.3)}$$

dimenze třídění d se mění s každou úrovní stromu.

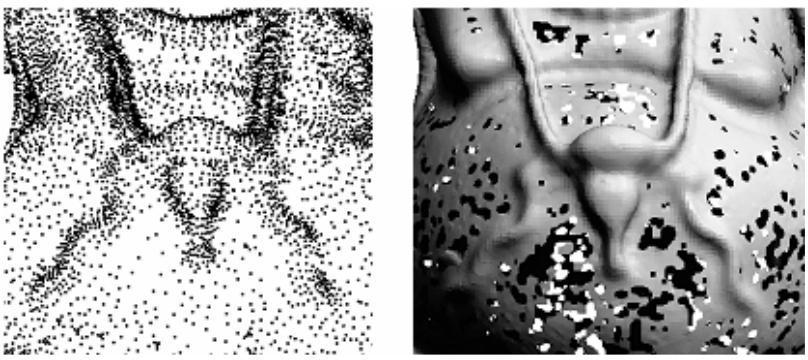
Použití CSRBF a vhodné datové struktury poskytuje urychlení ve všech fázích algoritmu interpolace [Morse01]:

- **Vytváření lin. systému** – protože mají CSRBF pouze omezený dosah (poloměr působnosti), nabývají příliš vzdálené body hodnoty 0. Proto lze např. za použití k-d stromu vypočítat hodnoty pro množinu všech bodů v poloměru působnosti od aktuálního bodu v čase *log n.* Výsledná matice je velice řídká.
- **Řešení lin. systému** – čas výpočtu řídké matice závisí na počtu nenulových hodnot.
- **Výpočet interpolantu** – opět lze využít k-d stromu pro vyhodnocení hodnoty v daném bodě, tzn. *log n* operací.

Volba poloměru působnosti je velmi důležitá a ovlivňuje výsledek při použití CSRBF. Pokud bude příliš malý, může se stát, že bázová funkce např. nedokáže vyplnit díry. Naopak příliš velký poloměr sníží řídkost matice $\boldsymbol{A}$ a zvětší se tak výpočetní náročnost lineárního sytému, tzn. že se připravujeme o výhodu CSRBF, která zjednodušeně řečeno říká, že body umístěné za poloměrem působnosti funkce už nemají na aktuální bod žádný vliv.
Výhodou klasických radiálních bázových funkcí Gaussova typu, TPS nebo multiquadric oproti CSRBF je jejich jednoduchá reprezentace, která platí pro libovolnou dimenzi prostoru d. Proto jsou definovány různé CSRBF pro různé dimenze.

Může se zdát, že CSRBF jsou nejlepším řešením pro RBF interpolace, ale i ony mají své nedostatky:

- nedokáží opravit neúplná či nepravidelně snímaná data (vznikají díry)
- poloměr působnosti musí být alespoň tak velký, jako je vzdálenost dvou sousedních nasamplovaných bodů



Obrázek 2.5.3: *Množina nerovnoměrně rozložených vstupních bodů a zrekonstruovaný povrch s nedostatečným poloměrem působnosti [Ohtake03].*

2.6. RBF interpolace v praxi

V této kapitole uvedeme způsob, jakým lze z rozptýlených dat získat interpolovaný povrch, a dozvíme se, s čím je potřeba se pro úspěšnou rekonstrukci vypořádat.

2.6.1. Implicitní povrchy

Problém popisu povrchu či rekonstrukce povrchu z bodů lze vyjádřit následujícím způsobem:

Definice [Carr01]**:**

Dána množina n vzájemně různých bodů $\{(x_i, y_i, z_i)\}_{i=1}^{n}$ na povrchu M v $\mathbf{E}^3$, nalezněte povrch $M´$, který je dostatečně přesnou aproximací M.

Model lze zapsat v tzv. *implicitním tvaru* funkcí $f(x, y, z)$. Obsahuje-li povrch M všechny body (x, y, z), splňující podmínku $f(x, y, z) = 0$, říkáme, že f je implicitním popisem M.

Implicitního popisu povrchů se s výhodou využívá např. v CSG (Constructive Solid Geometry), kdy je výsledný model vytvořen z jednoduchých tvarů, jejich kombinací pomocí Booleovských operací (průnik, sjednocení atd.) a vhodnou funkcí, která jednotlivé části vhodně prolíná. Tento popis se využívá především v CAD systémech.
My však chceme objekt definovat pouze jednou spojitou a diferencovatelnou funkcí. Výhoda takového popisu je např. ta, že může být vyhodnocena kdekoliv a s libovolným rozlišením.

2.6.2. Nalezení implicitního popisu povrchu

Jak již bylo řečeno, hledáme funkci f, která implicitně popisuje povrch $M´$ a splňuje:

$$f(x_i, y_i, z_i) = 0 \qquad \text{pro } i = 1, \ldots, n, \tag{2.6.2.1}$$

kde $\{(x_i, y_i, z_i)\}_{i=1}^{n}$ jsou body ležící na povrchu. Abychom se vyhnuli triviálnímu řešení, kdy jsou všechny hodnoty nulové, je potřeba do vstupních dat přidat tzv. *off-surface* body s nenulovými hodnotami.

2.6.2.1. Off-surface body

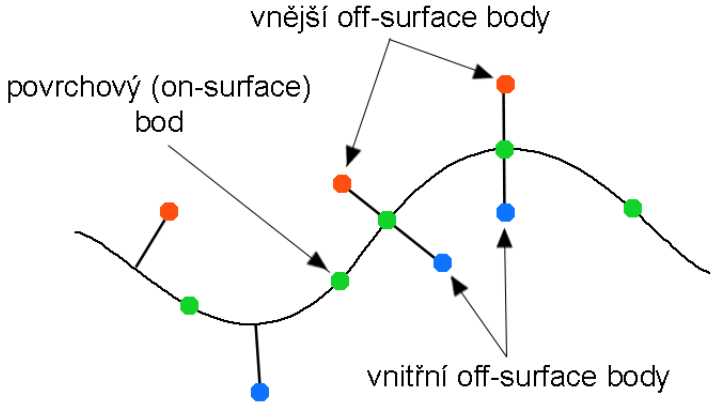
Pokud začneme přidávat tyto body, pak interpolační problém lze popsat následujícím způsobem:

$$\begin{aligned} &f(x_i, y_i, z_i) = 0 && \text{pro } i = 1, \ldots, n && \text{(povrchové body)} \\ &f(x_i, y_i, z_i) = d_i \neq 0 && \text{pro } i = n+1, \ldots, N && \text{(off-surface body)} \end{aligned} \tag{2.6.2.1.1}$$

Nyní však vyvstává problém, jak získat ony off-surface body $\{(x_i, y_i, z_i)\}_{i=n+1}^{N}$ a jim odpovídající hodnoty d_i. Možností se nabízí několik. Pravděpodobně nejjednodušší variantou bude přidání jediného bodu (se zápornou funkční hodnotou, kvůli orientaci) uvnitř povrchu, přibližně uprostřed tělesa (v jeho těžišti). Ne vždy je však snadné určit, zda jsme uvnitř či vně

tělesa, takže toto řešení, kdy nám stačí pouze jediný bod pro celý povrch, nemusí být vždy tak snadné, jak se na první pohled může zdát.
K problému lze přistupovat i opačným směrem, tzn. přidávat body vně tělesa. Vycházíme z předpokladu, že známe rozměry modelu a tedy obklopit jej několika off-surface body (vnější body obvykle mají kladnou funkční hodnotu) by neměl být problém.
Oba zmiňované přístupy však trpí dvěma hlavními problémy. Prvním je, že takto přidané off-surface body jsou získány hrubým odhadem, především co se funkční hodnoty těchto bodů týče. Obecně se v obou případech (body uvnitř či vně) hodnoty skutečně pouze odhadují, takže ve výsledku mohou negativně ovlivnit tvar hledaného povrchu.

Je-li možné na daných datech nějakým způsobem určit či alespoň odhadnout normály, je nejlepším řešením patrně použití *znaménkové funkce* (*signed-distance function*), kde d_i je vzdálenost k nejbližšímu povrchovému bodu. Znaménko hodnoty d_i se pak nastaví podle umístění off-surface bodu – body vně povrchu mají hodnotu kladnou, zatímco body uzavřené povrchem mají tuto hodnotu zápornou. Tyto off-surface body jsou generovány ve směru normály povrchu a měly by mít určeno, zda jsou vně či uvnitř povrchu.



Obrázek 2.6.2.1.A

Zkušenosti ukázaly, že je dobré pro každý bod mít dva off-surface body – jeden uvnitř a jeden vně povrchu. Je třeba si však uvědomit, že v takovém případě se lineární systém zvětší na trojnásobek oproti situaci, kdy bychom ho vytvářeli pouze ze známých vstupních bodů! Problémem však stále zůstává jak správně určit normálu pro přidání off-surface bodu a také určení vzdálenosti od povrchu, viz obrázek:

***Obrázek 2.6.2.1.B: Vlevo**: povrchové body získané laserovým scannerem jsou zobrazeny zeleně, červeně jsou pak nejvzdálenější vnější off-surface body, modře nejvzdálenější vnitřní off-surface body. **Uprostřed**: správné určení vzdáleností off-surface bodů. **Vpravo**: špatné určení vzdáleností off-surface bodů. [Carr01]*

V našem případě, kdy máme nerovnoměrně rozložená vstupní data, musíme normály odhadnout z okolí každého bodu. To zahrnuje odhad nejen směru normály, ale také zda míří z povrchu ven či dovnitř. Množinu bodů můžeme například lokálně aproximovat rovinou k určení těchto informací. K odhadu polohy bodu (uvnitř či vně) můžeme použít např. polohu scanneru je-li známa. Obecně je obtížné tento odhad provést kdekoliv, naštěstí není nutné odhadovat normálu v každém bodě. Pokud není některá zjištěná hodnota v konkrétním bodě jednoznačná, normálu v tomto bodě jednoduše neurčíme. Stále máme v daném místě informaci z povrchového bodu.
Jestliže máme určenou množinu normál, musíme zajistit, aby neprotínaly jiné části povrchu, jinak získáme nevhodné off-surface body. Vhodný off-surface bod je takový, že jemu nejbližší povrchový bod je bodem, který ho generuje. Pokud tomu tak není, dostáváme špatné výsledky, jak jsme mohli vidět na obrázku ***2.6.2.1.B*** vpravo, kdy byly off-surface body umístěny vždy v konstantní vzdálenosti od povrchu.

2.7. Řešení lineárního systému

V této kapitole si představíme vybrané metody řešení soustavy lineárních rovnic.

2.7.1. LU rozklad

Častou metodou pro řešení soustavy rovnic je LU rozklad, především díky své jednoduchosti a schopnosti řešit několik systémů se stejnou maticí $\boldsymbol{A}$ a různým vektorem pravých stran $\boldsymbol{b}$.
Každá regulární čtvercová matice $\boldsymbol{A}$ může být zapsána jako součin matic $\boldsymbol{A = LU}$. Dolní trojúhelníková matice $\boldsymbol{L}$ má na hlavní diagonále jedničky a nad ní pouze nuly, zatímco horní trojúhelníková matice $\boldsymbol{U}$ má nuly pod diagonálou. Jedničky na diagonále matice $\boldsymbol{L}$ nám zaručují existenci pouze jednoho LU rozkladu matice $\boldsymbol{A}$. Fáze, kdy se daná soustava převádí na soustavu s horní a dolní trojúhelníkovou maticí bývá označována jako tzv. *přímý chod*.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}. \qquad \textbf{(2.7.1.1)}$$

Výpočet řešení soustavy (označován jako *zpětný chod*) je pak

$$\begin{aligned} &\boldsymbol{Ax = LUx = b} \\ &\boldsymbol{Ly = b} \\ &\boldsymbol{Ux = y}. \end{aligned} \qquad \textbf{(2.7.1.2)}$$

Soustavu lineárních rovnic lze, jak již bylo uvedeno, řešit se stejnou maticí $\boldsymbol{A}$ (resp. jejím LU rozkladem) pro různé vektory pravých stran $\boldsymbol{b}$. Této vlastnosti lze dokonce využít pro řešení soustavy s více pravými stranami současně.
Vektor pravých stran $\boldsymbol{b}$ nahradíme maticí $\boldsymbol{B = (b_1, b_2, \ldots, b_n)}$, kde $\boldsymbol{b_i}$ jsou různé pravé strany. Obdobně upravíme i $\boldsymbol{x}$ a $\boldsymbol{y}$ na matice stejného typu jako je nyní $\boldsymbol{B}$. Potom platí, že *i*-tý sloupec matice $\boldsymbol{X = (x_1, x_2, \ldots, x_n)}$ je řešením pro *i*-tý sloupec $\boldsymbol{b_i}$ matice pravých stran $\boldsymbol{B}$.
Výše uvedené vlastnosti lze s výhodou využít např. při rekonstrukci obrazů pomocí RBF, kdy lze spočítat řešení pro všechny tři barevné složky RGB najednou (tzn. nalézt řešení pro

vektory pravých stran odpovídající jednotlivým barevným složkám). Výpočet se oproti případu, kdy řešíme každou barevnou složku samostatně (stále však s využitím jednoho a téhož LU rozkladu matice $\boldsymbol{A}$) zkrátí přibližně na polovinu.
Pro LU rozklad (přímý chod) je potřeba $2n^3/3$ operací, pro zpětný chod n^2 operací.

2.7.2. Choleského rozklad

Někdy označován též jako tzv. *odmocninová metoda.* Pokud je symetrická matice $\boldsymbol{A}$ pozitivně definitní, můžeme zvolit podmínky řešení takové, že $\boldsymbol{U}$ je rovno transponované matici $\boldsymbol{L}$. Pro dolní trojúhelníkovou matici s kladnými prvky na diagonále $\boldsymbol{L}$ a její konjugovanou transpozici $\boldsymbol{L}^*$ platí tedy $\boldsymbol{LL}^* = \boldsymbol{A}$. Pro matici $\boldsymbol{A}$, splňující výše uvedené požadavky, Choleského rozklad vždy existuje a je právě jeden.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & \cdots & l_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{nn} \end{bmatrix}. \qquad \textbf{(2.7.2.1)}$$

Pro $i = 1,\ldots,n$ a $j = i+1,\ldots,n$ dostaneme

$$l_{ii} = \sqrt{\left(a_{ii} - \sum_{k=1}^{i-1} {l_{ik}}^2 \right)} \qquad l_{ji} = \left(a_{ji} - \sum_{k=1}^{i-1} l_{jk} l_{ik} \right) / l_{ii}. \qquad \textbf{(2.7.2.2)}$$

Protože matice $\boldsymbol{A}$ je symetrická a pozitivně definitní, bude výraz pod odmocninou vždy kladný a všechny hodnoty l_{ij} reálné.

Při řešení lineárního systému $\boldsymbol{Ax} = \boldsymbol{b}$ postupujeme tak, že nejprve získáme Choleského rozklad $\boldsymbol{A} = \boldsymbol{LL}^T$, potom vyřešíme $\boldsymbol{Ly} = \boldsymbol{b}$ pro $\boldsymbol{y}$ a nakonec $\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{y}$ pro $\boldsymbol{x}$:

$$\begin{aligned} &\boldsymbol{Ax} = \boldsymbol{LL}^T\boldsymbol{x} = \boldsymbol{b} \\ &\boldsymbol{Ly} = \boldsymbol{b} \\ &\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{y}. \end{aligned} \qquad \textbf{(2.7.2.3)}$$

Pro Choleského rozklad je potřeba $n^3/3$ operací a je tedy dvakrát efektivnější než LU rozklad.

2.7.3. SVD rozklad (Singulární rozklad matice)

Pro každou matici $\boldsymbol{A}$ (i singulární) lze nalézt její SVD rozklad, vyjádřený součinem tří matic

$$\boldsymbol{A} = \boldsymbol{USV}^T, \qquad \textbf{(2.7.3.1)}$$

kde $\boldsymbol{U}$ a $\boldsymbol{V}$ jsou matice unitární (tzn. matice pro které platí $\boldsymbol{UU}^T = \boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$ a $\boldsymbol{VV}^T = \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$). Matice $\boldsymbol{U}$ je tvořena vlastními vektory matice $\boldsymbol{AA}^T$, matice $\boldsymbol{V}$ pak obsahuje vlastní vektory matice $\boldsymbol{A}^T\boldsymbol{A}$.
$\boldsymbol{S}$ je matice diagonální s nezápornými prvky s_{ii}, což jsou tzv. *singulární čísla* matice $\boldsymbol{A}$, umístěné na diagonále sestupně

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{D} & 0 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{D} = diag(s_{11}, s_{22}, \ldots, s_{nn}) \qquad s_{11} \geq s_{22} \geq s_{33} \geq \ldots \geq s_{nn} > 0$$

Využití při řešení soustav lineárních rovnic:
Pro soustavu $\boldsymbol{Ax} = \boldsymbol{b}$ hledáme matici $\boldsymbol{A}^+$ tak, aby platilo $\boldsymbol{x} = \boldsymbol{A}^+\boldsymbol{b}$.

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad \rightarrow \qquad \boldsymbol{USV}^T\boldsymbol{x} = \boldsymbol{b}$$

Z této rovnice postupným násobením vhodné matice zleva získáme hledaný tvar matice A^+

$$\begin{aligned} \left(\boldsymbol{U}^T\boldsymbol{U}\right)\boldsymbol{SV}^T\boldsymbol{x} &= \boldsymbol{U}^T\boldsymbol{b} && / \text{ vynásobíme zleva } \boldsymbol{U}^T \\ \left(\boldsymbol{S}^{-1}\boldsymbol{S}\right)\boldsymbol{V}^T\boldsymbol{x} &= \boldsymbol{S}^{-1}\boldsymbol{U}^T\boldsymbol{b} && / \text{ vynásobíme zleva } \boldsymbol{S}^{-1} \\ \left(\boldsymbol{VV}^T\right)\boldsymbol{x} &= \boldsymbol{VS}^{-1}\boldsymbol{U}^T\boldsymbol{b} && / \text{ vynásobíme zleva } \boldsymbol{V} \end{aligned}$$

Získali jsme tedy:

$$\boldsymbol{x} = \boldsymbol{VS}^{-1}\boldsymbol{U}^T\boldsymbol{b}, \qquad \textbf{(2.7.3.2)}$$

z čehož plyne

$$\boldsymbol{A}^+ = \boldsymbol{VS}^{-1}\boldsymbol{U}^T. \qquad \textbf{(2.7.3.3)}$$

Žádnou inverzi (abychom získali $\boldsymbol{S}^{-1}$) není potřeba dělat, protože pro diagonální matici platí, že její inverzní matice má hodnoty na diagonále převrácené.

$$\boldsymbol{S}^{-1} = \begin{bmatrix} \frac{1}{s_{11}} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{s_{22}} & 0 & \cdots & 0 \\ 0 & 0 & \frac{1}{s_{33}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{s_{nn}} \end{bmatrix}. \qquad \textbf{(2.7.3.4)}$$

Pokud je matice $\boldsymbol{A}$ singulární, pak $\boldsymbol{A}^+$ získaná popsaným způsobem je tzv. *pseudoinverzní.* Pro SVD rozklad je potřeba $26n^3$ operací.

2.7.4. Inverze blokové matice

Pro obecnou blokovou matici platí:

$$\left[\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{B} \\ \hline \boldsymbol{C} & \boldsymbol{D} \end{array}\right]^{-1} = \left[\begin{array}{c|c} \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D} - \boldsymbol{CA}^{-1}\boldsymbol{B})^{-1}\boldsymbol{CA}^{-1} & -\boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D} - \boldsymbol{CA}^{-1}\boldsymbol{B})^{-1} \\ \hline -(\boldsymbol{D} - \boldsymbol{CA}^{-1}\boldsymbol{B})^{-1}\boldsymbol{CA}^{-1} & (\boldsymbol{D} - \boldsymbol{CA}^{-1}\boldsymbol{B})^{-1} \end{array}\right]. \qquad \textbf{(2.7.4.1)}$$

V našem speciálním případě platí:

$$\left[\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{P} \\ \hline \boldsymbol{P^T} & \boldsymbol{0} \end{array}\right]^{-1} = \left[\begin{array}{c|c} \boldsymbol{A^{-1} - (A^{-1}P)(P^TA^{-1}P)^{-1}(A^{-1}P)^T} & \boldsymbol{(A^{-1}P)(P^TA^{-1}P)^{-1}} \\ \hline \boldsymbol{(P^TA^{-1}P)^{-1}(A^{-1}P)^T} & \boldsymbol{-(P^TA^{-1}P)^{-1}} \end{array}\right]^{-1} = \left[\begin{array}{c|c} \boldsymbol{A^{-1} - GFG^T} & \boldsymbol{GF} \\ \hline \boldsymbol{FG^T} & \boldsymbol{-F} \end{array}\right], \quad \textbf{(2.7.4.2)}$$

kde

$$\boldsymbol{G = A^{-1}P} \qquad \text{a} \qquad \boldsymbol{F = \left(P^TG\right)^{-1}}$$

Tato metoda je zde zmíněná spíše pro zajímavost, neboť výpočet submatice $\boldsymbol{F}$ je příliš náročný na to, abychom této metodě z hlediska rychlosti dále věnovali více pozornosti.

2.7.5. GMRES (Generalized Minimal Residual)

Iterativní metoda, jejímž výsledkem je pouze aproximace řešení systému $\boldsymbol{Ax = b}$.
Myšlenka metody GMRES spočívá v minimalizaci rezidua, tj. vektoru $\boldsymbol{r = Ax - b}$. GMRES patří mezi metody pracujícími s *Krylovovými podprostory*, tzn. že hledají řešení v množině

$$\boldsymbol{x_0 + K_m}, \qquad \textbf{(2.7.5.1)}$$

kde $\boldsymbol{x_0}$ je počáteční odhad a $\boldsymbol{K_m}$ (Krylovův podprostor) je vektorový prostor generovaný množinou vektorů

$$\boldsymbol{K_m} = span\left\{\boldsymbol{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0}\right\}, \qquad \textbf{(2.7.5.2)}$$

kde *span* značí lineární obal a $\boldsymbol{r_0 = b - Ax_0}$. Aproximace jsou pak ve tvaru

$$\boldsymbol{A^{-1}b \approx x_m = x_0 + q_{m-1}(A)r_0}, \qquad \textbf{(2.7.5.3)}$$

přičemž q_{m-1} označuje polynom stupně nejvýše $m-1$.

Vlastnosti Krylovových podprostorů:
Definujme $grade_A(\boldsymbol{v})$ jako stupeň minimálního polynomu p vektoru $\boldsymbol{v}$, tj. nenulového polynomu nejnižšího možného stupně, pro nějž platí $p(\boldsymbol{A})\boldsymbol{v} = 0$. Potom platí:

Věta: Nechť $m = grade_A(\boldsymbol{v})$. Pak $\boldsymbol{K_m}$ je invariantní s $\boldsymbol{A}$ a platí $\boldsymbol{K_m = K_l}$ pro všechna $l \geq m$
Věta: Krylovův podprostor $\boldsymbol{K_m}$ má dimenzi m právě tehdy, když $grade(\boldsymbol{v}) \geq m$.

Z těchto vět vyplývá, že $dim(\boldsymbol{K_m}) = min\{m, grade(\boldsymbol{v})\}$.
Důležitou částí metody je konstrukce posloupnosti ortogonálních vektorů. Po vygenerování nového vektoru je nutné provést ortogonalizaci vzhledem k předchozím a tedy s rostoucím počtem iterací roste i čas potřebný k provedení další iterace. V praxi se tedy často používá restartovaný GMRES – GMRES(m), kdy se po m iteracích celá posloupnost vygenerovaných

vektorů zahodí a dosažený výsledek se použije jako inicializace pro další posloupnost iterací. Neexistuje však univerzální způsob, jak určit optimální *m* pro daná vstupní data – pokud je zvolena příliš velká hodnota, může to mít výrazný dopad na rychlost výpočtu.
Pro GMRES(*m*) je potřeba *(2m+7)n* operací, kde *m* je počet iterací a *n* je počet prvků.

2.8. Urychlovací techniky

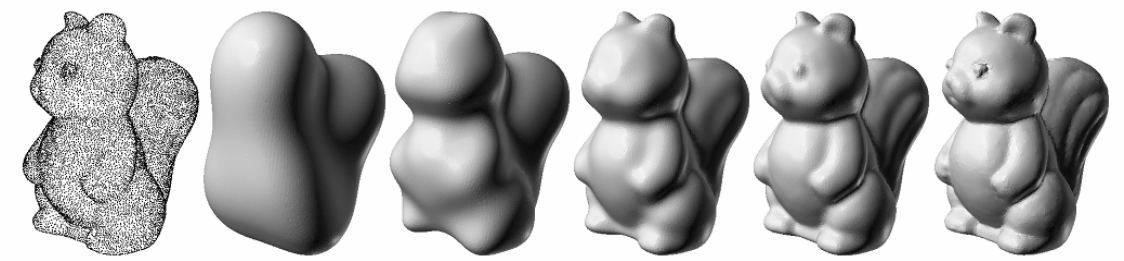
RBF metoda je výpočetně velice náročná, proto existuje mnoho různých způsobů, jak metodu urychlit. Některé z nich jsou založeny na preprocessingu (např. redukce vstupní množiny bodů) a jiné na urychlení řešení lineárního systému.

2.8.1. Volba metody pro řešení systému rovnic

Jedna z možných urychlovacích metod může být také volba metody na řešení lineárního systému rovnic. Více viz kapitola **2.7 Řešení lineárního systému**.

2.8.2. Použití CSRBF

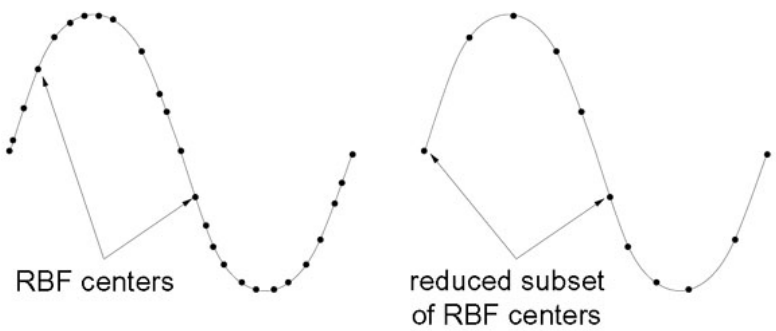
Pokud máme vhodná data a použijeme CSRBF funkce společně s vhodnou datovou strukturou, jako je např. *k*-d strom, můžeme dosáhnout významného urychlení. Více viz kapitola **2.5.3 Compactly Supported RBF**.



Obrázek 2.8.2: *Množina bodů a výsledky interpolace povrchu pomocí CSRBF od nejhrubších až po nejjemnější [Ohtake03]*

2.8.3. Redukce bodů

Základní RBF metoda používá všechny body ze vstupní množiny bodů. Ta samá vstupní data však mohou být aproximována se zadanou přesností pomocí mnohem méně bodů, viz obrázek



Obrázek 2.8.3 *[Carr01]*

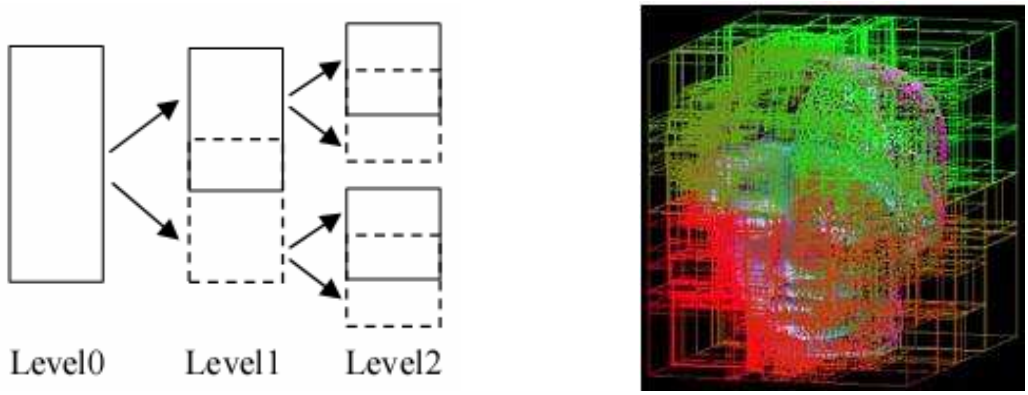
Můžeme zde použít algoritmus, který postupně prokládá množinu bodů s definovanou přesností. Postup algoritmu:

1. Vyber podmnožinu ze všech *n* bodů a spočítej RBF pouze pro tuto podmnožinu
2. Vypočítej rozdíl $\varepsilon_i = f_i - f(x_i)$ ve všech bodech
3. Pokud $\max\{|\varepsilon_i|\} <$ *požadovaná přesnost*, tak zastav vypočet
4. Jinak přidej další body kde $\varepsilon_i >$ *požadovaná přesnost*
5. Přepočítej RBF a jdi na 2.

Pokud je v každém bodě určena jiná přesnost δ_i, tak může být podmínka ve 3. kroku nahrazena podmínkou $|\varepsilon_i| < \delta_i$.
Redukce počtu bodů není potřeba, pokud je použita FMM metoda popsaná dále.

2.8.4. Partition of unity (POU)

Základní myšlenkou této metody je rozdělení množiny bodů na několik menších, vzájemně se překrývajících oblasti, které jsou odděleně vyřešeny. Výsledek je poté získán kombinací lokálních funkcí za použití příslušných vah.
Za předpokladu, že celou množinu rozdělíme do *m* podoblastí s přibližně stejným počtem bodů, bude každá podoblast obsahovat průměrně *n/m* bodů. Řešení lineárního systému pak potřebuje $m(n/m)^3$ operací. Lze-li podíl *n/m* brát jako konstantu, klesne složitost na *n*, protože je lineární vzhledem k počtu bodů [Tobor04] a [Wu05].

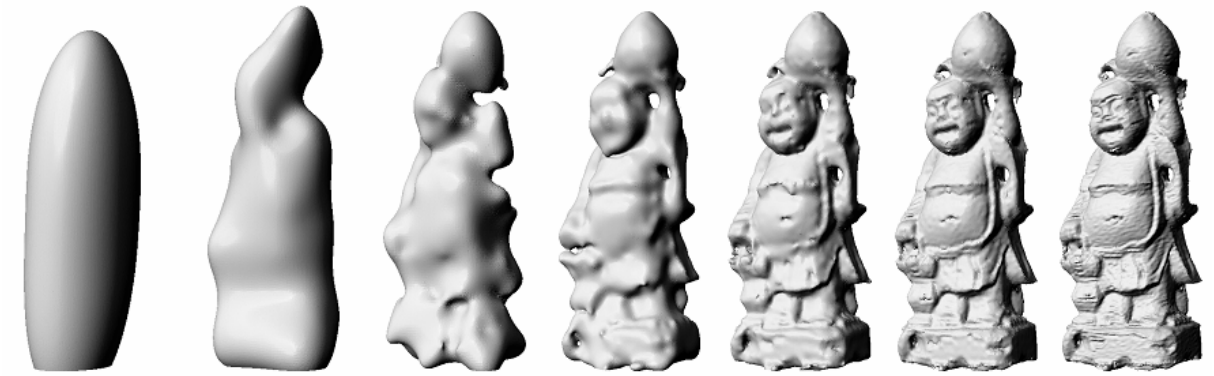


***Obrázek 2.8.4: Vlevo**: Schéma hierarchického rozdělení vstupní množiny na překrývající se oblasti. **Vpravo:** Příklad rozdělení celého prostoru [Wu05].*

2.8.5. Multi-level interpolace

Jako bázové funkce tato metoda používá CSRBF a těží tedy z jejich výhod, především řídkosti matice. Metoda rozdělí vstupní množinu bodů do hierarchické struktury podobné octree a poté pokračuje interpolací způsobem, kdy tvar objektu postupně zpřesňuje rekurzivní interpolací podprostorů. Při každém kroku využívá jiný poloměr působnosti.
Základní myšlenkou je začít s velkou množinou bodů za použití hladké bázové funkce k získání hrubého tvaru objektu. V každém dalším kroku se vezme menší množina bodů a za použití méně hladké bázové funkce s menším poloměrem působnosti se postupně na objektu formují detaily. Těmito postupnými aproximacemi tvaru se získává výsledný objekt. Tato

metoda odstraňuje nedostatky CSRBF, která ve své základní podobě může vytvářet na objektu díry v důsledku omezeného poloměru působnosti [Ohtake03].



Obrázek 2.8.5 *[Ohtake03]*

2.8.6. Fast Multipole Method (FMM)

Využívá zmenšení přesnosti výpočtu, takže pouze aproximuje data. Základem je určení přesnosti výpočtu, která je rozhodující pro kvalitu výsledné aproximace.
Na začátku algoritmu je prostor hierarchicky rozdělen (quadtree či octree). Pak pro výpočet konkrétního bodu rozdělí vstupní body do dvou skupin – *near field*, obsahující blízké body a *far field*, obsahující vzdálené body. Zatímco příspěvky blízkých bodů (*near filed*) jsou vypočítány přesně, pro příspěvky vzdálených bodů je použito několik Laurentových rozvojů. V tomto smyslu je za Laurentův rozvoj považován rozvoj platný mimo určitý poloměr. Každý z těchto rozvojů je pak zakombinován do jednoho výsledného Taylorova rozvoje, který pro naše potřeby znamená rozvoj platný uvnitř určitého poloměru. Skutečnost, že tyto rozvoje – především výsledný Taylorův rozvoj – mohou být vyhodnoceny mnohem rychleji než přesný výpočet *far field* příspěvků znamená podstatné urychlení FMM oproti přímému vyhodnocení např. LU rozkladem [Mullan04]. Případné zájemce odkazuji na www stránky patrně nejvýznamnějšího odborníka na FMM – R. Beatsona [Beatson_WWW].
Důležitým faktem a současně velkou nevýhodou FMM metody je ta skutečnost, že je velice náročná na implementaci.
Pro FMM je potřeba *n.log n* operací.

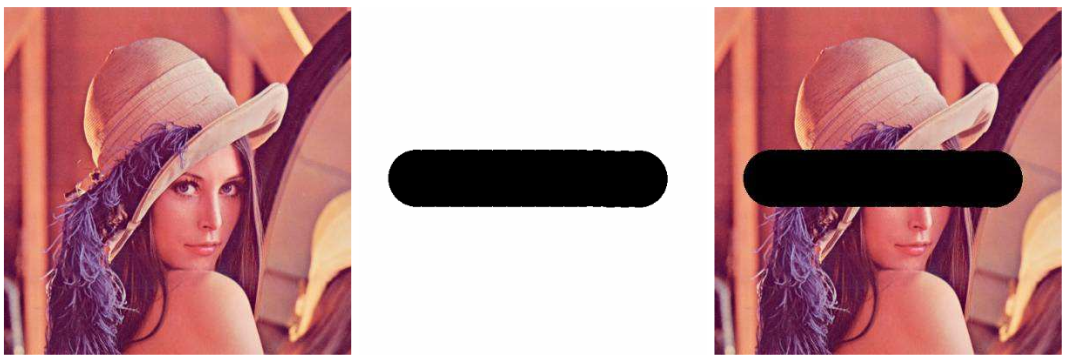
3. Realizační část

V této kapitole se seznámíme s použitými metodami, ukážeme způsoby jejich řešení a navrhneme některá nová řešení, nakonec si uvedeme a zhodnotíme dosažené výsledky. Kapitola je rozdělena na dvě části, kde první z nich se zabývá rekonstrukcí poškozených obrazů a druhá pak extrakcí isočar (2D) resp. isoploch (3D) pomocí RBF.

3.1. Rekonstrukce obrazů

Obrazy (fotografie) mohou být poškozeny několika typy poškození. Mohou být označeny nějakým textem či logem (např. fotografie, na níž je umístěna značka autorství, např. serveru kde byla publikována či se může jednat o záměrné znehodnocení náhledu textury, kdy za poplatek dostane zájemce texturu nepoškozenou). Tento typ se označuje jako *inpainting*. Dalším, v podstatě totožným, typem je poškození textem přes plochu obrazu (např. reklamní text přes pozadí či dokumentující informace). Charakterově úplně jiným typem poškození je *šum*, který se přimísil do obrazové informace např. slabým televizním signálem, přenosem na dlouhé vzdálenosti či prachem. Jiným typem poškození může být pravidelné opakování nějakého vzoru.
Rekonstrukce obrazů pomocí RBF zvládne všechny výše zmiňované typy poškození. Výsledky jsou samozřejmě závislé na množství známé informace, zejména v blízkém okolí poškozeného bodu. Lze očekávat, že u velkých souvislých poškozených ploch bude kvalita rekonstrukce postupně klesat s tím, jak se bude vzdálenost právě rekonstruovaného pixelu vzdalovat od okraje oné plochy. Na okrajích totiž stále známe informace z okolních bodů, ale s každým následujícím rekonstruovaným bodem, vzdalujícím se od okraje, se dále šíří a akumuluje chyba, ke které se během interpolace dopouštíme a které se nevyhneme.

Pro rekonstrukci obrazů je potřeba definovat v obrazu oblast, kterou chceme opravit. Tuto oblast se můžeme pokusit nějakým způsobem lokalizovat automaticky, k tomu bychom však potřebovali informace o způsobu poškození. Další možností je nechat na uživateli, aby nějakým způsobem označil oblast poškození. Nutno poznamenat, že bez tohoto uživatelského zásahu se pro poškození inpaintingem neobejdeme. Pro naše případy jsme zvolili variantu, kdy máme kromě poškozeného obrazu ještě obraz druhý, definující oblast poškození maskou. Tato maska je pak nejlépe černobílý obraz, kde černé pixely označují poškození a tedy místo, kde chceme provést rekonstrukci.



Obrázek 3.1: ***a)*** *Originální obraz,* ***b)*** *maska,* ***c)*** *poškozený obraz*

V následujícím textu se seznámíme s algoritmem Ing. Uhlíře [Uhlir05], který také využívá RBF interpolaci, dále si představíme algoritmus nový, ukážeme si rekonstrukci lineární interpolací a provedeme srovnání těchto metod se zaměřením především na vizuální kvalitu

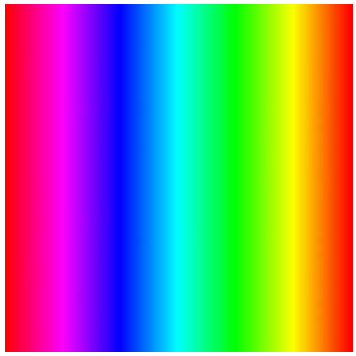
rekonstrukce. Ukážeme si výsledky a provedeme testy vlivu zvolené bázové funkce a polynomu a také vlivu velikosti okolí, ze kterého chceme rekonstruovat na výslednou kvalitu rekonstrukce. Zmíníme také interpolaci v několika vybraných barevných systémech.

3.1.1. Testovací obrazy a masky

Nejprve zavedeme značení vybraných obrazů a masek, na které se budeme dále odkazovat uvedenými názvy. Rozměry všech obrazů i masek jsou 512x512.

Obrazy:

Fruit



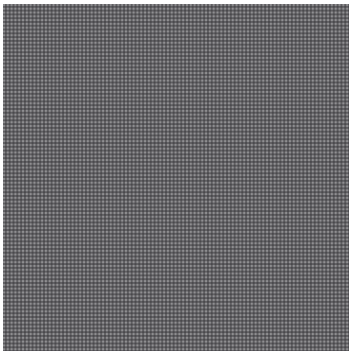
Gradient

Lena

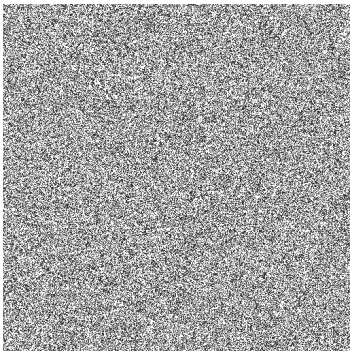
Masky:

Připomeňme, že černá barva označuje oblast poškození a tedy místo, které chceme zrekonstruovat.

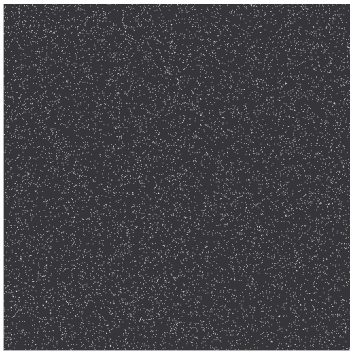
Fun



Grid



Noise45,815%



Noise95,503%

Wide

Maska Grid byla vytvořena tak, že jsme každý lichý řádek a sloupec obrazu označili jako vadný, tzn. známe pouze ¼ pixelů. Noise95,503% je extrémní šum, dosahující 95,503% poškození obrazu.

3.1.2. Algoritmy zpracování obrazu

Pro rekonstrukci obrazu můžeme volit mezi dvěma základními přístupy:

Globální přístup

V globálním přístupu se hledané koeficienty λ_i (v případě použití rozšířeného lineárního systému i koeficienty γ_i) spočítají pro všechny nepoškozené pixely najednou.

- **výhody**: pouze jedno vyhodnocení lineárního systému pro celý obraz, jeden průchod obrazem

- **nevýhody**: velká paměťová a výpočetní náročnost matice vysokého řádu.
  Podívejme se na názorný příklad – řád matice lineárního systému vychází ze součinu rozměrů obrazu, od kterého odečteme počet poškozených pixelů. Např. pro obraz o velikosti 512x512 s 50% poškozením dostáváme matici řádu 131072 (+ 0 až 6 podle typu použitého rozšiřujícího elementu).
  Jenom uložení takové matice v paměti, kdy každá její položka je typu double, zabere 131072 * 131072 * 8 bytů = 128GB!

  V případě rekonstrukce obrazů je vzájemný vliv vzdálených pixelů velice diskutabilní. Proto se v následujícím textu globálním přístupem již nebudeme zabývat.

Lokální přístup

V tomto případě pracujeme vždy jen s částí rekonstruovaného obrazu. Pro každý poškozený pixel nalezneme okolí v zadaném poloměru r následujícím způsobem. Vytvoříme okno o rozměrech $(2r+1)x(2r+1)$ (není-li řečeno jinak) takovým způsobem, že v obou směrech os x a y obrazu se posuneme o r pixelů a nalezneme „bounding box“. Z něj pak vybereme pouze nepoškozené pixely, ze kterých vytvoříme lineární systém. Ostatní body obrazu neovlivní výsledek interpolace. Tedy např. pro poloměr 2 nalezneme okno o velikosti 5x5, přičemž uprostřed tohoto okna se nachází právě zpracovávaný poškozený pixel, jehož interpolant chceme vypočítat. Celý postup opakujeme, dokud v obraze zbývají nezrekonstruované pixely.

- **výhody**: lineární systém je malého řádu a výpočet je tedy rychlejší. Do výpočtu také nezahrnujeme vzdálené pixely, které mohou pouze zanášet chybu.

- **nevýhody**: opakovaný výpočet koeficientů λ_i (resp. γ_i) tzn. vyhledávání okolí, sestavení a vyřešení lineárního systému, opakující se pro každý poškozený pixel

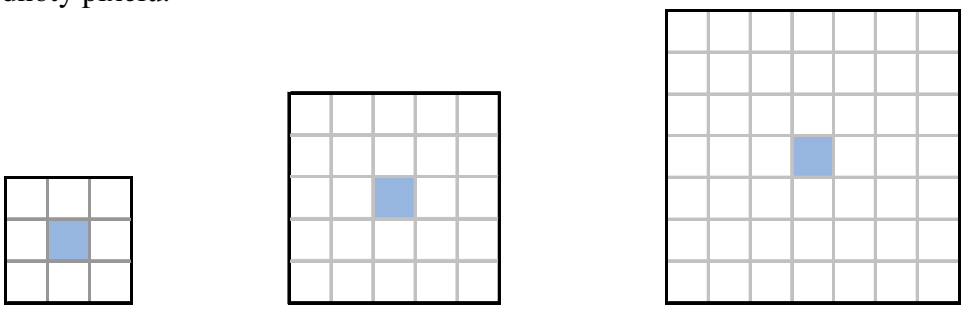
Zvolení lokálního přístupu k rekonstrukci s sebou nese potřebu nastavit některé další parametry, zásadně ovlivňující výslednou rekonstrukci:
poloměr okolí, směr pohybu po obraze, vyhledávání okolí + jeho případná úprava a vyhledávání a přeskakování děr

Algoritmus lokálního přístupu zapsaný v pseudokódu:

```
while ((pocet_poskozenych_pixelu > 0) && pixely_lze_jeste_rekonstruovat)
{
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
        {
            if (pixel_je_poskozeny)
            {
                okoli = Najdi_okoli_pixelu(i, j, polomer_okoli);
                bVect = Vytvor_vektor_pravych_stran(okoli);
                A = Vytvor_linearni_system(i, j, okoli);
                lambdaVect = Vyres_linearni_system(A, bVect);
                pixel = Vypocitej_pixel(i, j, okoli, lambdaVect);
                pocet_poskozenych_pixelu--;
            }
        }
}
```

A. Poloměr okolí

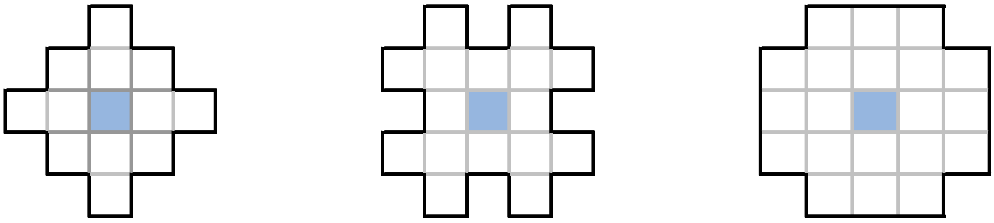
Volba poloměru okolí je velice důležitým faktorem, ovlivňujícím výsledek interpolace. Udává, kolik informace z okolí se má pro interpolaci právě zpracovávaného pixelu použít. Pro rozsáhlé poškození je jeho volba klíčová, protože v okolí bodu se při takovém druhu poškození vyskytuje jen několik málo známých bodů a interpolační funkce procházející jen těmito několika málo body se může chovat nevhodně. Jinou možnost, jak tuto nežádoucí situaci vyřešit si ukážeme v kapitole **3.1.3.2 Přidávání bodů (AddWeightedAveragePoints)**. Z výše uvedeného vyplývá, že čím větší poloměr okolí zvolíme, tím lepších výsledků dosáhneme. Ovšem pozor – po určité hodnotě jsou do interpolace zapojovány buď body příliš vzdálené a zanášejí chybu nebo je rozdíl mezi touto a nižší hodnotou poloměru neznatelný na kvalitu výsledné interpolace či nevýhodný s ohledem na prodloužení doby výpočtu. Zde narážíme na fakt, že pro poloměr r typicky vytváříme okno o rozměrech $(2r+1)x(2r+1)$ s právě zpracovávaným pixelem umístěným uprostřed, které následně prohledáváme na známé hodnoty pixelů.



$r=1$ 8-okolí $r=2$ 24-okolí $r=3$ 48-okolí

Pro poloměr $r=1$ tedy získáme okno $3x3$, což znamená, že z takového okna můžeme získat maximálně $(2r+1)x(2r+1)-1=3x3-1=8$ platných pixelů, a to v případě, že je poškozený pouze prostřední (právě zpracovávaný) pixel. Pokud nastane taková situace, získáme po vytvoření lineárního systému matici řádu 8 (+ počet příspěvků polynomu, tzn. celkový počet koeficientů γ_i. Konkrétně 0 pro případ bez polynomu, 1 pro konstantu, 3 pro lineární polynom a 6 pro kvadratický polynom). Pro poloměr $r=2$ získáme obdobně maximálně matici řádu $5x5-1=24$, pro poloměr $r=3$ už máme matici řádu 48, pro $r=4$ řádu 80 a tak dále. Je vidět, že tato hodnota rychle roste a přihlédneme-li k tomu, že takovouto matici řešíme pro každý poškozený pixel, snadno poznáme, že je potřeba tuto hodnotu poloměru držet na co nejnižší hodnotě, dávající dostatečně kvalitní výsledky interpolace.

Uhlíř [Uhlir05] zmiňuje i jiné vzory volby okolí, než je pravidelné čtvercové okolí.



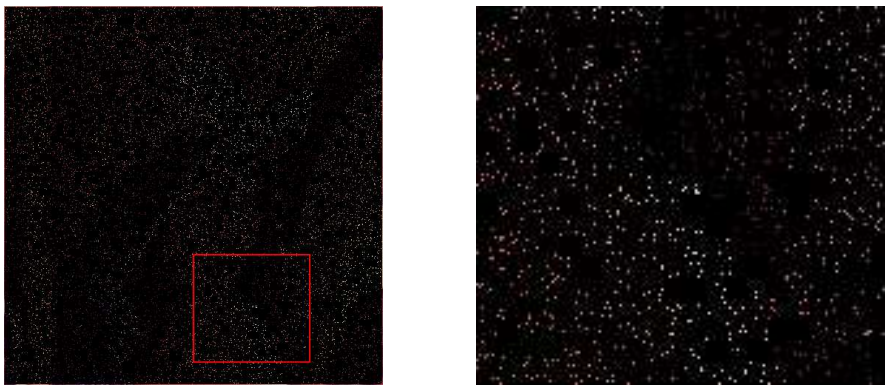
12-okolí 16-okolí 20-okolí

Nakonec však označil za nejvhodnější velikost okolí 24, tzn. poloměr $r = 2$. Tento poloměr dává skutečně dobré výsledky, jak si ukážeme dále, za relativně krátkou dobu výpočtu. Pokud však chceme dosáhnout znatelně ještě lepších výsledků, použijeme $r = 3$, tzn. 48 okolí, ovšem za cenu delšího výpočtu. Konkrétní příklady a jejich porovnání s ohledem na kvalitu výstupu i na čas výpočtu si ukážeme dále v kapitole **3.1.6 Dosažené výsledky**.

Poznámka: pokud z okolí získáme počet bodů menší či roven počtu koeficientů polynomu γ_i (tzn. např. 3 pro lineární polynom), je vhodné tento bod neinterpolovat a vyčkat do další iterace, dokud nebude počet platných pixelů větší. V případě, že interpolaci provedeme rovnou, získáme lineární systém, kde je počet řádek získaných ze známých pixelů menší než počet řádek náležících příspěvkům polynomu. Tyto příspěvky polynomu pak nežádoucím způsobem příliš ovlivní řešení systému, místo aby matici pouze ochránily před singularitou.

B. Směr pohybu po obraze

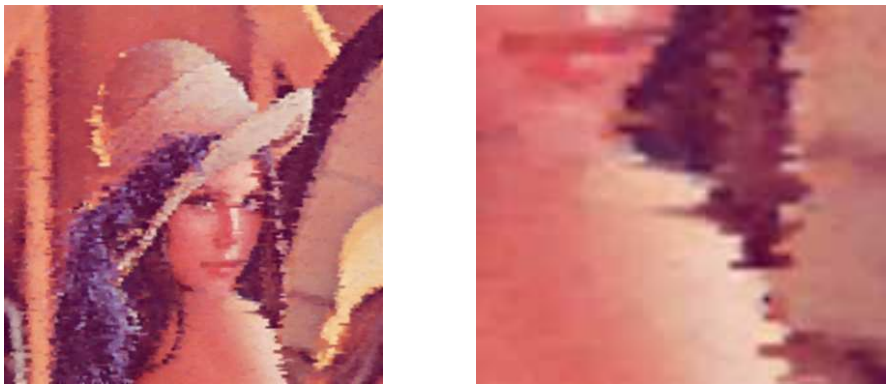
Parametr, ovlivňující z jakého směru budeme zpracovávat obraz, tzn. i směr šíření chyby.
Přiložené ukázky jsou rekonstrukcí Leny + Noise95,503%, byl použit námi navržený algoritmus s přidáváním bodů (viz dále **3.1.2.2 Náš algoritmus**).



Obrázek 3.1.2.A: *Celý poškozený obraz a jeho zvětšený výřez vyznačené oblasti*

a. Zleva a zprava (LeftRight)

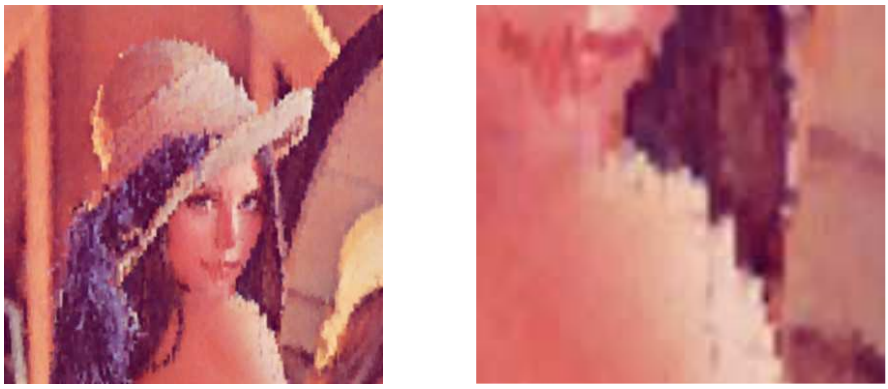
Rekonstrukce probíhá pouze ve směru osy x obrazu a tedy se chyba šíří horizontálně. Algoritmus pracuje tak, že začne nejlevějším pixelem na daném řádku a otestuje ho, zda je poškozený a tedy je-li ho třeba rekonstruovat. Jeho další chování na této řádce závisí na zvoleném algoritmu (Uhlíř či námi navržený). Pokud s daným řádkem v této iteraci skončil, přesune se na řádek následující a proces opakuje.



Obrázek 3.1.2.B: *LeftRight rekonstrukce a jeho zvětšený výřez.*

b. Shora a zdola (TopBottom)

Liší se od předchozího pouze tak, že se rekonstruuje ve směru osy y obrazu a chyba se tedy šíří vertikálně. Metodu lze tedy nahradit použitím metody LeftRight, které na vstup dáme obraz otočený o 90 stupňů. Z tohoto důvodu se jí dále zabývat nebudeme.

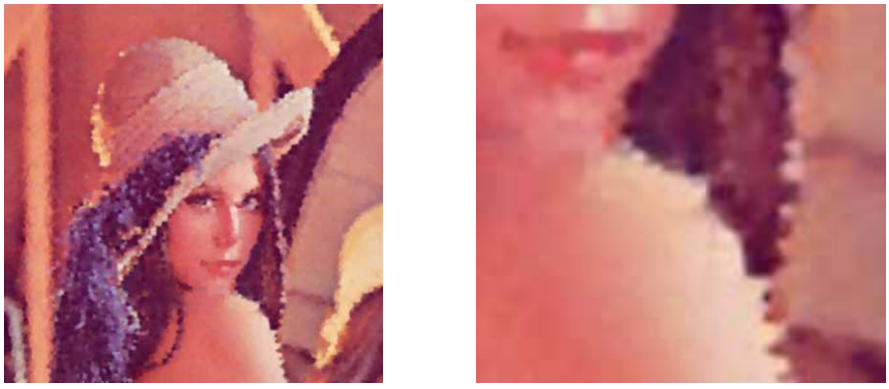


Obrázek 3.1.2.C: *TopBottom rekonstrukce a jeho zvětšený výřez.*

c. Zleva a zprava + Shora a zdola (LeftRightTopBottom)

Metoda nejprve provede LeftRight, poté TopBottom rekonstrukci a až poté přejde k další iteraci (opraví body, které zatím neměly dostatek známých bodů v okolí).

Metoda dává velice dobré výsledky, informace z okolí je do poškozené oblasti šířena z obou os obrazu a tedy se odstraní artefakty objevující se na obou předchozích metodách, kdy je na první pohled zřejmé v jakém směru k rekonstrukci došlo. Chyba se tedy také rozptyluje do okolí v obou osách obrazu a to takovým způsobem, že obraz vypadá nejlépe ze všech popsaných směrů, nicméně poněkud rozmazaně.



Obrázek 3.1.2.D: *LeftRighTopBottomt rekonstrukce a jeho zvětšený výřez.*

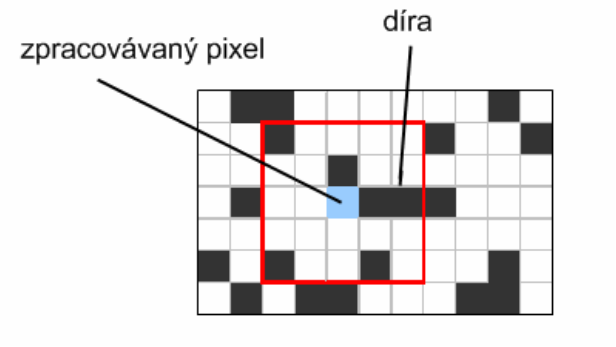
Samozřejmě lze obraz procházet i v jiných kombinacích směrů, tímto jsme si pouze nastínili základní možnosti a jejich vliv na výsledek rekonstrukce.

C. Vyhledávání okolí + jeho případná úprava

Zde se pouze odkážeme na následující kapitoly, které se zabývají těmito problémy. Konkrétně kapitola **3.1.3.3 Dynamické vyhledávání sousedů (DynamicNeighbourhood)** pro popis alternativního způsobu vyhledávání okolí. Pro upravení tohoto okolí a vysvětlení důvodů, které nás k něčemu takovému mohou vést viz kapitola **3.1.3.2 Přidávání bodů (AddWeightedAveragePoints)**.

D. Vyhledávání a přeskakování děr

Další možnou volbou při zpracovávání poškozeného obrazu je určení způsobu, jak se vypořádat s nalezenou dírou. Dírou rozumíme řadu několika poškozených pixelů bezprostředně za sebou, o délce rovné minimálně velikosti poloměru okolí.



Obrázek 3.1.2.E:

Jsou dvě možnosti, jak této problematice čelit:

Přímá rekonstrukce

Každý pixel je okamžitě opraven (výjimkou je situace kdy neznáme dostatek okolních bodů, tzn. méně než počet koeficientů polynomu) a může být hned použit pro následující poškozený pixel (i bezprostředně sousedící) jako známý bod. Metoda má málo iterací, v ideálním případě pouze jednu. V situaci, kdy je iterací potřeba více (popsána výše - nedostatek okolních bodů), odložíme rekonstrukci daného bodu až na další iteraci.

Nevýhodou tohoto přístupu je rychlý vznik, akumulace a následné šíření chyby, protože se pixel opravuje okamžitě, jakmile na něj algoritmus narazí, téměř bez ohledu na stav okolí (opět kromě zmíněné situace nedostatku bodů v okolí). Chyba se vždy resetuje, když algoritmus narazí na platný pixel, takže po zrekonstruované oblasti, kde byla původně díra se náhle objeví pixel s jinou barevnou hodnotou, což vede k nepřirozeným skokům a ostrým změnám v obraze.

Víceprůchodová rekonstrukce – detekce děr

Tato rekonstrukce se při nalezení díry zachová tak, že zrekonstruuje pouze její krajní pixel, zbytek přeskočí a nechá si na další iteraci. Konkrétní chování si rozebereme v následující části u popisu algoritmů. Tato metoda tedy evidentně vede k rekonstrukci probíhající v několika iteracích. Lze očekávat lepší výsledky, neboť nalezené již zrekonstruované pixely v okolí budou mít menší chybu než u přímé rekonstrukce. Tato metoda bude tedy v dalším textu používána, nebude-li řečeno jinak. V následujícím srovnání byl použit námi navržený algoritmus s přidáváním bodů. Všimněme si ostrého skoku u přímé rekonstrukce na spodní hraně poškozené oblasti.

Obrázek 3.1.2.F: *Poškozený obraz, rekonstrukce přímou metodou, rekonstrukce víceprůchodovou metodou.*

3.1.2.1. Algoritmus Ing. Uhlíře

[Uhlir05] Algoritmus je založen na myšlence, že jakmile nalezneme díru a opravíme její první pixel, provedeme příkaz *break*, který ukončí zpracování řádky (pro LeftRight směr zpracovávání). Poté tu samou řádku projdeme od druhého konce, tedy nejčastěji zprava a opět rekonstruujeme do té doby, dokud nenalezneme další díru. Opět opravíme její první krajní pixel (napravo) a provedeme break. Teprve poté se algoritmus přesune na následující řádku a proces opakuje. Metoda postupně od krajů „vyžírá“ poškozené pixely. Její nevýhodou je velký počet iterací.

Uhlířův LeftRight algoritmus zapsaný v pseudokódu:

```
while ((pocet_poskozenych_pixelu > 0) && pixely_lze_jeste_rekonstruovat)
{
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (pixel_je_poskozeny)
            {
                okoli = Najdi_okoli_pixelu(i, j, polomer_okoli);
                bVect = Vytvor_vektor_pravych_stran(okoli);
                A = Vytvor_linearni_system(i, j, okoli);
                lambdaVect = Vyres_linearni_system(A, bVect);
                pixel = Vypocitej_pixel(i, j, okoli, lambdaVect);
                pocet_poskozenych_pixelu--;
                 if (napravo_je_dira) break;
            }
        }
        for (int j = N - 1; j >=0 ; j--)
        {
            if (pixel_je_poskozeny)
            {
                okoli = Najdi_okoli_pixelu(i, j, polomer_okoli);
                bVect = Vytvor_vektor_pravych_stran(okoli);
                A = Vytvor_linearni_system(i, j, okoli);
                lambdaVect = Vyres_linearni_system(A, bVect);
                pixel = Vypocitej_pixel(i, j, okoli, lambdaVect);
                pocet_poskozenych_pixelu--;
                 if (nalevo_je_dira) break;
            }
        }
    }
}
```

Obrázek 3.1.2.1: *Uhlířův algoritmus, průběh LeftRighTopBottom rekonstrukce s přidáváním bodů pro Lena + Noise95,503%. Rekonstrukce po 60., 120. a 180. iteraci.*

3.1.2.2. Náš algoritmus

Nyní si představíme nový vlastní algoritmus zpracování poškozeného obrazu. Algoritmus, na rozdíl od Uhlířova, po nalezení díry a opravě krajního pixelu neukončí zpracování řádky příkazem break, ale opraví i koncový pixel díry (pravý) a pokračuje na řádce ve zpracovávání, dokud nenarazí na její konec. Proces dále opakujeme na následující řádce. Metoda oproti Uhlířovi ve většině případu sníží počet iterací i výpočetní čas a podává lepší výsledky.

Námi navržený algoritmus LeftRight zapsaný v pseudokódu:

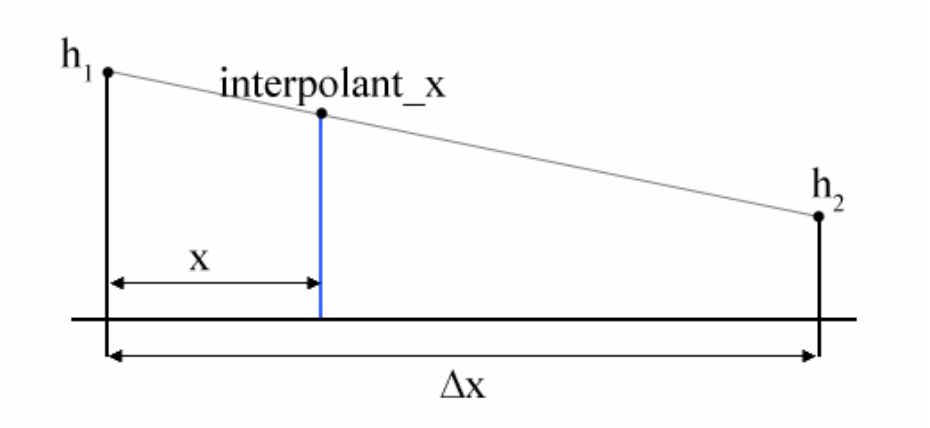
```
while ((pocet_poskozenych_pixelu > 0) && pixely_lze_jeste_rekonstruovat)
{
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
             if (pixel_vpravo_je_NEposkozeny) dira = false;

            if (pixel_je_poskozeny && !dira)
            {
                okoli = Najdi_okoli_pixelu(i, j, polomer_okoli);
                bVect = Vytvor_vektor_pravych_stran(okoli);
                A = Vytvor_linearni_system(i, j, okoli);
                lambdaVect = Vyres_linearni_system(A, bVect);
                pixel = Vypocitej_pixel(i, j, okoli, lambdaVect);
                pocet_poskozenych_pixelu--;
                 if (napravo_je_dira) dira = true;
            }
        }
    }
}
```

Obrázek 3.1.2.2: *Náš algoritmus, průběh LeftRighTopBottom rekonstrukce s přidáváním bodů pro Lena + Noise95,503%. Rekonstrukce po 1., 2. a 4. iteraci.*

3.1.2.3. Bilineární interpolace

Bilineární interpolace je kombinací dvou lineární interpolací, vypočítaných v obou osách obrazu. Použijeme tedy vzorce vycházející z následujícího nákresu:



Obrázek 3.1.2.3.A: *Výpočet interpolantu v ose x*

Nejprve získáme interpolanty v obou osách:

$$interpolant\_x = h_1\left(\frac{\Delta x - x}{\Delta x}\right) + h_2\left(\frac{x}{\Delta x}\right)$$

$$interpolant\_y = g_1\left(\frac{\Delta y - y}{\Delta y}\right) + g_2\left(\frac{y}{\Delta y}\right). \qquad \textbf{(3.1.2.3.1)}$$

Poté váhy těchto interpolantů:

$$w_x = \frac{\min(y, \Delta y - y)}{\min(x, \Delta x - x) + \min(y, \Delta y - y)}$$

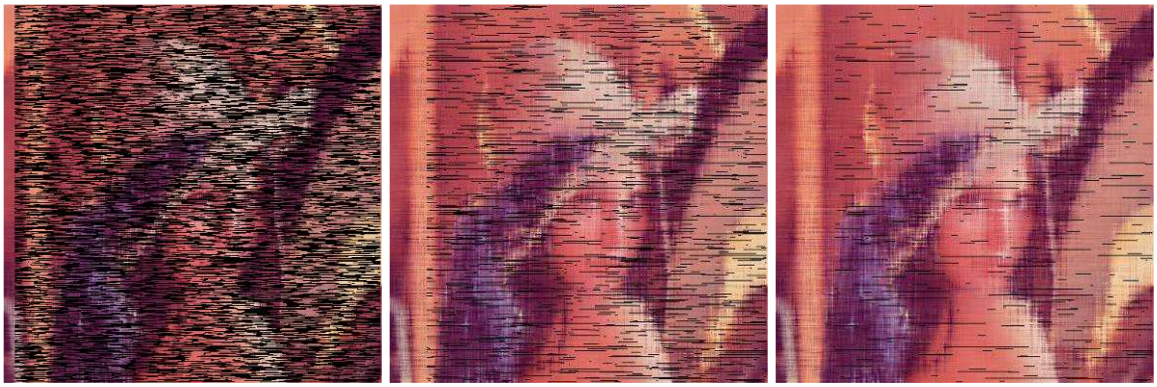
$$w_y = \frac{\min(x, \Delta x - x)}{\min(x, \Delta x - x) + \min(y, \Delta y - y)}. \qquad \textbf{(3.1.2.3.2)}$$

Výsledek je kombinací obou interpolantů:

$$interpolant = interpolant\_x * w_x + interpolant\_y * w_y. \qquad \textbf{(3.1.2.3.3)}$$

Výsledkem této interpolace je vytvoření plynulého přechodu z nalezených okolních známých bodů. Způsob hledání těchto bodů se liší od postupu, používaném pro vyhledávání známých bodů v okolí s definovaným poloměrem. Zde nám stačí nalézt nejbližší levý a pravý pixel pro výpočet interpolantu v ose x, pro výpočet v ose y obdobně vyhledáme nejbližší spodní a horní známý bod. Hodnoty těchto pixelů jsou pak označené ve vzorcích jako h_1, h_2 a g_1, g_2.

Poznámka: pokud se nepodaří nalézt všechny 4 body, provede se lineární interpolace pouze ve směru osy, ve kterém jsme nalezli 2 známé pixely.



Obrázek 3.1.2.3.B: *Bilineární interpolace využívající pouze původní body Lena + Noise95,503%. Rekonstrukce po 15., 50. a 80. iteraci.*

3.1.3. Další navržená řešení a vylepšení

Kromě nalezení nového algoritmu, označovaném v celé práci jako *náš algoritmus* nebo *navržený algoritmus*, bylo navrženo několik dalších postupů, které buď urychlují výpočet, nebo zlepšují vizuální kvalitu interpolace. Tato navržená řešení si ukážeme v této kapitole.

3.1.3.1. Ukládání inverzních matic

Lineární systém pro vyřešení každého pixelu je v obecném případě potřeba vždy pro každý rekonstruovaný pixel znovu sestavit a vyřešit. Tento problém je možné obejít a tím pádem výpočet urychlit.

Připomeňme si, že matice lineárního systému osahuje i souřadnice známých pixelů a pro výpočet hodnoty radiální bázové funkce potřebujeme i jejich vzájemné vzdálenosti. Souřadnice pixelů jsou v obecném případě zadávány v absolutních souřadnicích, můžeme je ovšem zadávat i v relativní vzdálenosti vůči právě rekonstruovanému pixelu.

Základním předpokladem této metody tedy je, že vytvářená matice bude obsahovat relativní souřadnice okolních známých pixelů. Pro určité typy poškození, například poškození opakujícím se pravidelným vzorem, bude takových matic s relativními souřadnicemi jen několik. Pro lineární systém zapsaný

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

lze nalézt řešení ve tvaru

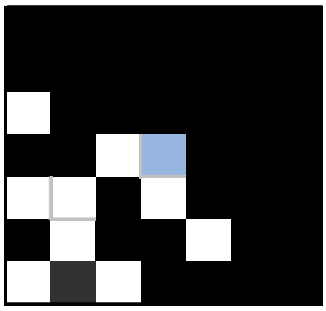
$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}\ .$$

Stačí tedy lineární systém sestavit, vypočítat k němu inverzní matici a tu uložit pro případné použití při rekonstrukci jiného pixelu, který bude mít stejnou matici konfigurace okolí. Již spočítané inverzní matice tedy můžeme ukládat do kolekce typu *Dictionary*, kde každá z nich bude identifikovaná jednoznačným hashem, spočítaným na základě rozložení (konfigurace) známých bodů v okolí poškozeného pixelu. Hash tedy budeme počítat pro každý pixel a použijeme ho pro vyhledání již vypočítané matice v kolekci. Pokud spočítanému hashi v kolekci neodpovídá žádná uložená matice, vytvoříme lineární systém, provedeme jeho inverzi a vložíme ho do kolekce. Pokud jsme matici v kolekci nalezli spočtenou již z předchozí rekonstrukce, získáme koeficienty λ_i přenásobením této matice vektorem pravých stran a dále postupujeme již klasickým způsobem k získání interpolovaného pixelu.

Jak již bylo řečeno, tato metoda může významným způsobem urychlit výpočet pro obrazy poškozené pravidelným vzorem např. rastrem. V takovém případě se lineární systém sestaví a invertuje pouze několikrát pro celý obraz (v řádu desítek či stovek). Nevýhodou je zřejmý nárůst spotřeby paměti pro ukládání matic, je-li konfigurací známých pixelů v okolí pro všechny poškozené body příliš mnoho. V takovém případě je navíc matice počítána často, takže metoda pak ztrácí svou výhodu. Také s rostoucími hodnotami poloměru okolí roste i variabilita rozložení pixelů v okolí a tedy potřeba počítat matici znovu pro téměř každý rekonstruovaný pixel.
Doporučením pro použití metody ukládání inverzních matic tedy je, řešit výpočet touto metodou v případech, kdy je v zadaném okolí každého poškozeného pixelu poškození podobného či dokonce stejného typu jako u ostatních pixelů, či pro poškození opakujícími se vzory a zároveň malým poloměrem okolí (tzn. pro poloměr menší či roven přibližně 3).

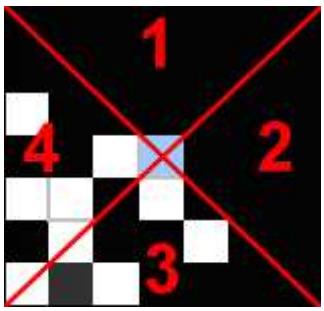
3.1.3.2. Přidávání bodů (AddWeightedAveragePoints)

Ukažme si situaci, kdy máme v okolí rekonstruovaného pixelu dostatek platných hodnot, nicméně ty jsou v okolí rozloženy nežádoucím způsobem:



Obrázek 3.1.3.2.A: *Bílé pixely opět představují známé body*

V uvedeném případě jsme sice při prohledání okolí nalezli dostatečný počet známých bodů a lze tedy provést rekonstrukci, z obrázku je však patrné, že tyto známé body jsou rozloženy velice nerovnoměrně. Z horní a pravé části nemáme téměř žádnou informaci o hodnotách pixelů a spoléháme se na spodní a levou část. V takovýchto případech se ale průběh interpolantu chová nevhodně, jelikož v nedefinovaných oblastech si může dělat „co chce“. Bylo by tedy vhodné ho v těchto oblastech alespoň trochu omezit nějakými dodatečnými podmínkami a zamezit mu, aby např. konvergoval k extrémním hodnotám.
Pokusíme se tedy dodefinovat nějaké další body. Nejprve je třeba rozhodnout, zda je v té které oblasti potřeba tyto body vůbec přidat. Proto celé okolí rozdělíme do 4 kvadrantů:



Obrázek 3.1.3.2.B

Na začátku algoritmu vygenerujeme 4 hashe podobně jako při ukládání inverzních matic, ovšem pro případy, že v daném kvadrantu není žádný známý bod. Takto získáme pro každý

kvadrant jeden hash, podle kterého jsme jednoznačně schopni určit, ve kterém kvadrantu je potřeba dodefinovat body. Nyní při každém prohledávání okolí na známé sousedy budeme generovat hash, jako u metody ukládání inverzních matic, jednoznačně popisující konfiguraci okolí rekonstruovaného pixelu. Tento hash poté porovnáme s přepočítanými hashi pro každý kvadrant, abychom zjistili, zda je potřeba v příslušném kvadrantu body přidávat.
Zjistíme-li kvadrant bez definovaných bodů, provedeme operaci přidání bodu. Přidávaný bod umístíme do větší vzdálenosti než je poloměr okolí. Důvodem je to, že tento bod budeme odhadovat a tím pádem zanášet do obrazu záměrně chybu, proto ho umístíme dále, než jsou hodnoty skutečně nalezené v okolí, aby interpolant příliš neovlivňoval. Určení správné vzdálenosti je prakticky nemožné, pokusy na různých typech poškození a poloměrech okolí jsme stanovili univerzální vzdálenost 10 pixelů v ose příslušného kvadrantu od rekonstruovaného bodu.
Nyní se podívejme na způsob, jak vůbec nalézt pixel, který budeme do této vzdálenosti přidávat. Nejprve se do této vzdálenosti podíváme, abychom zjistili, zda se tam nějaký známý pixel již nenalézá – to proto, abychom si ho úplně nevymýšleli. Pokud tam žádný takový pixel nenalezneme, nezbývá, než ho nějakým způsobem odhadnout.
Pro tento odhad jsme použili vážený průměr všech již známých pixelů z okolí rekonstruovaného bodu. Příspěvek každého takového pixelu (a tedy jeho váha) je dán jeho vzdáleností od rekonstruovaného bodu, která s rostoucí vzdáleností klesá.
Nejprve tedy spočítáme součet vzdáleností všech pixelů:

$$distSum = \sum_{i=0}^{n} Distance(actualPixel, pixel[i]), \qquad \textbf{(3.1.3.2.1)}$$

kde n je počet známých bodů. Výsledný pixel tedy získáme váženým průměrem dle vzorce:

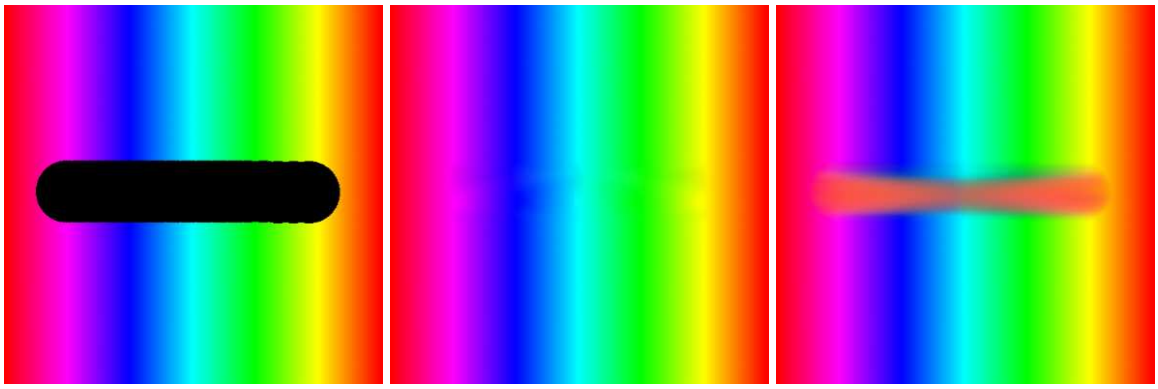
$$actualPixel = \sum_{i=0}^{n} pixel[i] * Distance(actualPixel, pixel[i]) / distSum. \qquad \textbf{(3.1.3.2.2)}$$

Metoda velice úspěšně zlepšuje kvalitu většiny rekonstruovaných obrazů, ovšem v některých situacích rekonstrukci ovlivňuje negativně. Příkladem budiž situace, kdy nepočítáme vážený průměr, protože jsme v daném kvadrantu v oné univerzální vzdálenosti nalezli existující platný bod, který ovšem mezi body nalezené v regulérním okolí z podstaty obrazu nepatří.

Pro následující ukázky byl použit náš algoritmus, lineární polynom i bázová funkce, okolí=2:

Obrázek 3.1.3.2.C: *Lena +Wide, LeftRighTopBottom - poškozený obraz, bez přidávání bodů, s přidáváním bodů*

Obrázek 3.1.3.2.D: *Lena + Noise95,503%, LeftRighTopBottom rekonstrukce bez přidávání a s přidáváním bodů*



Obrázek 3.1.3.2.E: *Gradient+ Wide, LeftRight – poškozený obraz, bez přidávání bodů, s přidáváním bodů (zde naprosto selhává)*

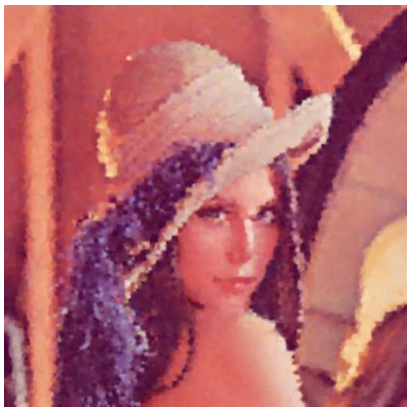
3.1.3.3. Dynamické vyhledávání sousedů (DynamicNeighbourhood)

Alternativní metoda vyhledávání sousedů. Byla inspirována vyhledáváním potřebných bodů u bilineární interpolace. Stejně jako v případě bilineární interpolace, i zde nalezneme pro každý pixel nejbližší známý levý, pravý, horní a dolní pixel. Z těchto 4 nalezených bodů vytvoříme bounding box, který následně prohledáme na platné pixely.

Metoda je časově velice náročná, neboť pro každý pixel je hledáno specifické okolí přes bounding box, ve kterém není nijak výjimečné nalezení stovky či tisíce platných pixelů. Řešení takové soustavy je pak výpočetně velice náročné. Kombinaci této metody společně s metodou ukládání inverzních matic důrazně nedoporučujeme, neboť téměř každý pixel má jiný bounding box. Zároveň ukládání takto velkých matic do kolekce, byť jen pro pár případů, je prakticky nemyslitelné.

Na druhou stranu metoda poskytuje velice dobré výsledky, ovšem cena za ně v podobě času výpočtu je příliš veliká. Proto lze tuto metodu doporučit pouze pro poškození s malým rozsahem. Navíc námi navržený algoritmus společně s metodou AddWeightedAveragePoints dává velmi podobné výsledky za zlomek času.

Níže vidíme porovnání výstupů (Lena + Noise95,503%) našeho algoritmu pro LeftRighTopBottom směr rekonstrukce, s poloměrem okolí = 2. V závorkách jsou uvedeny doby výpočtu na testovacím stroji (uvedeme dále).




Obrázek 3.1.3.3: *Metoda AddWeightedAveragePoints (21,9 s) a metoda DynamicNeighbourhood (601,7 s). Pokud bychom zvolili u přidávání bodů větší poloměr okolí, dosáhneme výsledku velice podobnému metodě s dynamickým vyhledáváním sousedů*

3.1.4. Barevné prostory

Pro metodu bez přidávání bodů či rekonstrukci za pomocí kvadratického polynomu jsme často dostávali výsledky s rušivými barevnými artefakty. Jako původce těchto artefaktů jsme označili interpolaci v RGB prostoru, kdy je každá složka interpolována nezávisle a výsledný obraz získáme teprve smícháním výsledných RGB kanálů, přičemž každý kanál má na výsledek stejný vliv. Pokusili jsme se tedy provést interpolaci i v jiných barevných prostorech, konkrétně CIE L*a*b*, YUV a XYZ.

Prostory CIE L*a*b* a YUV se od RGB a XYZ liší především v tom, že mají oddělenou intenzitu od chromacity. Očekávali jsme tedy menší pravděpodobnost, že při interpolaci kanálů budou jejich hodnoty (a tedy i výsledné barvy) konvergovat k extrémům, jako v RGB. Výsledky však byly překvapivě podobné interpolaci v RGB prostoru (Lena + Wide).


Obrázek 3.1.4.A: *CIE L*a*b** *YUV*


RGB *XYZ*

Zopakovali jsme experiment i na šedotónovém obraze, ovšem se stejným výsledkem:

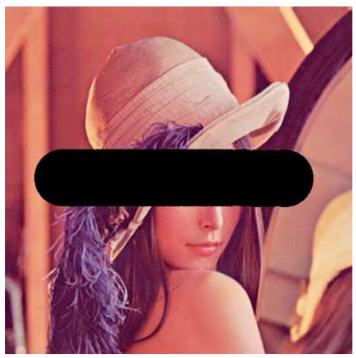
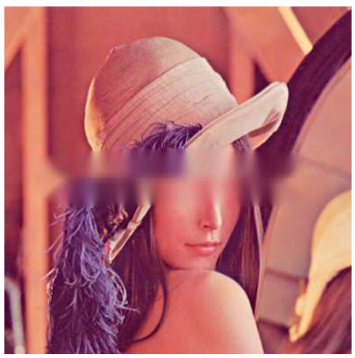

Obrázek 3.1.4.B: *CIE L*a*b** *YUV*


RGB *XYZ*

Z experimentu vidíme, že vliv barevného prostoru (některého námi testovaného) na výsledek rekonstrukce je minimální a k barevným artefaktům stále dochází. Tyto artefakty se vyskytují u rozsáhlého poškození, bez ohledu na použitý typ rozšiřujícího elementu. Jedním ze způsobů, jak je potlačit, je přidáváním bodů (AddWeightedAveragePoints), nicméně ani tehdy nemáme úspěch vždy zaručen, zejména je-li použit kvadratický polynom.

3.1.5. Metody porovnání výsledků

Pro porovnávání obrazů můžeme vytvářet rozdílové obrazy originálu a rekonstrukce, kde na výsledném obraze vidíme rozdíly mezi porovnávanými obrazy, které mohou být různými metodami zvýrazněné. Jinou možností je použití nějaké zavedené metriky. Pro většinu z nich je velkou nevýhodou strojové zpracování, kdy obrazy identifikované počítačem jako rozdílné (např. kvůli nepatrnému barevnému posunu) jsou lidským okem vyhodnoceny jako totožné. Výsledky níže uvedených metod porovnávání odpovídají těmto obrazům (levý a pravý):

Obrázek 3.1.5: *Originální obraz* *Lena + Wide* *Rekonstrukce (náš algoritmus, LeftRightTopBottom + přidávání)*

3.1.5.1. MSE

Základní metrikou je *střední kvadratická chyba (mean square error)* neboli *MSE*. Jedná se o rozdíl obrazových hodnot, umocněný pro zvýraznění chyby. Součet těchto hodnot pro každý pixel obrazu, vydělený počtem pixelů nám dá výslednou hodnotu MSE, tzn. průměrnou chybu každého pixelu. Hodnotu počítáme pro každý RGB kanál zvlášť a také pro Grayscale obraz.

$$MSE = \frac{1}{MxN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (original[i, j] - reconstructed[i, j])^2 , \qquad \textbf{(3.1.5.1.1)}$$

kde *original* je originální obraz, *reconstructed* je rekonstruovaný obraz, i a j jsou indexy tj. souřadnice pixelu v obraze, M je výška obrazu a nakonec N je šířka obrazu.

Platí, že čím menší hodnota MSE, tím menší chyby jsme se dopustili a tedy tím lépe. V našem případě jsme dosáhli hodnot:

MSE (R kanál) = 165,6995
MSE (G kanál) = 212,0171
MSE (B kanál) = 104,7286
MSE ((R+G+B)/3) = 160,8151
MSE (Grayscale) = 173,3145

3.1.5.2. Vizualizace MSE

MSE hodnotu lze také zobrazit jako jasovou hodnotu v každém pixelu a tedy jednoduše zjistit, kde se originální a rekonstruovaný obraz liší. V našem případě jsme vykreslovali neumocněný rozdíl originálního a rekonstruovaného obrazu, tedy:

$$pixel[i, j] = Abs(original[i, j] - reconstructed[i, j]) . \qquad \textbf{(3.1.5.2.1)}$$

Nulový rozdíl, tedy místo, kde je originál i rekonstruovaný obraz shodný, má tedy zřejmě hodnotu 0, což vyjádřeno jako jas pixelu, znamená černý pixel. Čím je pixel světlejší, tím větší chyby jsme se rekonstrukcí dopustili. Tuto hodnotu můžeme také invertovat tak, že ji odečteme od 255 (maximální jasové hodnoty) pro jiný způsob vykreslení. V takovém případě zřejmě platí, že bílé pixely znamenají stejný pixel u originálu i rekonstruovaného obrazu a čím je pixel tmavší, k tím větší chybě při rekonstrukci došlo.

Obrázek 3.1.5.2: *Vizualizace MSE* *Invertovaná vizualizace MSE*

3.1.5.3. Vizualizace MSE zvýrazněná logaritmickým operátorem

Postup používaný pro HDR. Dynamický rozsah obrazu může být snížen tak, že každou hodnotu pixelu nahradíme jejím logaritmem, což způsobí zvýraznění pixelů s nízkou intenzitou. Použitím logaritmického operátoru tedy umožníme zobrazení informací, jejichž dynamický rozsah je příliš velký pro klasické vykreslení. Logaritmický operátor mapuje hodnoty na logaritmickou křivku, neboli každá hodnota pixelu je nahrazena jejím (nejčastěji desítkovým) logaritmem. Logaritmický operátor (resp. jeho mapovací funkce) je dán jako:

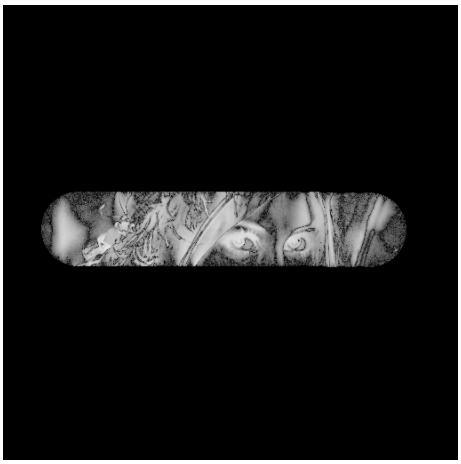
$$pixel[i, j] = c \cdot log_{10}(1 + Abs(original[i, j] - reconstructed[i, j])), \qquad (\textbf{3.1.5.3.1})$$

kde

$$c = \frac{255}{log_{10}(1+255)}$$

je konstanta zvolená tak, aby maximální výsledná hodnota byla 255.
Další informace o logaritmickém operátoru viz [IPLR_WWW].

Jak je vidět na následujících obrázcích, logaritmický operátor zvýrazní oblasti, které se v originálním a zrekonstruovaném obraze liší tak, že je na první pohled patrné, kde je rozdíl. A to dokonce i v oblastech, ve kterých bychom to jinak pouhým okem nepostřehli.



Obrázek 3.1.5.3: *Vizualizace MSE vylepšená logaritmickým operátorem*

Invertovaná vizualizace MSE vylepšená logaritmickým operátorem

3.1.5.4. Chybový obraz

Zobrazuje chybu obrazových bodů podobně jako vizualizace MSE. Vizualizace MSE není ve všech oblastech dostatečně výrazná a lidské oko nepostřehne rozdíl v místech, kde sice k chybě došlo, ale jasová hodnota je blízká 0 (resp. 255 pro invertovaný případ). Proto lze tuto chybu zvýraznit následujícím vzorcem:

$$pixel[i, j] = k \cdot Abs(\, original[i, j] - reconstructed[i, j]) + 128, \qquad (\textbf{3.1.5.4.1})$$

kde k je konstanta, zvýrazňující chybu (my jsme použili $k = 2$). Konstanta 128 pak celý obraz posune do šedé barvy.

Nevýhodou tohoto způsobu zobrazení je, že pro větší chyby dojde velice snadno k přetečení zobrazované hodnoty. Takový pixel pak skokem změní hodnotu z bílé na černou, viz obrázek.

Obrázek 3.1.5.4: *Chybový obraz*

3.1.5.5. PSNR

[Berkeley_WWW] Zkratka slov *peak signal to noise ratio*, jedná se o hodnotu, vyjadřující kvalitu rekonstrukce pro celý obraz. Tato porovnávací metoda je často používána při kódování videa a kompresi obrazů. Vyjadřuje poměr mezi maximální možnou silou signálu a šumu. Protože mnoho signálů má velmi široký dynamický rozsah, je většinou PSNR škálována logaritmicky. Pro její výpočet potřebujeme vypočítat MSE, viz vzorec:

$$PSNR = 20 \cdot log_{10}\left(\frac{255}{\sqrt{MSE}}\right). \qquad \textbf{(3.1.5.5.1)}$$

Hodnota udávaná v decibelech na dvě desetinná místa, typické jsou pak hodnoty mezi 20 a 40 dB, přičemž hodnoty kolem 30dB jsou považovány za kvalitní a hodnoty nad 50 dB jsou pak lidským okem nepostřehnutelné. Zde tedy platí, že čím vyšší hodnota PSNR, tím lépe. Pro naše obrazy získáme:

PSNR (R kanál) = 25,9376 dB
PSNR (G kanál) = 24,8671 dB
PSNR (B kanál) = 27,9301 dB
PSNR ((R+G+B)/3) = 26,2449 dB
PSNR (Grayscale) = 25,7425 dB

3.1.5.6. SSIM (Structural SIMilarity) index

Pro příklad strojového porovnání obrazů, snažící se posuzovat kvalitu rekonstrukce podobně jako lidské oko, uvádíme tuto metodu. Bližší informace lze nalézt v [Wang04].
Je to metoda k hledání podobnosti obrazů, která může být také použita k posuzování jejich kvality, kdy jeden z dvojice porovnávaných obrazů považujeme za obraz perfektní kvality. Metoda vychází z předpokladu, že lidský vizuální systém je silně uzpůsoben na získávání strukturálních informací ze scény. Porovnává lokální vzory intenzit pixelů s normovaným jasem a kontrastem. Index nabývá hodnot 0 až 1, kdy 1 znamená totožné obrazy.
Pro naši dvojici testovaných obrazů jsme dostali SSIM index = 0,929694.

Obrázek 3.1.5.6: *Vizualizace SSIM*

3.1.6. Dosažené výsledky

V této kapitole se zaměříme na vliv různých nastavení rekonstrukce na výslednou interpolaci, předvedeme dosažené výsledky a provedeme srovnání měřených veličin. Sledujeme především hodnoty MSE, PSNR a dobu výpočtu, ovšem provedeme i srovnání výsledků z hlediska lidského vnímání.
Veškeré výsledky rekonstrukcí (v originálních velikostech), včetně vizualizací MSE (i varianty s logaritmickým operátorem) a chybových obrazů, jsou pro zájemce k dispozici na CD, přiloženém k této diplomové práci.

V každé podkapitole ukážeme typický výsledek rekonstrukce a jeho měření, pro podrobnější výsledky odkazujeme čtenáře k nahlédnutí do **Příloha D** (str.74). Pro tuto přílohu platí, že:

- v tabulkách zvýrazníme zeleně (v tištěné verzi tmavší šeď) nejlepší sledovaný dosažený výsledek, oranžově (světle šedá) pak nejhorší
- za tabulkou následují rekonstrukce samotné, vždy v takovém pořadí sledovaných parametrů, v jakém jsou zapsány v řádkách tabulky
- zkratka GS znamená grayscale, tedy šedotónový obraz

V případech, kde jsou rozdíly mezi rekonstrukcemi téměř neznatelné či malé a okem nepostřehnutelné (vzhledem k velikosti obrázků, které zde prezentujeme), pro úsporu místa ukážeme pouze jeden (první) z nich. Pokud si jsou rekonstrukce a originální obraz (v prezentovaných velikostech) natolik podobné, že rozdíly nelze okem zaznamenat, neuvádíme výsledné obrazy pro úsporu místa vůbec (uvádíme pouze tabulky s naměřenými hodnotami).

Všechna měření byla provedena na konfiguraci AMD Athlon 3200+, 2GB RAM na operačním systému Windows XP Professional SP2.

3.1.6.1. Barevné systémy

Jak jsme již uvedli v kapitole **3.1.4 Barevné prostory**, vliv některého z testovaných systémů na výsledek rekonstrukce je minimální.

Nastavení testu:

Vstup:	Lena + Wide
Algoritmus:	Námi navržený
Směr:	LeftRightTopBottom
Detekce děr:	Ano
Výpočet:	LU rozklad
Výběr okolí:	AddWeightedAveragePoints
Polynom:	Kvadratický
RBF:	Lineární
Poloměr okolí:	2

Barevný prostor	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	
L*a*b*	355,62	348,52	228,91	311,02	286,23	22,62	22,71	24,53	23,29	23,56	4 236
RGB	330,79	350,92	248,28	310,00	276,83	22,94	22,68	24,18	23,27	23,71	4 286
XYZ	409,00	352,28	271,20	344,16	288,14	22,01	22,66	23,80	22,82	23,53	4 299
YUV	328,00	354,93	264,87	315,93	280,19	22,97	22,63	23,90	23,17	23,66	4 231

Tabulka 3.1.6.1

Ukázky viz Kapitola **3.1.4 Barevné prostory**.
Jako nejhorší se ukázala interpolace v XYZ systému, zatímco ostatní systémy jsou každý v něčem nejlepší. Rozdíly jsou ovšem natolik malé, že je lze prohlásit za nepodstatné.

3.1.6.2. Získávání okolních bodů a vliv poloměru okolí

Tuto podkapitolu rozdělíme na dvě části

- získávání okolních bodů
- volba poloměru okolí

3.1.6.2.1. Získávání okolních bodů

Zde se zaměříme na způsoby, jak z okolí vyhledávat známé body a jejich vliv na rekonstrukci. V předchozích kapitolách jsme si popsali tyto možnosti:

NAP = NoAddingPoints (základní postup – bez přidávání bodů)
AWAP = AddWeightedAveragePoints (s přidáváním bodů váženým průměrem)
DN = DynamicNeighbourhood (dynamické vyhledávání sousedů)

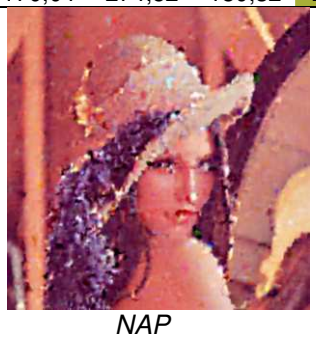
Nastavení testu:

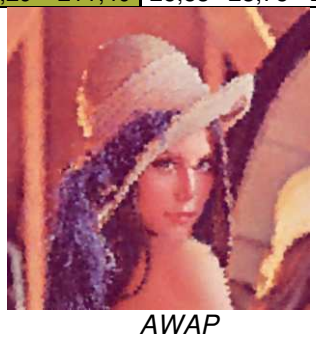
Algoritmus:	Námi navržený
Detekce děr:	Ano
Výpočet:	LU rozklad
Polynom:	Lineární
RBF:	Lineární
Poloměr okolí:	2

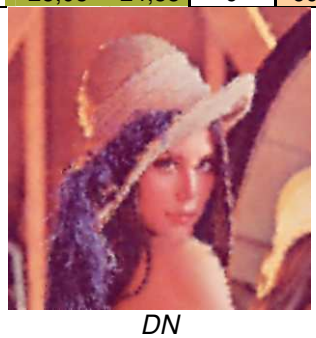
Z tabulek vidíme, že pro poškození extrémním šumem dává nejlepší výsledky použití dynamického vyhledávání sousedů. Ovšem při pohledu na dobu výpočtu lze za nejvhodnější prohlásit použití metody přidávání bodů, která dává lepší výsledky než obyčejná metoda bez přidávání bodů, za téměř stejný čas.
V případě rozsáhlého souvislého poškození je situace podobná – podle očekávání trvá nejdéle dynamické vyhledávání sousedů, ale už nedává lepší výsledky než zbylé dvě metody. Také v tomto případě doporučujeme metodu s přidáváním bodů.
Zároveň vidíme, že se vyplatí použít LeftRightTopBottom směr zpracovávání, nejen kvůli lepším (tabulkovým i vizuálním) výsledkům ale i kratším časům výpočtu.

Tabulka 3.1.6.2.1.B: Vstup: Lena + Noise95,503%, Směr: LeftRighTopBottom

Typ	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
NAP	395,99	592,24	483,39	490,54	453,23	22,15	20,41	21,29	21,28	21,57	14	21 844
AWAP	194,95	315,65	203,45	238,02	242,64	25,23	23,14	25,05	24,47	24,28	9	21 890
DN	170,04	274,52	180,32	208,29	211,40	25,83	23,75	25,57	25,05	24,88	9	601 735


NAP


AWAP


DN

3.1.6.2.2. Volba poloměru okolí

Zde se podíváme na to, jaké výsledky dostaneme pro různě nastavené poloměry okolí a tentokráte i pro různé vstupní obrazy.

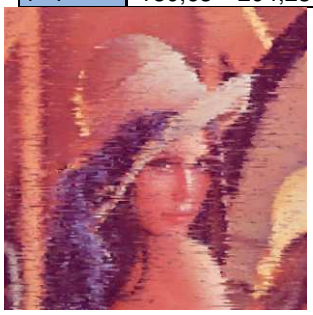
Nastavení testu:

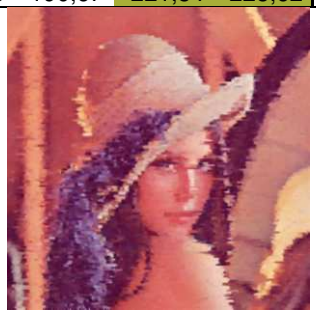
Algoritmus:	Námi navržený
Detekce děr:	Ano
Výpočet:	LU rozklad
Výběr okolí:	AddWeightedAveragePoints
Polynom:	Lineární
RBF:	Lineární

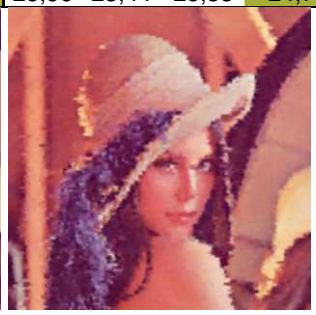
Výsledky jsou zde jednoznačné a odpovídají očekáváním – vyšším poloměrem dosáhneme lepšího výsledku (vizuálního i tabulkového), ovšem za delší dobu. Proto lze jako nejvhodnější velikost poloměru okolí (kompromis mezi časem a kvalitou) doporučit 3.

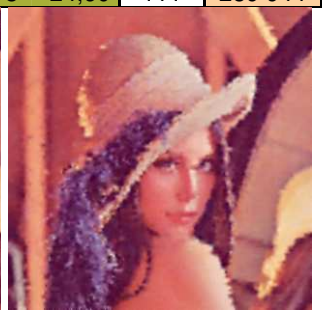
Tabulka 3.1.6.2.2.AA: Vstup: Lena + Noise95,503%, Směr: LeftRight

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	423,15	592,59	311,15	442,30	471,91	21,87	20,40	23,20	21,82	21,39	112	8 195
r=2	261,97	402,74	237,54	300,75	313,72	23,95	22,08	24,37	23,47	23,17	112	25 425
r=3	206,61	330,52	204,81	247,31	254,59	24,98	22,94	25,02	24,31	24,07	111	96 560
r=4	180,63	294,23	190,67	221,84	225,62	25,56	23,44	25,33	24,78	24,60	111	289 944


r=1


r=2


r=3


r=4

3.1.6.3. Volba radiální bázové funkce

Zde se podáváme na to, jak může změna použité bázové funkce ovlivnit výsledek rekonstrukce.

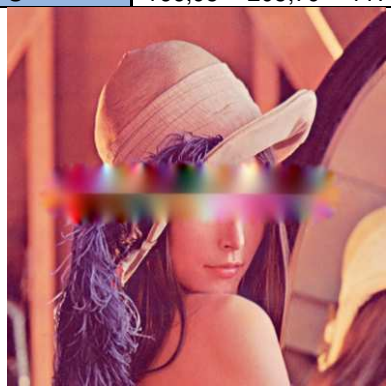
Nastavení testu:

Algoritmus:	Námi navržený
Směr:	LeftRighTopBottom
Detekce děr:	Ano
Výpočet:	LU rozklad
Polynom:	Lineární
Poloměr okolí:	3
C (epsilon):	1

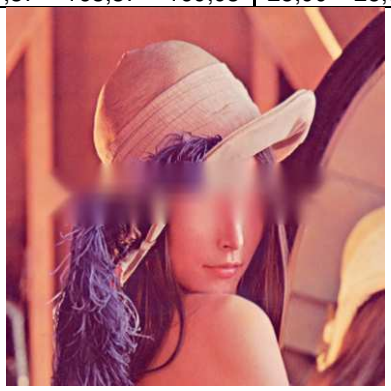
Z výsledků lze vyřadit kubickou funkci, která dává z použitých funkcí nejhorší vizuální výsledky a tabulky také hovoří v její neprospěch. Ze zbývajících funkcí bychom tedy doporučili lineární, jelikož dává kvalitní výsledky za (většinou) nejkratší dobu. Zároveň lze z přiložených ukázek (viz str.74) snadno vidět, jak pomáhá přidávání bodů.

Tabulka 3.1.6.3.I: Vstup: Lena + Wide, Výběr okolí: AddWeightedAveragePoints

Bázová funkce	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Cubic	257,46	314,54	286,72	286,24	239,77	24,02	23,15	23,56	23,58	24,33	9 328
Gauss	172,29	219,82	106,50	166,20	179,79	25,77	24,71	27,86	26,11	25,58	9 746
Linear	164,11	206,20	102,73	157,68	169,45	25,98	24,99	28,01	26,33	25,84	8 924
Multiquadric	168,09	209,10	107,21	161,47	172,55	25,88	24,93	27,83	26,21	25,76	9 514
TPS	166,95	205,79	117,37	163,37	169,93	25,90	25,00	27,44	26,11	25,83	10 084

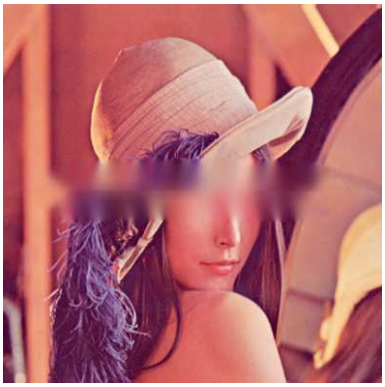


Cubic

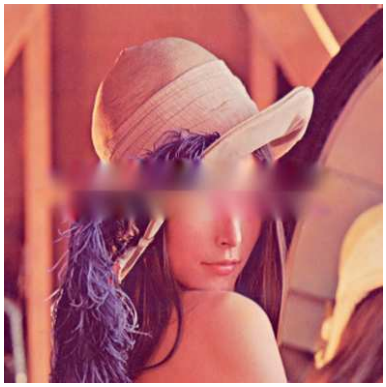


Gauss

Linear



Multiquadric



TPS

3.1.6.4. Vliv rozšiřujícího elementu

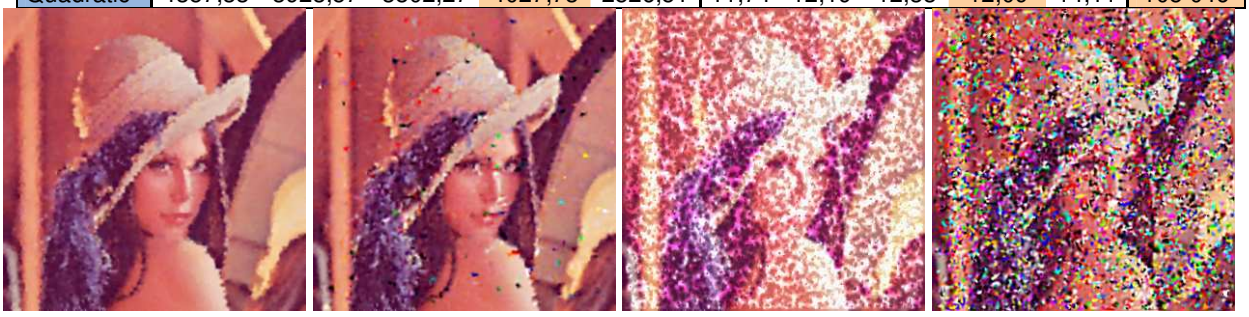
Rozšiřující element má na rekonstrukci zásadní vliv, zejména proto, že některé bázové funkce jej vyžadují. Podívejme se tedy, jakých výsledků dosáhneme s různými elementy. Nastavení:

Algoritmus: Námi navržený
Směr: LeftRighTopBottom
Detekce děr: Ano
Výpočet: LU rozklad
RBF: Lineární
Poloměr okolí: 3

Z výsledků opět plyne několik závěrů. Prvním z nich je, že nejlepším rozšiřujícím elementem lineárního systému není ani jeden polynom, ale konstanta. Lineární polynom dává také kvalitní výsledky, ovšem kvadratický je pro rozsáhlejší poškození nepoužitelný, jelikož vzniklá chyba díky umocňování velice rychle roste do extrémních hodnot. Zároveň vidíme, že pokud systém nerozšíříme nijak, k dobrému výsledku se (dle zvolené RBF) nedostaneme. A opět jsme si potvrdili (nejlépe ze všech provedených měření), jak přidávání bodů zachraňuje rekonstrukci i ve velice komplikovaných případech.

Tabulka 3.1.6.4.H: Vstup: Lena + Noise95,503%, Výběr okolí: NoAddingPoints

Rozšiřující element	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Constant	180,38	294,84	193,74	222,99	226,07	25,57	23,43	25,26	24,75	24,59	73 422
Linear	352,42	447,09	364,48	388,00	345,02	22,66	21,63	22,51	22,27	22,75	84 266
None	2262,78	3708,99	4481,31	3484,36	3022,50	14,58	12,44	11,62	12,88	13,33	70 232
Quadratic	4357,33	3923,57	3802,27	4027,73	2526,51	11,74	12,19	12,33	12,09	14,11	105 046



Constant *Linear* *None* *Quadratic*

3.1.6.5. Volba algoritmu

Jako poslední měření jsme provedli srovnání s Uhlířovým algoritmem a bilineární interpolací (= Lerp, která využívá pouze originální známé body) pro tato nastavení:

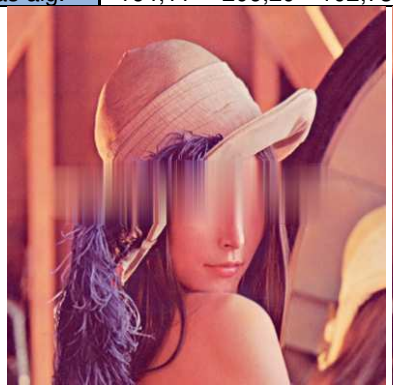
Směr: LeftRighTopBottom
Detekce děr: Ano
Výpočet: LU rozklad
Výběr okolí: AddWeightedAveragePoints(námi navržený), NoAddingPoints (Uhlíř)
RBF: Lineární
Polynom: Lineární
Poloměr okolí: 3

U Uhlířova algoritmu záměrně nepřidáváme body, abychom si udělali představu, jakých výsledků lze pouze na základě jeho práce dosáhnout.

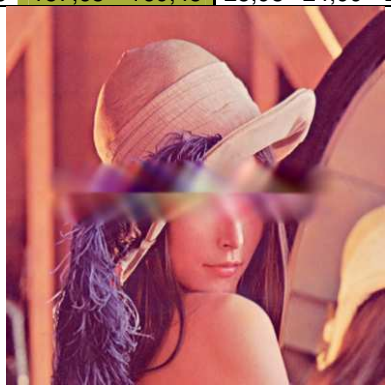
Z výsledků vidíme, že lineární interpolace je sice nejrychlejší pro většinu případů, ale RBF interpolace podává lepší výsledky. Uhlířův algoritmus je s naším v určitých případech téměř totožný, nicméně v situacích s rozsáhlým poškozením, oproti našemu algoritmu selhává.

Tabulka 3.1.6.5.D: Vstup: Lena + Wide

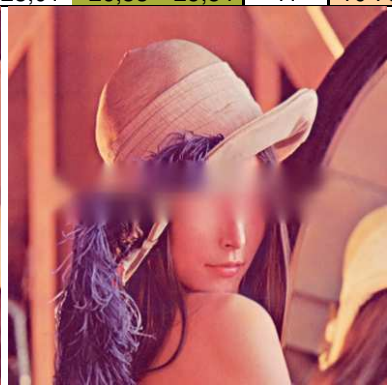
Algoritmus	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS		
Lerp	186,70	247,50	118,25	184,15	200,59	25,42	24,20	27,40	25,67	25,11	410	2 230
Uhlíř	166,15	253,39	161,60	193,71	193,68	25,93	24,09	26,05	25,35	25,26	41	9 435
Náš alg.	164,11	206,20	102,73	157,68	169,45	25,98	24,99	28,01	26,33	25,84	41	10 782



Bilineární interpolace



Uhlíř



Náš algoritmus

3.2. Hledání isočar a isoploch

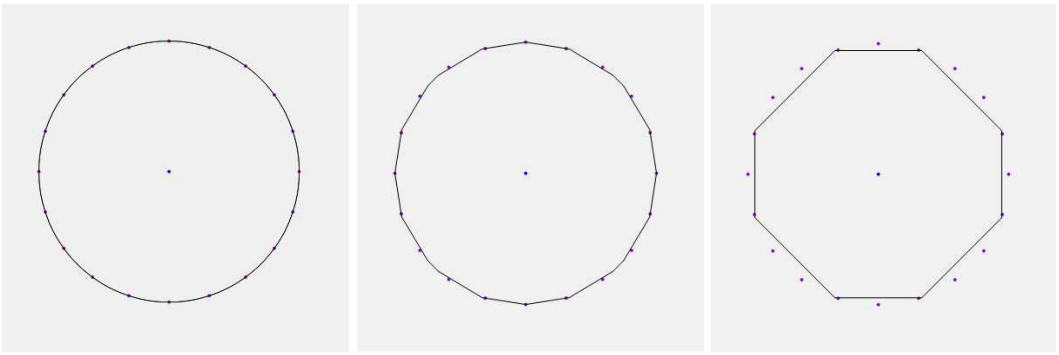
Na začátek si připomeňme co je to isočára, resp. isoplocha. Isočára je křivka, spojující místa stejných hodnot dané veličiny (nadmořská výška, tlak, teplota, rychlost, hustota, funkční hodnota…). Isoplocha je pak ve 3D analogicky povrchem, který prochází body se stejnými hodnotami dané veličiny v prostoru. Problému rekonstrukce povrchu z bodů jsme se již věnovali v kapitole **2.6 RBF interpolace v praxi.**

3.2.1. Isočáry (2D)

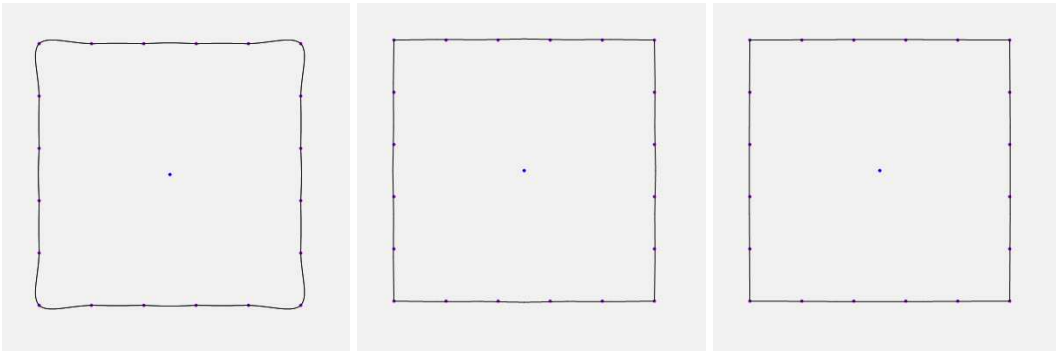
Pro zjednodušení problematiky bylo nejprve započato s extrakcí isočar, tzn. 2D problém. Ve 3D lze očekávat obdobné chování, rozšířené o jednu dimenzi.

Pro rekonstrukci základních primitiv byla vygenerována data tak, že jsme si nasamplovali body např. na obvodu kružnice či čtverce v určitých odstupech. Následně byla tato data použita pro vytvoření lineárního systému. Jako off-surface bod jsme vzali odhad těžiště samplovaných dat (neboli aritmetický průměr souřadnic samplovaných bodů). Tento bod nám pro základní konvexní primitiva stačí pro definování orientace.
Poté jsme zvolili rozlišení rastru (v pixelech), v jakém chceme výsledná data získat. Tím jsme tedy definovali rastr, v jehož vrcholech potřebujeme spočítat hodnoty. Před výpočtem těchto interpolovaných hodnot ještě nalezneme bounding box objektu, který zvětšíme na všech stranách o velikost rastru (isočáry získané pomocí RBF jsou křivky, které se nám do bounding boxu nemusí vejít a mohou ho přesahovat, proto jej zvětšujeme) a teprve poté vytvoříme rastr definované jemnosti uvnitř tohoto bounding boxu. Nyní již můžeme spočítat hodnoty ve vrcholech rastru interpolací pomocí RBF. Pro zobrazení výsledku jsme použili algoritmus Marching Squares, kterým jsme nalezli segmenty isočáry v jednotlivých políčkách na základě interpolace hodnot napočítaných ve vrcholech rastru.

Ukázky rekonstrukce (body na okraji značí vstupní známé body, uprostřed vidíme nalezený odhad středu) pro různě definované velikosti rastru (lineární polynom, lineární bázová funkce + přidání aproximace středu):

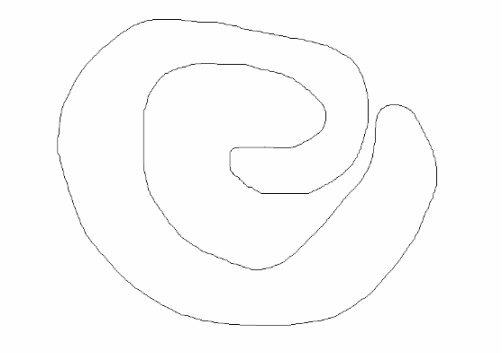


Obrázek 3.2.1.A *Rastr 1 pixel* *Rastr 50 pixelů* *Rastr 100 pixelů*

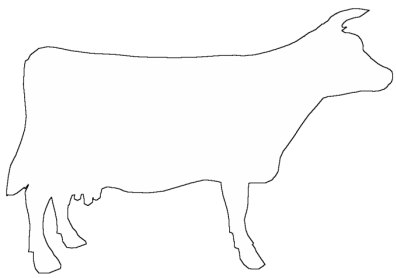


Obrázek 3.2.1.B

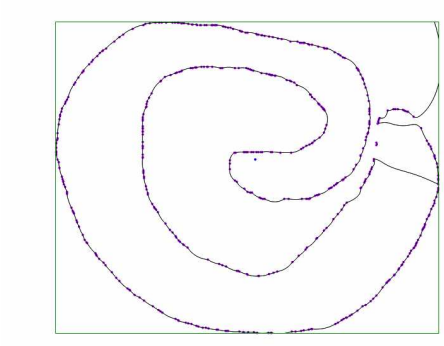
Použili jsme také složitější data, získaná náhodným samplováním těchto obrázků:



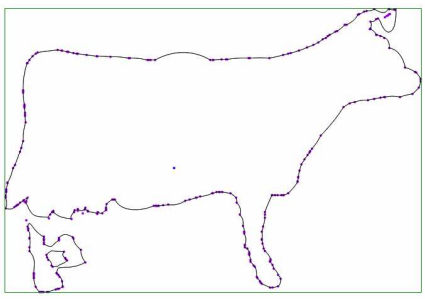
Obrázek 3.2.1.C



Obrázek 3.2.1.D

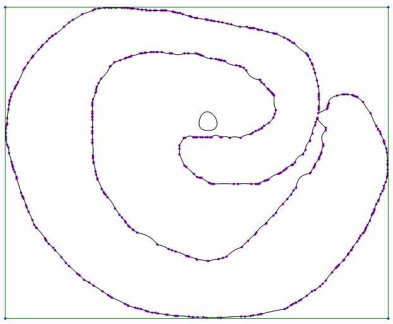


Obrázek 3.2.1.E

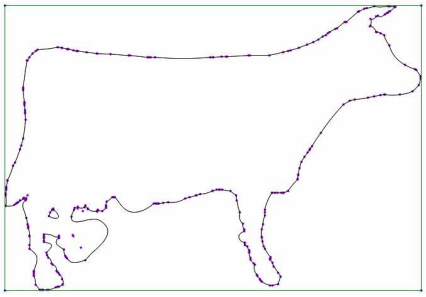


Obrázek 3.2.1.F

Pro rekonstrukci bylo použito lineárního polynomu, lineární bázové funkce a přidání aproximace středu (rastr 1 pixel), výpočet byl proveden LU rozkladem. Kolem množiny bodů vidíme také bounding box, mimo který se již isočára nehledá.



Obrázek 3.2.1.G



Obrázek 3.2.1.H

Na obrázku výše vidíme případ, kdy máme off-surface body generovány v rozích bounding boxu. Pro výpočet byl použit opět lineární polynom a lineární bázová funkce (rastr 1 pixel). Artefakty jsou zaviněné nedostatečným množstvím těchto off-surface bodů a jejich pozicí, která pro takto složitá data nestačí. S obdobným problémem se setkáme i v další kapitole, zabývající se extrakcí isoploch.

Provedli jsme také následující pokus. Místo generování off-surface bodů, které nám do lineárního systému vnášejí nenulové hodnoty a udávají orientaci plochy, jsme upravili lineární systém tak, že jsme ho rozšířili o jednu řádku, stejně tak i jeho pravou stranu:

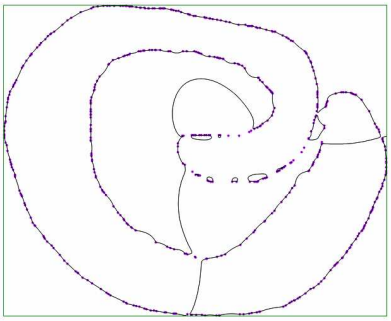
$$\left[\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{P} \\ \hline \boldsymbol{P}^T & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{1} \end{array}\right]\left[\begin{array}{c} \boldsymbol{\lambda} \\ \hline \boldsymbol{\gamma} \end{array}\right]=\left[\begin{array}{c} \boldsymbol{0} \\ \hline \boldsymbol{0} \\ \hline \boldsymbol{1} \end{array}\right]$$

Tím jsme nadefinovali podmínku, že součet všech příspěvků polynomu je roven 1. Abychom však mohli tento systém vyřešit, potřebujeme získat čtvercovou matici (nyní máme matici, která má o jednu řádku víc než sloupců). Provedeme tedy operaci přezásobení matice její transpozicí, čímž získáme požadovanou čtvercovou matici. Musíme však přenásobit i pravou stranu rovnice:

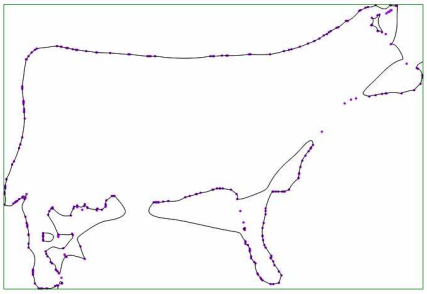
$$\left[\begin{array}{c|c|c} \boldsymbol{A} & \boldsymbol{P}^T & \boldsymbol{0} \\ \hline \boldsymbol{P} & \boldsymbol{0} & \boldsymbol{1} \end{array}\right]\left[\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{P} \\ \hline \boldsymbol{P}^T & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{1} \end{array}\right]\left[\begin{array}{c} \boldsymbol{\lambda} \\ \hline \boldsymbol{\gamma} \end{array}\right]=\left[\begin{array}{c|c|c} \boldsymbol{A} & \boldsymbol{P}^T & \boldsymbol{0} \\ \hline \boldsymbol{P} & \boldsymbol{0} & \boldsymbol{1} \end{array}\right]\left[\begin{array}{c} \boldsymbol{0} \\ \hline \boldsymbol{0} \\ \hline \boldsymbol{1} \end{array}\right]$$

Tento systém je tedy již řešitelný. Výsledky, které získáme, pokud hledáme isočáry touto metodou (pro lineární polynom, lineární bázovou funkci, rastr 1 pixel) viz obrázek ***3.2.1.I*** a ***3.2.1.J.***

Výsledky často trpí artefakty, jejichž příčinou je přidaná podmínka, která prokládanou rovinu deformuje nežádoucím způsobem.



Obrázek 3.2.1.I



Obrázek 3.2.1.J

Pro kvalitní extrakci isočar je potřeba použití nějakého propracovaného algoritmu generování off-surface bodů, jelikož ani jeden z námi testovaných nedokázal pro složitější data nalézt kvalitní isočáry.

Pojďme se nyní podívat na obdobný problém v prostoru rozšířeném o další rozměr – 3D, který se jeví jako zajímavější a ve kterém se budeme potýkat s prakticky stejnými problémy.

3.2.2. Isoplochy (3D)

Pro situaci v 3D jsme vycházeli z analogického postupu. V tomto případě jsme jako vstupní data použili modely (případně jejich alternativy po převodu do TRI formátu) získané z [BioMesh_WWW], [SGI] a [MT_WWW].

Z načteného TRI modelu jsme použili pouze seznam vrcholů modelu. Z těchto hodnot jsme nejprve nalezli bounding box modelu, který je z výše popsaných důvodů potřeba zvětšit. Poté je třeba do seznamu bodů přidat off-surface body. Zde jsme opět získali aproximaci těžiště modelu (neboli aritmetický průměr souřadnic vrcholů). Druhým způsobem je přidání off-surface bodů do vrcholů rozšířeného bounding boxu. Ve druhém případě jsme tedy přidali celkem 8 bodů, zatímco v prvním pouze jeden, u kterého navíc nemáme zaručeno, že skutečně leží uvnitř modelu. Dalším krokem je již sestavení a vyřešení lineárního systému pro získání vah λ_i (eventuelně γ_i). Pokud bychom se pokusili tento systém řešit LU rozkladem, velice brzy bychom narazili na problém s nedostatkem paměti pro jeho uložení. Používaná data totiž obsahují počty bodů, které jsou v řádech stovek, tisíců a výše. Pro řešení systému jsme tedy zvolili GMRES – ten uložení lineárního systému v paměti nevyžaduje.
Nyní potřebujeme, podobně jako při hledání isočar ve 2D případě, vypočítat hodnoty ve vrcholech 3D rastru, abychom posléze mohli např. pomocí algoritmu Marching Cubes (viz [Lorensen87], [Patera02]) zobrazit nalezený povrch. Zde se naskýtá otázka, jak prostor rozdělit, Vyřešili jsme to tím způsobem, že jsme nalezli nejdelší hranu bounding boxu a tu jsme rozdělili na zvolený počet částí. Tím jsme získali délku hrany jedné krychle 3D mřížky a z této hodnoty jsme již mohli určit na kolik krychlí lze bounding box rozdělit i ve zbylých rozměrech. Teď již můžeme prostor rozdělit do pravidelné mřížky a spočítat hodnoty v jejích vrcholech. Na mřížku poté nasadíme zmiňovaný Marching Cubes algoritmus, kterým získáme snadno zobrazitelný polygonální model.
Měření byla provedena na konfiguraci: Pentium 4 2,8 GHz, 2GB RAM, ATI FireGL T2. Měřeno na modelech eight.tri (766 vrcholů) a cow.tri (2905 vrcholů, získáno z modelu distribuovaného s [SGI]).

3.2.2.1. Vliv použité bázové funkce a polynomu

Podíváme se, jaké výsledky nám RBF rekonstrukce poskytne, pokud použijeme různé bázové funkce a různé rozšiřující elementy lineárního systému.

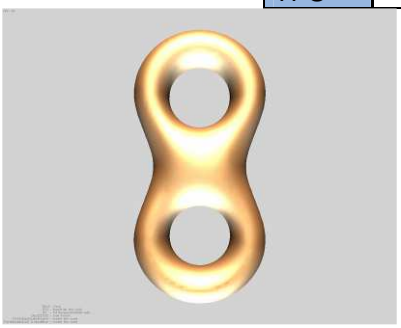
Nastavení měření:

Bounding box zvětšený o 20% (v každém směru, tzn. o 40% pro každý rozměr)
Off-surface body v rozích bounding boxu
Hodnota v off-surface bodech = 1
Solver = GMRES (1000 iterací)
Počet krychlí v nejdelší ose bounding boxu = 100

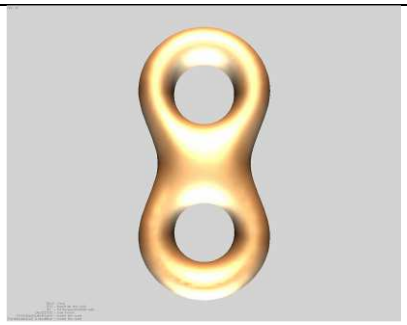
Rozšíření lin. systému konstantou:

Tabulka 3.2.2.1.A: Model eight.tri,

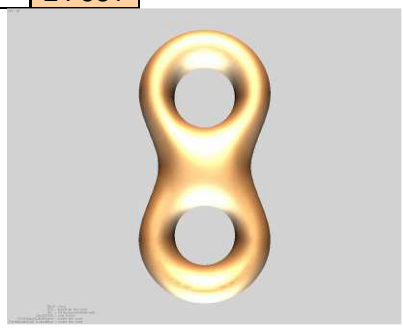
Bázová funkce	Vyřešení systému [ms]	Samplování dat [ms]	Extrakce [ms]	Celkem [ms]
Cubic	7625	8641	437	16 703
Linear	3485	8218	437	12 140
TPS	8453	15656	422	24 531



Cubic



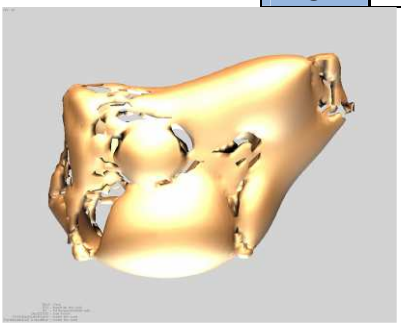
Linear



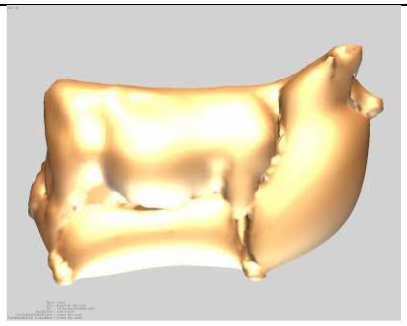
TPS

Tabulka 3.2.2.1.B: Model cow.tri

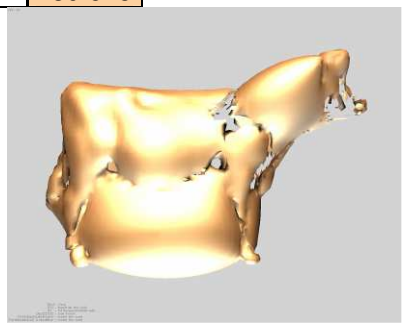
Bázová funkce	Vyřešení systému [ms]	Samplování dat [ms]	Extrakce [ms]	Celkem [ms]
Cubic	294031	66016	937	360 984
Linear	123515	63282	797	187 594
TPS	362812	116969	844	480 625



Cubic



Linear



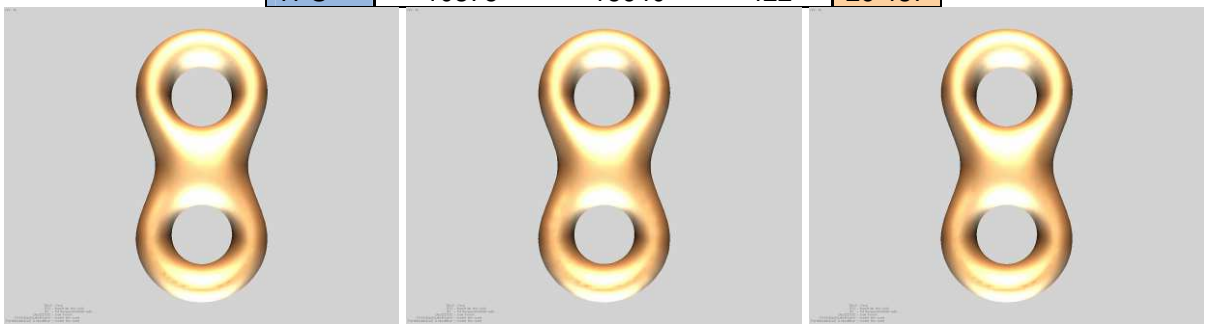
TPS

Model eight.tri se zrekonstruoval perfektně, cow.tri o poznání hůře. Vinou je zřejmě přidání pouhých 8 bodů do rohů bounding boxu, což pro zrekonstruování takto komplexního modelu nestačí ani počtem, ani jejich pozicí. Nejrychleji se dle očekávání vypočítal systém tvořený lineárními bázovými funkcemi (viz vzorce pro výpočet jednotlivých funkcí).

Rozšíření lin. systému lineárním polynomem:

Tabulka 3.2.2.1.C: Model eight.tri

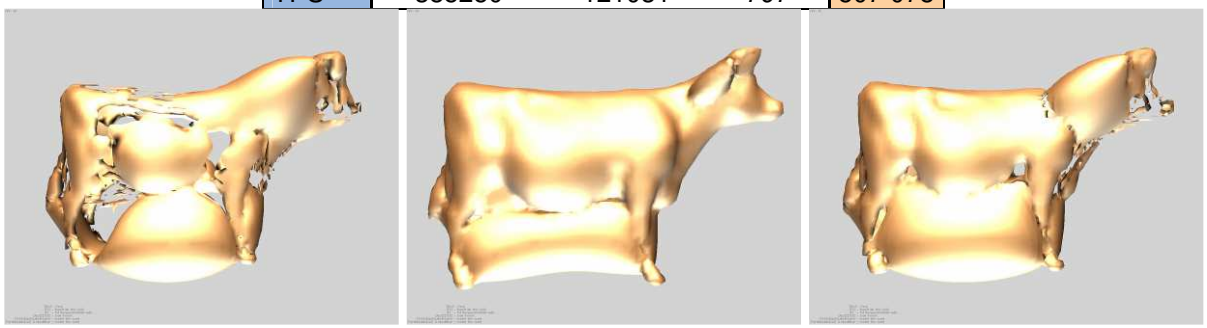
Bázová funkce	Vyřešení systému [ms]	Samplování dat [ms]	Extrakce [ms]	Celkem [ms]
Cubic	8437	8703	437	17 577
Linear	3750	8172	437	12 359
TPS	10375	15640	422	26 437



Cubic *Linear* *TPS*

Tabulka 3.2.2.1.D: Model cow.tri

Bázová funkce	Vyřešení systému [ms]	Samplování dat [ms]	Extrakce [ms]	Celkem [ms]
Cubic	286328	63828	859	351 015
Linear	132703	61062	672	194 437
TPS	385250	121031	797	507 078



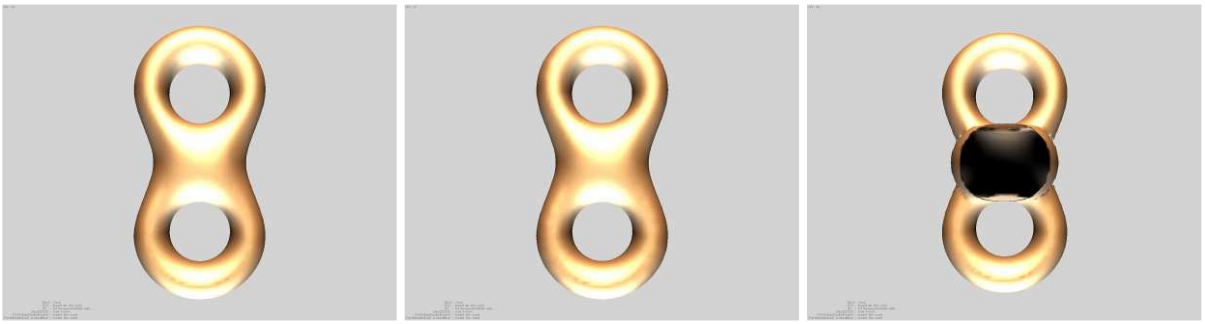
Cubic *Linear* *TPS*

Také v tomto případě se první model zrekonstruoval velmi dobře.
Druhý model je při použití lineární bázové funkce opět nejen nejrychleji vypočítaný, ale také již poměrně dobře zrekonstruovaný. Nicméně stále platí, že pro generování off-surface bodů bude pro dosažení nejlepších výsledků potřeba použít nějakého sofistikovanějšího algoritmu, než je pouhé přidání 8 bodů do rohů bounding boxu.

Lin. systém bez rozšíření:

Tabulka 3.2.2.1.E: Model eight.tri

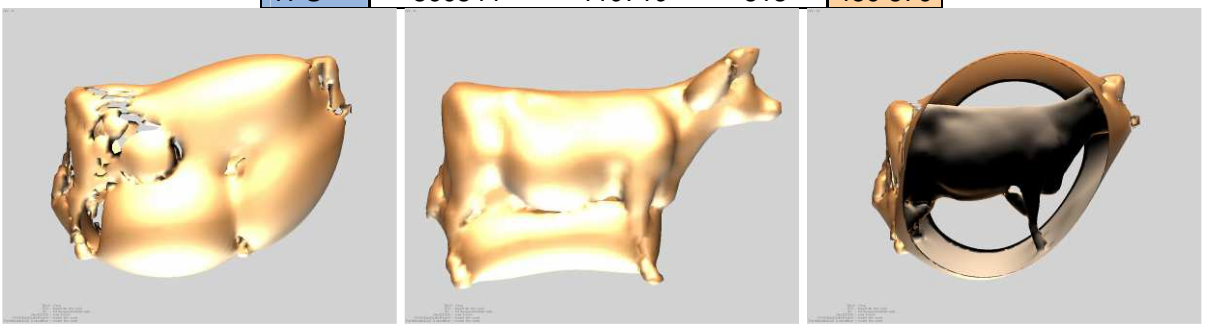
Bázová funkce	Vyřešení systému [ms]	Samplování dat [ms]	Extrakce [ms]	Celkem [ms]
Cubic	7078	8704	438	16 220
Linear	3031	8125	437	11 593
TPS	7750	15594	485	23 829



Cubic *Linear* *TPS*

Tabulka 3.2.2.1.F: Model cow.tri

Bázová funkce	Vyřešení systému [ms]	Samplování dat [ms]	Extrakce [ms]	Celkem [ms]
Cubic	223922	65235	922	290 079
Linear	116922	62266	672	179 860
TPS	366344	119719	813	486 876



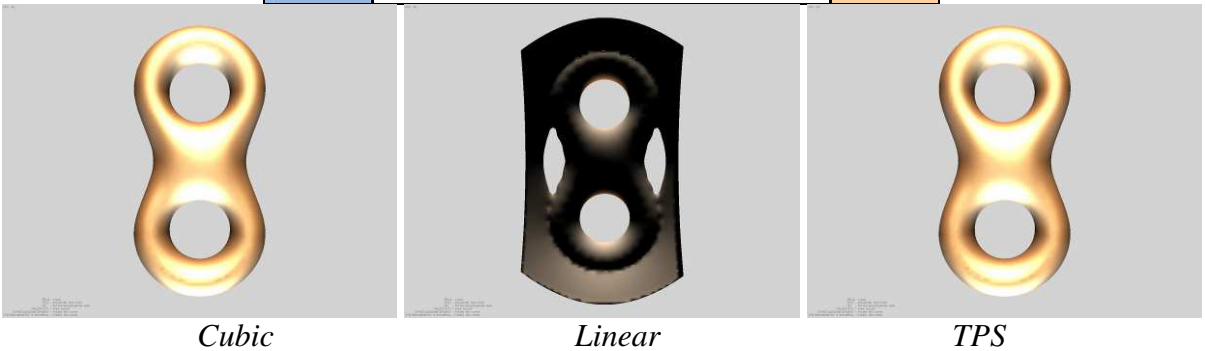
Cubic *Linear* *TPS*

Tabulkové hodnoty opět potvrdily předchozí výsledky.
Ovšem při použití TPS bázové funkce dochází k tvorbě velice nepříjemných artefaktů, které pravděpodobně vycházejí ze skutečnosti, že tato bázová funkce vyžaduje přítomnost jakéhokoliv rozšiřujícího elementu při vytváření lineárního systému.

Rozšíření lin. systému kvadratickým polynomem:

Tabulka 3.2.2.1.G: Model eight.tri

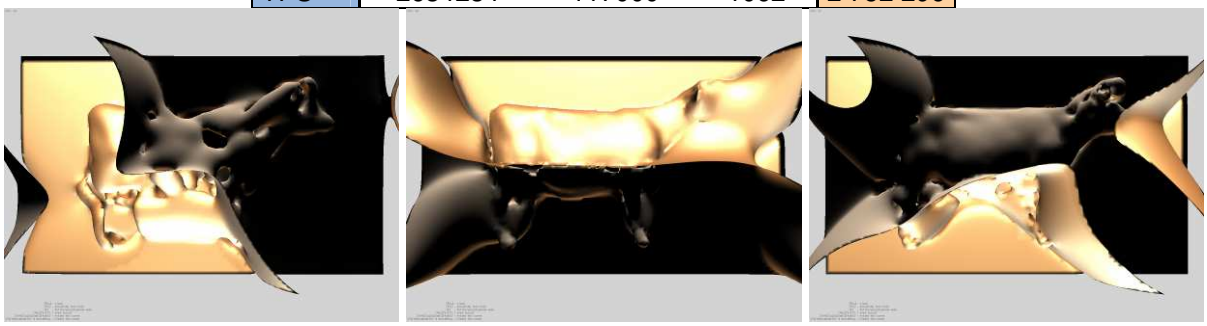
Bázová funkce	Vyřešení systému [ms]	Samplování dat [ms]	Extrakce [ms]	Celkem [ms]
Cubic	10640	8750	422	19 812
Linear	5500	8141	531	14 172
TPS	15156	15719	422	31 297



Cubic *Linear* *TPS*

Tabulka 3.2.2.1.H: Model cow.tri

Bázová funkce	Vyřešení systému [ms]	Samplování dat [ms]	Extrakce [ms]	Celkem [ms]
Cubic	1541000	72265	1218	1 614 483
Linear	2170984	70735	1250	2 242 969
TPS	2634234	117000	1062	2 752 296



Cubic *Linear* *TPS*

Kvadratický polynom nás opět (podobně jako při rekonstrukci obrazů) překvapil svým chováním. Zrekonstruované modely jsou na první pohled nejhorší, zejména díky přítomnosti plochy, která modelem prochází. Všimněme si však, že se podařilo nalézt nohy krávy, které v předchozích případech dělaly největší problémy. Dále je překvapivý také čas výpočtu při použití kubické funkce a celkové časy výpočtu systému vůbec, které jsou o řád vyšší než v předchozích měřeních.

Pokud se podíváme na výsledky rekonstrukce z pohledu použitého rozšiřujícího elementu, vidíme, že nejlepšího výsledku jsme dosáhli s lineárním polynomem a lineární bázovou funkcí. Jak již však bylo řečeno, výsledky jsou silně ovlivněné způsobem generování off-surface bodů, které v našem případě bylo triviální.

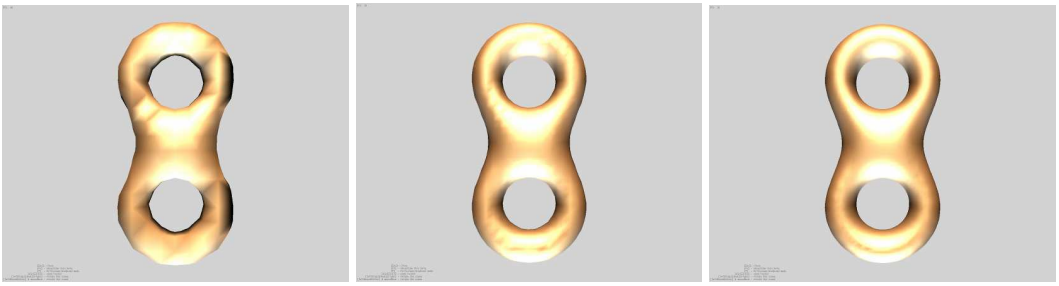
3.2.2.2. Vliv počtu krychlí v 3D rastru

V předchozí podkapitole jsme všechna měření provedli pro rozdělení bounding boxu modelu v nejdelším směru na 100 krychlí. Tento parametr nám udává rozlišení, v jakém chceme výsledný model zobrazit a tedy jemnost jeho povrchu.

Bounding box zvětšený o 20% (v každém směru, tzn. o 40% pro každý rozměr)
Off-surface body v rozích bounding boxu
Hodnota v off-surface bodech = 1
Solver = GMRES (1000 iterací)
RBF = lineární
Polynom = lineární

Tabulka 3.2.2.2.A

Počet krychlí	Vyřešení systému [ms]	Samplování dat [ms]	Extrakce [ms]	Celkem [ms]
25	3735	140	16	3 891
50	3750	1047	62	4 859
100	3750	8172	437	12 359



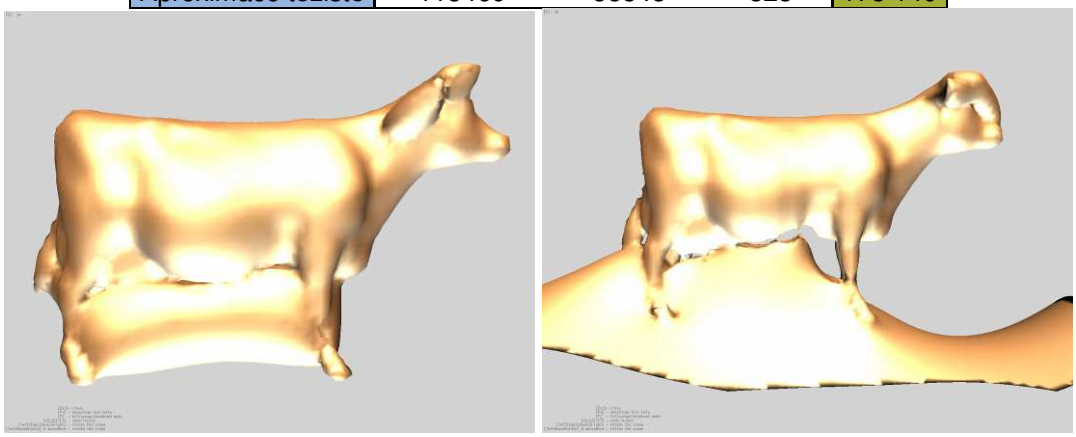
25 samplů *50 samplů* *100 samplů*

Pro zajímavost ještě uvádíme srovnání generování off-surface bodů v rozích bounding boxu s metodou, kdy umístíme jediný off-surface bod do aproximace těžiště modelu.

Bounding box zvětšený o 20% (v každém směru, tzn. o 40% pro každý rozměr)
Hodnota v off-surface bodech = 1
Solver = GMRES (1000 iterací)
RBF = lineární
Polynom = lineární
Počet krychlí v nejdelší ose bounding boxu = 100

Tabulka 3.2.2.2.B

Umístění off-surface bodů	Vyřešení systému [ms]	Samplování dat [ms]	Extrakce [ms]	Celkem [ms]
Bounding box	132703	61062	672	194 437
Aproximace těžiště	118469	58843	828	178 140



Bounding box *Aproximace těžiště*

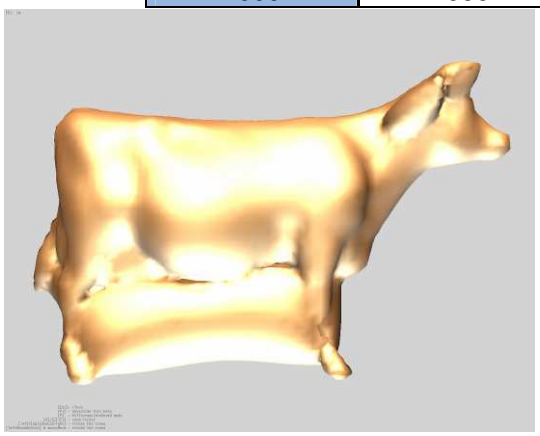
Dle očekávání vidíme, že pro složitější modely je metoda, kdy přidáme pouze jediný off-surface bod do těžiště modelu, nevhodná.

Jako druhou ukázku si předvedeme, co se stane, pokud do rohů bounding boxu umístíme jinou hodnotu než standardní 1.

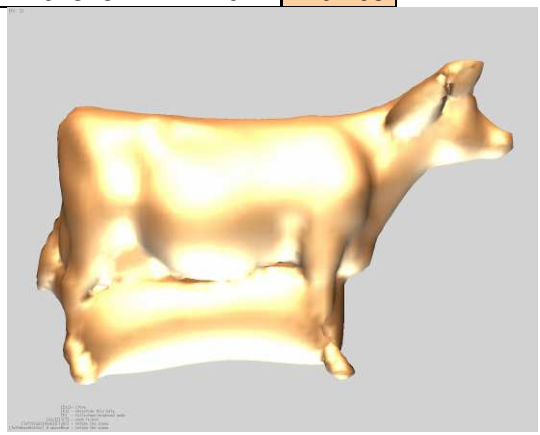
Bounding box zvětšený o 20% (v každém směru, tzn. o 40% pro každý rozměr)
Off-surface body v rozích bounding boxu
Solver = GMRES (1000 iterací)
RBF = lineární
Polynom = lineární
Počet krychlí v nejdelší ose bounding boxu = 100

Tabulka 3.2.2.2.C

Fční hodnota off-surface bodů	Vyřešení systému [ms]	Samplování dat [ms]	Extrakce [ms]	Celkem [ms]
1	132703	61062	672	194 437
1000	217656	61328	719	279 703



Fční hodnota off-surface bodů: 1



1000

Z obrázků vidíme, že rozdíly jsou minimální, a tedy tento parametr nebudeme sledovat.

4. Závěr

V oblasti rekonstrukce obrazů bylo navrženo několik nových metod, jejichž popis i výsledky jsme si představili. Zároveň jsme navrhli nový algoritmus rekonstrukce, se kterým jsme chtěli dosáhnout lepších vizuálních výsledků v porovnání s předchozími pracemi – zejména Ing. Karla Uhlíře.
Podle provedených měření a testů lze prohlásit, že se nám podařilo jeho řešení překonat a námi navržený algoritmus ve srovnatelné době výpočtu dosahuje znatelně lepších výsledků. Především obrazy poškozené extrémním šumem se nám podařilo zrekonstruovat způsobem, jenž předčil mnohá naše očekávání. (Průměr PSNR všech 3 RGB kanálů u Uhlíře dosahuje hodnoty PSNR 22.12 dB, náš algoritmus dokonce 24.75 dB, navíc vizuální vjem je podstatně lepší.)
Provedli jsme srovnání několika sledovaných faktorů ovlivňujících kvalitu výsledné rekonstrukce a výsledky vyhodnotili. Na základě nich jsme vyvodili doporučení, kterých je vhodné se držet, chceme-li dosáhnout co nejlepších výsledků. Připomeňme tedy, že metoda přidávání bodů značně vylepšuje vizuální vjem rekonstrukce a směr rekonstrukce doporučujeme v obou osách obrazu současně. Jako nejvýhodnější poloměr okolí, dávající kvalitní výsledky v rozumném čase, doporučujeme hodnotu 3, tzn. okolí o rozměrech 7x7 pixelů. Použitá bázová funkce by měla být lineární funkce, celý lineární systém je vhodné rozšířit nejlépe konstantou místo polynomu. To vše, ve spojení s námi představeným algoritmem, se ukázalo jako nejlepší volbou pro rekonstrukce.
Do budoucna je zde prostor pro další optimalizace a zvyšování kvality rekonstrukce. Především problematika přidávání bodů (viz kapitola **3.1.3.2 Přidávání bodů (AddWeightedAveragePoints))** se jeví jako velice perspektivní a lze se tedy touto cestou do budoucna ubírat a hledat nové možnosti jejího využití a zlepšení.

V oblasti extrakce isočar jsme ověřili základní principy a dosáhli dílčích výsledků. Téma interpolace pomocí RBF v zadaných oblastech je natolik rozsáhlé, že jsme se rozhodli věnovat většinu úsilí rekonstrukci poškozených obrazů. Z tohoto důvodu je rozsah zkoumání v oblasti extrakce isočar podstatně menší než u výše zmiňované rekonstrukce obrazů, která byla podrobena našemu podrobnému zájmu. Uvědomme si, že pro extrakce je třeba naprosto odlišných dat a celkově jiný přístup k problému, jehož zvládnutí přesahuje rozsah této práce. Nicméně se budeme i této oblasti nadále věnovat, především kvůli perspektivnímu směru, kterým se daná oblast ubírá.

Literatura

[Amid02] AMIDROR,I.: Scattered data interpolation methods for electronic imaging systems: a survey, Journal of Electronic Imaging, Vol. 11 No. 2, Lausanne, 2002

[Beatson_WWW] R.BEATSON Homepage (University of Canterbury, New Zealand) [online], http://www.math.canterbury.ac.nz/~r.beatson/, březen 2007

[Berkeley_WWW] Multimedia Systems and Applications - Image Quality Computation [online], http://bmrc.berkeley.edu/courseware/cs294/fall97/assignment/psnr.html, leden 2007

[BioMesh_WWW] BioMesh Project – dataset site [online], http://www-unix.mcs.anl.gov/~csverma/BioMesh/Dataset.html, duben 2007

[Carr01] CARR,J.C., et al.: Reconstruction and Representation of 3D Objects with Radial Basis Functions, Computer Graphics (SIGGRAPH 2001 proceedings), s. 67-76, 2001

[Cerm05] ČERMÁK,L., HLAVIČKA,R.: Numerické metody, Akademické nakladatelství CERM, Brno, 2005

[MT_WWW] The MT (Multi-Tesselation) package – dataset site [online], ftp://ftp.disi.unige.it/person/MagilloP/MT_SW/DATA/datasets.html, duben 2007

[Duchon77] DUCHON,J.: Splines minimizing rotation-invariant semi-norms in Sobolev space, Constructive Theory of Functions of Several Variables, Lecture Notes in Math No. 571, Springer-Verlag, Berlin, 1977

[Hardy71] HARDY,L.R.: Multiquadric equations of topography and other irregular surfaces. Journal of Geophysical Research,76(8):1905–1915, 1971.

[IPLR_WWW] Image processing learning resources – Logarithm Operator [online], http://homepages.inf.ed.ac.uk/rbf/HIPR2/pixlog.htm, březen 2007

[Kubic05] KUBÍČEK,M., DUBCOVÁ,M., JANOVSKÁ,D.: Numerické metody a algoritmy, Vydavatelství VŠCHT Praha, 2005

[Lorensen87] LORENSEN,W.E., CLINE,H.E.: Marching cubes: A high resolution 3D surface construction algorithm, Computer Graphics, Proceedings of SIGGRAPH 87, Volume 21(4), s. 163–169, July 1987

[MathNet_WWW] Math.NET Project [online], http://mathnet.opensourcedotnet.info/, prosinec 2006

[Micchelli86] MICCHELLI,C.A.: Interpolation of scattered data: distance matrice and conditionally positive definite functions, Constr. Approx., s. 11-22, 1986.

[Morse01] MORSE,B.S., et al.: Interpolating Implicit Surfaces From Scattered Surface Data Using Compactly Supported Radial Basis Functions, Shape Modeling International 2001, s. 89–98, Genova, 2001

[Mullan04] MULLAN,M. J.: The level sets of surfaces fit with radial basis functions, M.S. thesis,University of Illinois at Urbana-Champaign, 2004

[Ohtake03] OHTAKE,Y., BELYAEV,A., SEIDEL,H.-P.: A Multi-scale Approach to 3D Scattered Data Interpolation with Compactly Supported Basis Functions, Proceedings of SMI ´2003, Soul, 2003

[Patera02] PATERA,J.: Metody extrakce iso-ploch pro volumetrická data a zobrazování skalárních veličin, Diplomová práce (vedoucí Václav Skala), Plzeň, 2002

[SGI] SGI Powerflip demo, aplikace (platforma OS IRIX)

[Schagen79] SCHAGEN,I.P.: Interpolation in Two Dimension – A New Technique, J. Inst. Maths Applics, s. 53-59, 23 (1979)

[Schoen38] SCHOENBERG,I.J.: Metric Spaces and Completely Monotone Functions, The Annals of Mathematics, 2nd Ser., Vol.39, No.4 s.811-841, 1938

[Tobor04] TOBOR,I., REUTER,P., SCHLICK,CH.: Efficient Reconstruction of Large Scattered Geometric Datasets using the Partition of Unity and Radial Basis Functions, vol. 12 of Journal of WSCG 2004, s. 467-474, Plzeň

[Uhlir05] UHLÍŘ,K.: Aplikace RBF v počítačové grafice a zpracování obrazu, Draft disertační práce (vedoucí Václav Skala), Plzeň, 2005

[Uhlir04] UHLÍŘ,K., PATERA,J., SKALA,V.: Radial Basis Function method for iso-line extraction, Electronic computers and informatics: proceedings of the sixth international scientific konference, s. 439–444, Košice, 2004

[Wang04] WANG,Z., et al.: Image quality assessment: From error visibility to structural similarity, IEEE Transactions on Image Processing, vol. 13, no. 4, s. 600-612, 2004

[Wend95] WENDLAND,H. - Piecewise polynomial, positive definite and compactly supported radial basis functions of minimal degree, Advances in Computational Mathematics 4, s. 389-396, 1995

[Wiki_WWW] Wikipedia, the free encyclopedia - Numerical linear algebra [online], http://en.wikipedia.org/wiki/Category:Numerical_linear_algebra, leden 2007

[Wright03] WRIGHT,G.B.: Radial Basis Function Interpolation: Numerical and Analytical Developments, Thesis, University of Colorado, 2003

[Wu05] WU,X.J., WANG,M.Y., XIA,Q.: Implicit fitting and smoothing using radial basis functions and partition of unity, In Proc. of 9th International Computer-Aided-Design and Computer Graphics Conference (CAD/Graphics'05), Hong Kong, 2005

[Zhang02] ZHANG,Y., ROHLING,R., PAI,D.K.: Direct Surface Extraction from 3D Freehand Ultrasound Images, IEEE Visualization 2002, Boston, 2002

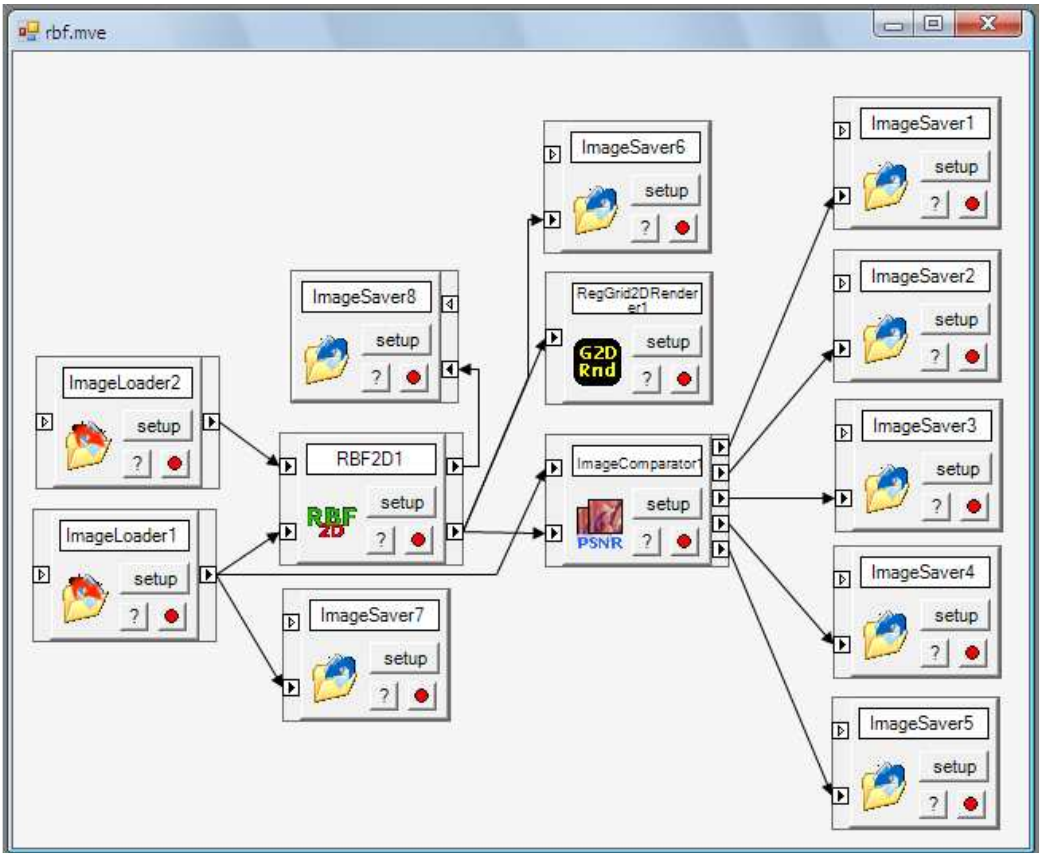
Příloha A

Uživatelská dokumentace

Jelikož výsledkem naší implementace jsou moduly do MVE-2 (klikněte pro stažení MVE-2), je ovládání shodné jako při použití každého jiného modulu v prostředí MVE-2.

Moduly je potřeba vhodně zapojit do MVE mapy, nastavování jejich parametrů probíhá klasicky stisknutím tlačítka *Setup* na příslušném modulu.
Mapy byly testovány v aktuální verzi – MVE-2 version 1.0 beta 5.

Ukázkové mapy jsou na přiloženém CD společně s příklady vstupních dat.



Příklad propojení modulů pro rekonstrukci obrazů

Popis nastavení modulů a jejich vstupních a výstupních portů

Rekonstrukce obrazů:

Modul ***RBF2D*** – rekonstruuje obraz RBF interpolací

- Vstupní porty
  - `InOriginalImage` (MVE-2 datový typ `RegGrid2D`)
    - Nejlépe originální obraz, pokud nemáme, použijeme poškozený obraz
  - `InMaskImage` (`RegGrid2D`)
    - Obraz masky poškození (černo-bílý obraz, kde černá značí poškození)
- Výstupní porty
  - `Output` (`RegGrid2D`)
    - Zrekonstruovaný obraz
  - `OutDefected` (`RegGrid2D`)
    - Poškozený obraz (kombinace vstupního obrazu a masky)

- Možnosti nastavení modulu
  - `Algorithm`
    - Algoritmus zpracování obrazu
  - `BrightnessThreshold`
    - Maximální jasová hodnota pixelu masky, která je brána jako poškození
  - `C`
    - Parametr některých bázových funkcí
  - `CellAttribute`
    - Atribut vstupních dat, pro převod MVE-2 typů do .NET datových typů
  - `ColorSpace`
    - Barevný prostor, ve kterém se má interpolace provádět
  - `DetectHoles`
    - Mají se při zpracovávání obrazu lokalizovat a přeskakovat díry?
  - `Direction`
    - Směr, ve kterém má být obraz zpracováván
  - `NeighbourhoodType`
    - Volba okolí zpracovávaného pixelu
  - `NeighbourRadius`
    - Velikost poloměru okolí
  - `Polynomial`
    - Jaký rozšiřující element lin. systému má být použit
  - `Rbf`
    - Volba radiální bázové funkce
  - `Solver`
    - Metoda řešení lineárního systému

Modul ***LinearInterpolation*** – rekonstruuje obraz bilineární interpolací

- Vstupní a výstupní porty – viz modul ***RBF2D***

- Možnosti nastavení modulu
  - `Algorithm`, `BrightnessThreshold`, `ColorSpace`, `DetectHoles` – viz modul ***RBF2D***

- `UseOnlyOriginalPixels`
  - Mají se v každé iteraci používat pouze originální pixely (True) nebo zda používat i pixely interpolované v předchozích iteracích (False)
- `HoleLength`
  - Pokud se mají lokalizovat a přeskakovat díry, hodnota určuje, jak mají být díry dlouhé (podobně jako parametr `NeighbourRadius` u ***RBF2D***)

Modul ***ImageComparator*** – porovnává dva vstupní obrazy a generuje chybové obrazy

- Vstupní porty
  - `InOriginalImage` (`RegGrid2D`)
    - Referenční obraz, se kterým chceme porovnávat
  - `InReconstructedImage` (`RegGrid2D`)
    - Obraz, který chceme porovnat vzhledem k referenčnímu
- Výstupní porty
  - `OutErrorImage` (`RegGrid2D`)
    - Chybový obraz
  - `OutErrorMSEImage` (`RegGrid2D`)
    - Vizualizace MSE
  - `OutErrorMSEImageLog` (`RegGrid2D`)
    - Vizualizace MSE zvýrazněná logaritmickým operátorem
  - `OutErrorMSEImageNegativ` (`RegGrid2D`)
    - Invertovaná vizualizace MSE
  - `OutErrorMSEImageNegativLog` (`RegGrid2D`)
    - Invertovaná vizualizace MSE zvýrazněná logaritmickým operátorem

- Možnosti nastavení modulu
  - `AddTimeStamp`
    - Má se do jména log souboru vložit čas jeho vytvoření?
  - `CreateSubdirectoryByDate`
    - True, chceme-li v zadané cestě pro uložení výsledků vytvořit podadresář podle aktuálního data a ukládat výsledky do něj
  - `ErrorImage`
    - Má se vytvořit chybová obraz?
  - `ErrorMSEImage`
    - Má se vytvořit vizualizace MSE?
  - `ErrorMSEImageLog`
    - Má se vytvořit vizualizace MSE s logaritmickým operátorem?
  - `ErrorMSEImageInv`
    - Má se vytvořit invertovaná vizualizace MSE?
  - `ErrorMSEImageInvLog`
    - Má se vytvořit invertovaná vizualizace MSE s log. operátorem?
  - `Path`
    - Cesta, kam se mají ukládat výsledky (log a vygenerované obrazy)
  - `Prefix`
    - Prefix log souboru (může obsahovat i požadavek o vytvoření nadřazeného adresáře, např. MojeVýsledky\logfile)
  - `SaveLog`
    - Má se ukládat logovací soubor?

Extrakce isoploch:

Modul ***RBFIsoSurfaceSolver*** – sestaví a vyřeší lineární systém

- Vstupní porty
  - `InTriangleMesh` (`TriangleMesh`)
    - Načtený model ve formátu .tri
- Výstupní porty
  - `OutBoundingBoxCoords` (`VectorND`)
    - 6-prvkový vektor minimální a maximální souřadnice v každé ose (minX, maxX, minY, maxY, minZ, maxZ)
  - `OutLambdaVector` (`VectorND`)
    - Vyřešený vektor vah ***x*** soustavy ***Ax=b***
  - `OutVertices` (`UniformDataArray`)
    - Výstupní pole bodů, ze kterých je sestaven lineární systém (tedy načtené body .tri modelu + vygenerované off-surface body)

- Možnosti nastavení modulu
  - `Algorithm`
    - Algoritmus generování off-surface bodů
  - `BboxExtension`
    - O kolik procent se má bounding box zvětšit oproti min/max hodnotám bodů modelu?
  - `C, Polynomial, Rbf, Solver` – viz modul ***RBF2D***
  - `ExtendedBoundingBox`
    - Má se zvětšit bounding box?
  - `FunctionValue`
    - Funkční hodnota implicitního povrchu
  - `OffSurfaceValue`
    - Funkční hodnota v off-surface bodech
  - `SupportRadius`
    - Poloměr působnosti CSRBF bázových funkcí

Modul ***RBFVolumeDataSampler*** – vzorkuje prostor v požadovaném rozlišení pro vizualizaci isoplochy např. pomocí Marching Cubes algoritmu

- Vstupní porty
  - `InBoundingBoxCoords` (`VectorND`)
    - 6-prvkový vektor minimální a maximální souřadnice v každé ose
  - `InLambdaVector` (`VectorND`)
    - Vyřešený vektor vah ***x*** soustavy ***Ax=b***
  - `InVertices` (`UniformDataArray`)
    - Výstupní pole bodů, ze kterých je sestaven lineární systém
- Výstupní porty
  - `OutVolData` (`RegGridND`)
    - Navzorkovaná mřížka hodnot pro Marching Cubes algoritmus

- Možnosti nastavení modulu
  - `C, Polynomial, Rbf` – viz modul ***RBF2D***
  - `SamplesInLongestDimension`
    - Počet vzorků 3D mřížky v podél nejdelší hrany bounding boxu
  - `SupportRadius` – viz modul ***RBFIsoSurfaceSolver***

Příloha B

Dokumentace pro instalaci

Podobně jako v Příloze A, kdy je používáni našich modulů stejné jako u jiných MVE-2 modulů, je také jejich instalace shodná jako při používání libovolného modulu v MVE-2.
Stačí tedy adresářovou strukturu v archívu naší implementace z přiloženého CD nakopírovat do adresáře, kde máme MVE-2 umístěné na disku. O natažení příslušných knihoven se postará jádro MVE-2 samo, na uživateli je tedy pouze vytvoření nové mapy, případně použití a modifikace některé z ukázkových map.
Pro zajištění funkčnosti, je na tomto CD také umístěna aktuální funkční verze MVE-2 prostředí s aktuálními knihovnami MVE-2 jádra, které mimo jiné zajišťují komunikaci mezi moduly a jsou v nich definovány datové typy pro tuto komunikaci používané.

Příloha C

Programátorská dokumentace

Implementace byla dle zadání provedena v jazyce C# (2.0) do prostředí MVE-2 (beta 5).
Bylo použito IDE Microsoft Visual Studio 2005, pro matematické výpočty jsme použili volně šiřitelnou knihovnu Math.Net (MathNet.Numerics verze 0.3.0.0) [MathNet_WWW].
Zdrojové kódy jsou hojně komentovány, ke každému modulu je z těchto komentářů vygenerována MMDoc i XML dokumentace, na které případné zájemce odkazujeme. Další informace o Solutions lze nalézt také v readme souborech u obou archivů s implementací.

Výsledkem naší implementace jsou 2 Solutions:

1. ***RBF Solution*** pro rekonstrukci obrazů, obsahující projekty
   - **ImageComparator**
     modul pro porovnávání dvou vstupních obrazů (MSE a PSNR metriky), může také vygenerovat 5 různých chybových obrazů a ukládat logovací soubor o provedeném porovnání
   - **ImageLoader**
     modul pro načítání obrázků z formátů BMP, GIF, JPG, PNG a TIF
   - **ImageSaver**
     modul pro ukládání obrazů ve formátech BMP, GIF, JPG, PNG a TIF
   - **ImagingLib**
     pomocná knihovna, jejíž metody používají ostatní projekty z této Solution, Obsahuje metody pro konverzi mezi barevnými systémy, dále jsou v ní implementovány metody GetPixelPtr a SetPixelPtr, které přistupují k obrazům přes pointery a jsou tedy podstatně rychlejší než defaultní GetPixel a SetPixel metody z .NETu 2.0., dále zde nalezneme metody pro výpočet MSE či PSNR a některé další metody.
   - **LinearInterpolation**
     modul pro rekonstrukci poškozených obrazů bilineární interpolací
   - **RBF2D**
     modul pro rekonstrukci poškozených obrazů RBF interpolací
2. ***RBFIsoSurface Solution*** pro extrakci isoploch, která obsahuje projekty
   - **RBFIsoSurfaceSolver**
     modul sestaví a vyřeší lineární systém pro extrakci isoploch
   - **RBFLib**
     pomocná knihovna pro výpočet řešení RBF různými metodami a také výpočet vzorků 3D mřížky
   - **RBFVolumeDataSampler**
     modul pro vzorkování 3D mřížky v požadovaném rozlišení

Příloha D

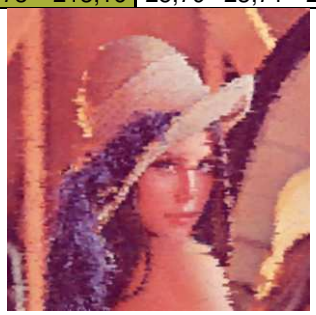
Výsledky měření

Získávání okolních bodů (kapitola 3.1.6.2.1, strana 51)

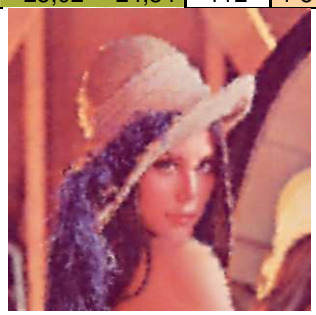
Tabulka 3.1.6.2.1.A: Vstup: Lena + Noise95,503%, Směr: LeftRight

Typ okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
NAP	528,09	742,94	604,35	625,12	570,65	20,90	19,42	20,32	20,21	20,57	96	23 579
AWAP	261,97	402,74	237,54	300,75	313,72	23,95	22,08	24,37	23,47	23,17	112	23 300
DN	171,40	276,75	181,03	209,73	213,16	25,79	23,71	25,55	25,02	24,84	112	1 547 428

NAP



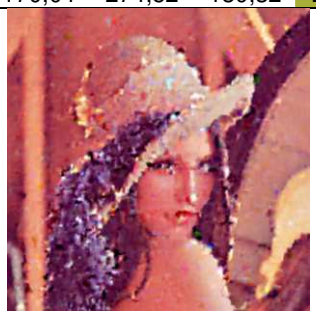
AWAP



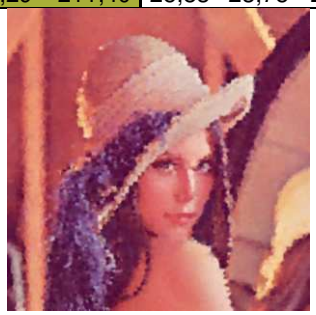
DN

Tabulka 3.1.6.2.1.B: Vstup: Lena + Noise95,503%, Směr: LeftRighTopBottom

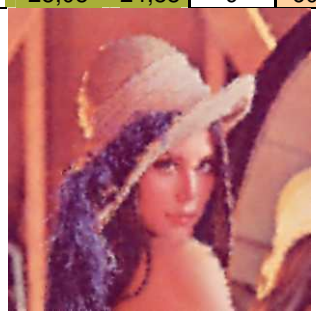
Typ okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
NAP	395,99	592,24	483,39	490,54	453,23	22,15	20,41	21,29	21,28	21,57	14	21 844
AWAP	194,95	315,65	203,45	238,02	242,64	25,23	23,14	25,05	24,47	24,28	9	21 890
DN	170,04	274,52	180,32	208,29	211,40	25,83	23,75	25,57	25,05	24,88	9	601 735



NAP



AWAP



DN

Tabulka 3.1.6.2.1.C: Vstup: Lena + Wide, Směr: LeftRight

Typ okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
NAP	175,67	224,04	134,21	177,98	183,94	25,68	24,63	26,85	25,72	25,48	206	3 837
AWAP	305,67	355,14	126,28	262,36	295,94	23,28	22,63	27,12	24,34	23,42	206	3 951
DN	235,70	279,63	115,11	210,15	233,48	24,41	23,66	27,52	25,20	24,45	206	113 476


NAP


AWAP


DN

Tabulka 3.1.6.2.1.D: Vstup: Lena + Wide, Směr: LeftRightTopBottom

Typ okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
NAP	214,83	297,44	227,24	246,50	232,01	24,81	23,40	24,57	24,26	24,48	41	2 936
AWAP	165,70	212,02	104,73	160,82	173,31	25,94	24,87	27,93	26,24	25,74	41	3 250
DN	169,88	223,74	103,63	165,75	181,81	25,83	24,63	27,98	26,15	25,53	41	1 657 221


NAP


AWAP


DN

Volba poloměru okolí (kapitola 3.1.6.2.2, strana 52)

A. Testovací obraz Fruit

Tabulka 3.1.6.2.2.A: Vstup: Fruit + Fun, Směr: LeftRight

Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	2,02	2,02	2,02	2,02	2,02	45,08	45,08	45,08	45,08	45,08	31	407
r=2	1,43	1,43	1,43	1,43	1,43	46,58	46,58	46,58	46,58	46,58	31	984
r=3	1,32	1,32	1,32	1,32	1,32	46,93	46,93	46,93	46,93	46,93	30	3 014
r=4	1,26	1,26	1,26	1,26	1,26	47,14	47,14	47,14	47,14	47,14	30	9 752

Tabulka 3.1.6.2.2.B: Vstup: Fruit + Fun, Směr: LeftRighTopBottom

Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	1,68	1,68	1,68	1,68	1,68	45,87	45,87	45,87	45,87	45,87	8	328
r=2	1,36	1,36	1,36	1,36	1,36	46,80	46,80	46,80	46,80	46,80	8	893
r=3	1,28	1,28	1,28	1,28	1,28	47,05	47,05	47,05	47,05	47,05	8	3 001
r=4	1,25	1,25	1,25	1,25	1,25	47,17	47,17	47,17	47,17	47,17	7	9 985

Tabulka 3.1.6.2.2.C: Vstup: Fruit + Grid, Směr: LeftRight

Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	120,10	120,10	120,10	120,10	120,10	27,34	27,34	27,34	27,34	27,34	511	9 212
r=2	117,32	117,32	117,32	117,32	117,32	27,44	27,44	27,44	27,44	27,44	509	22 764
r=3	115,03	115,03	115,03	115,03	115,03	27,52	27,52	27,52	27,52	27,52	507	88 915
r=4	115,18	115,18	115,18	115,18	115,18	27,52	27,52	27,52	27,52	27,52	505	221 415

Tabulka 3.1.6.2.2.D: Vstup: Fruit + Grid, Směr: LeftRighTopBottom

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	120,10	120,10	120,10	120,10	120,10	27,34	27,34	27,34	27,34	27,34	1	5 390
r=2	117,32	117,32	117,32	117,32	117,32	27,44	27,44	27,44	27,44	27,44	1	20 609
r=3	115,03	115,03	115,03	115,03	115,03	27,52	27,52	27,52	27,52	27,52	1	86 468
r=4	115,18	115,18	115,18	115,18	115,18	27,52	27,52	27,52	27,52	27,52	1	220 406

Tabulka 3.1.6.2.2.E: Vstup: Fruit + Noise45,815%, Směr: LeftRight

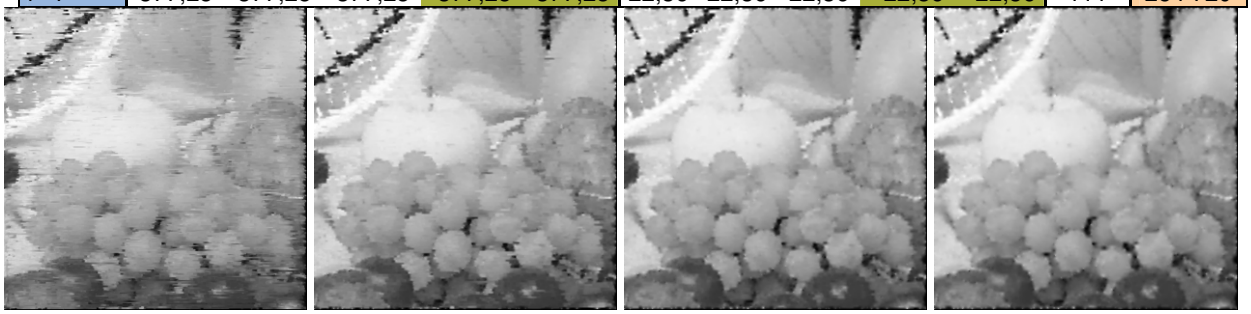
Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	42,81	42,81	42,81	42,81	42,81	31,82	31,82	31,82	31,82	31,82	8	3 485
r=2	34,86	34,86	34,86	34,86	34,86	32,71	32,71	32,71	32,71	32,71	8	18 189
r=3	33,36	33,36	33,36	33,36	33,36	32,90	32,90	32,90	32,90	32,90	8	72 032
r=4	32,81	32,81	32,81	32,81	32,81	32,97	32,97	32,97	32,97	32,97	7	213 048

Tabulka 3.1.6.2.2.F: Vstup: Fruit + Noise45,815%, Směr: LeftRighTopBottom

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	42,60	42,60	42,60	42,60	42,60	31,84	31,84	31,84	31,84	45,08	2	4 016
r=2	34,92	34,92	34,92	34,92	34,92	32,70	32,70	32,70	32,70	46,58	2	18 703
r=3	33,39	33,39	33,39	33,39	33,39	32,90	32,90	32,90	32,90	46,93	2	72 359
r=4	32,81	32,81	32,81	32,81	32,81	32,97	32,97	32,97	32,97	47,14	1	220 422

Tabulka 3.1.6.2.2.G: Vstup: Fruit + Noise95,503%, Směr: LeftRight

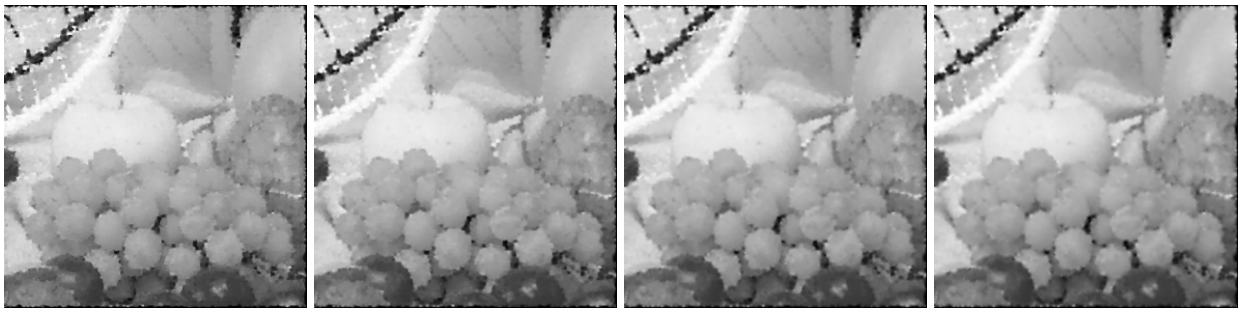
Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	518,76	518,76	518,76	518,76	518,76	20,98	20,98	20,98	20,98	20,98	112	7 894
r=2	433,37	433,37	433,37	433,37	433,37	21,76	21,76	21,76	21,76	21,76	112	27 903
r=3	390,85	390,85	390,85	390,85	390,85	22,21	22,21	22,21	22,21	22,21	111	92 623
r=4	377,28	377,28	377,28	377,28	377,28	22,36	22,36	22,36	22,36	22,36	111	254 720



r=1 *r=2* *r=3* *r=4*

Tabulka 3.1.6.2.2.H: Vstup: Fruit + Noise95,503%, Směr: LeftRighTopBottom

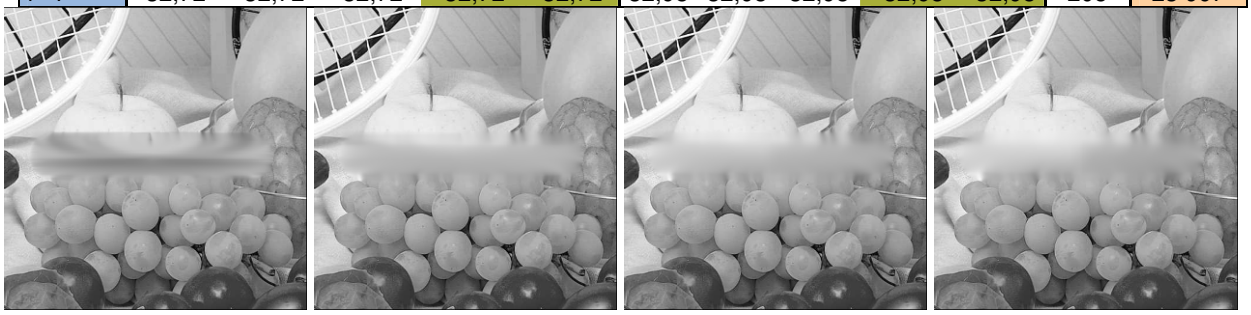
Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	444,32	444,32	444,32	444,32	444,32	21,65	21,65	21,65	21,65	21,65	9	6 016
r=2	411,10	411,10	411,10	411,10	411,10	21,99	21,99	21,99	21,99	21,99	9	23 251
r=3	389,39	389,39	389,39	389,39	389,39	22,23	22,23	22,23	22,23	22,23	8	82 673
r=4	375,64	375,64	375,64	375,64	375,64	22,38	22,38	22,38	22,38	22,38	8	231 905



r=1 *r=2* *r=3* *r=4*

Tabulka 3.1.6.2.2.I: Vstup: Fruit + Wide, Směr: LeftRight

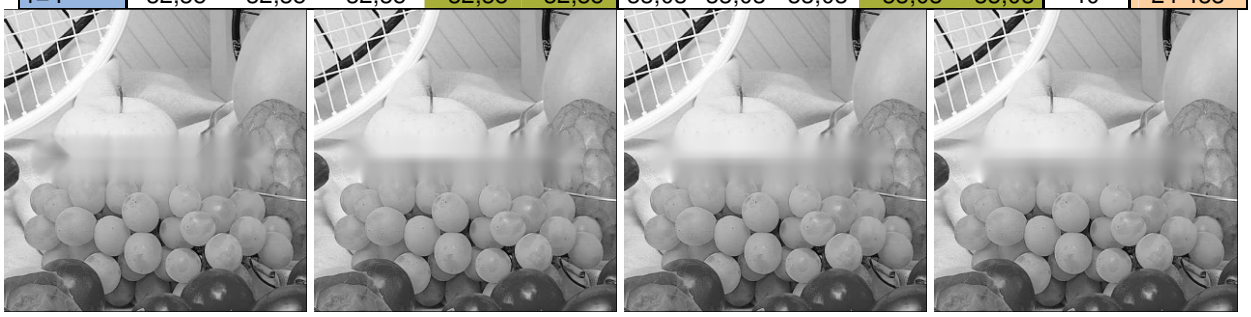
Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	130,99	130,99	130,99	130,99	130,99	26,96	26,96	26,96	26,96	26,96	206	2 235
r=2	60,08	60,08	60,08	60,08	60,08	30,34	30,34	30,34	30,34	30,34	206	4 029
r=3	41,27	41,27	41,27	41,27	41,27	31,97	31,97	31,97	31,97	31,97	205	9 648
r=4	32,72	32,72	32,72	32,72	32,72	32,98	32,98	32,98	32,98	32,98	205	23 997



r=1 *r=2* *r=3* *r=4*

Tabulka 3.1.6.2.2.J: Vstup: Fruit + Wide, Směr: LeftRighTopBottom

Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	61,87	61,87	61,87	61,87	61,87	30,22	30,22	30,22	30,22	30,22	41	1 423
r=2	36,78	36,78	36,78	36,78	36,78	32,47	32,47	32,47	32,47	32,47	41	3 228
r=3	33,81	33,81	33,81	33,81	33,81	32,84	32,84	32,84	32,84	32,84	41	9 052
r=4	32,35	32,35	32,35	32,35	32,35	33,03	33,03	33,03	33,03	33,03	40	24 438

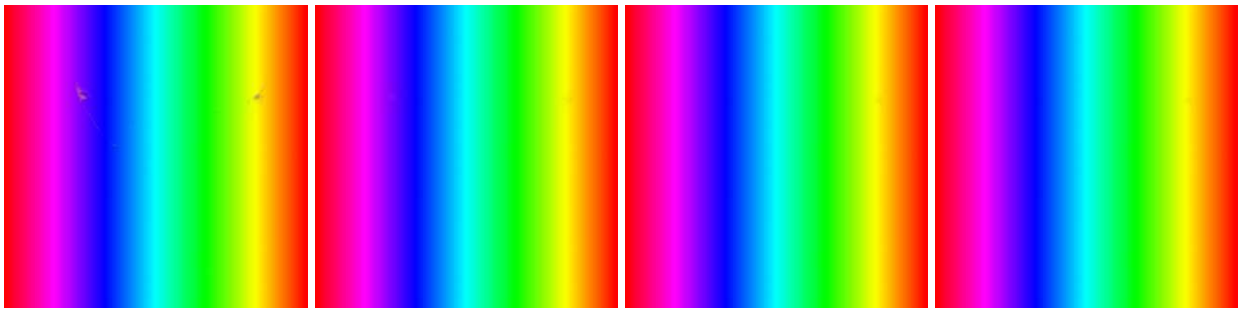


r=1 *r=2* *r=3* *r=4*

B. Testovací obraz Gradient

Tabulka 3.1.6.2.2.K: Vstup: Gradient + Fun, Směr: LeftRight

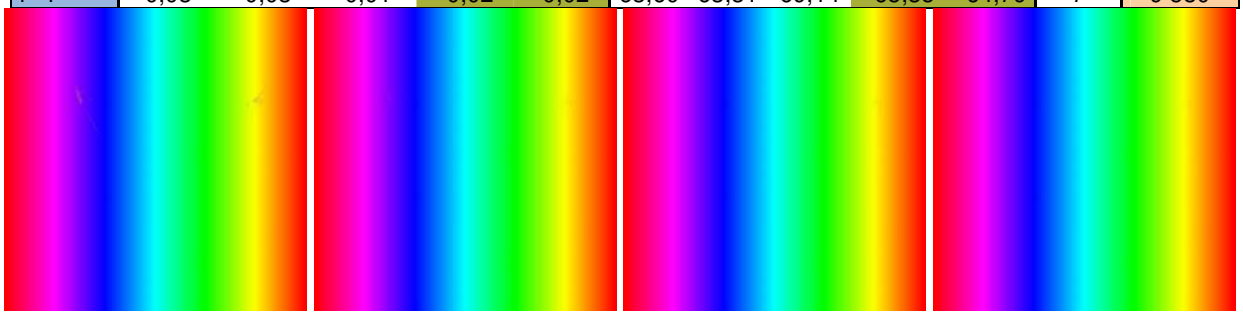
Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	2,36	4,16	5,13	3,88	1,96	44,41	41,94	41,03	42,46	45,22	31	311
r=2	0,11	0,14	0,09	0,11	0,08	57,77	56,76	58,60	57,71	58,93	31	904
r=3	0,05	0,06	0,01	0,04	0,04	60,99	60,35	67,56	62,97	61,84	30	2 953
r=4	0,04	0,04	0,01	0,03	0,03	62,01	61,88	69,24	64,38	63,21	30	9 592



r=1 *r=2* *r=3* *r=4*

Tabulka 3.1.6.2.2.L: Vstup: Gradient + Fun, Směr: LeftRighTopBottom

Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	0,85	1,25	1,59	1,23	0,62	48,85	47,15	46,11	47,37	50,20	8	312
r=2	0,07	0,08	0,04	0,07	0,05	59,53	58,97	61,61	60,04	60,87	8	827
r=3	0,04	0,04	0,01	0,03	0,03	62,44	61,65	67,83	63,97	63,22	8	2 939
r=4	0,03	0,03	0,01	0,02	0,02	63,60	63,31	69,14	65,35	64,79	7	9 580



r=1 *r=2* *r=3* *r=4*

Tabulka 3.1.6.2.2.M: Vstup: Gradient + Grid, Směr: LeftRight

Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	0,18	0,17	0,17	0,17	0,09	55,64	55,86	55,77	55,75	58,60	511	8 936
r=2	0,15	0,15	0,15	0,15	0,07	56,31	56,44	56,47	56,41	59,77	509	22 689
r=3	0,38	0,39	0,38	0,38	0,24	52,30	52,28	52,29	52,29	54,34	507	83 291
r=4	0,32	0,32	0,32	0,32	0,19	53,08	53,04	53,05	53,06	55,25	505	215 578

Tabulka 3.1.6.2.2.N: Vstup: Gradient + Grid, Směr: LeftRighTopBottom

Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	0,18	0,17	0,17	0,17	0,09	55,64	55,86	55,77	55,76	58,60	1	5 375
r=2	0,15	0,15	0,15	0,15	0,07	56,31	56,44	56,47	56,41	59,77	1	20 187
r=3	0,38	0,39	0,38	0,38	0,24	52,29	52,28	52,29	52,29	54,34	1	84 047
r=4	0,32	0,32	0,32	0,32	0,19	53,09	53,04	53,05	53,06	55,25	1	218 969

Tabulka 3.1.6.2.2.O: Vstup: Gradient + Noise45,815%, Směr: LeftRight

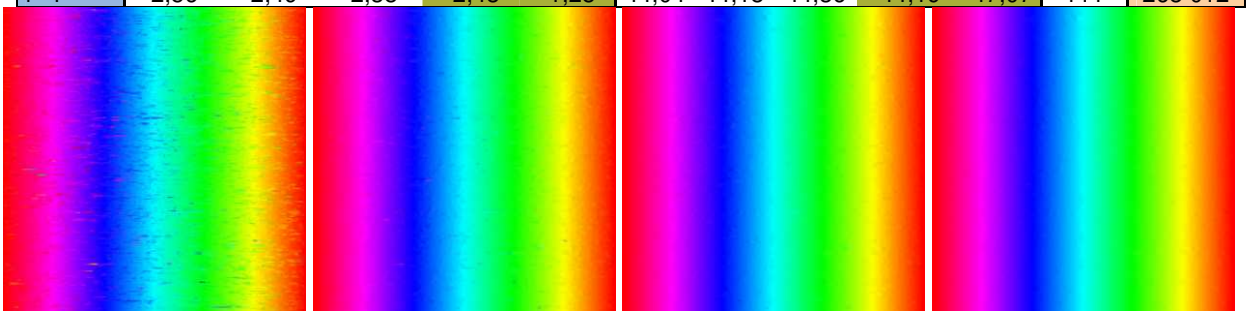
Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	0,10	0,10	0,10	0,10	0,05	58,17	58,20	58,23	58,20	61,45	8	3 485
r=2	0,16	0,16	0,16	0,16	0,09	56,00	56,07	56,03	56,03	58,45	8	18 344
r=3	0,20	0,19	0,20	0,20	0,12	55,21	55,23	55,21	55,22	57,43	8	73 671
r=4	0,12	0,12	0,12	0,12	0,06	57,30	57,41	57,37	57,36	60,22	7	209 906

Tabulka 3.1.6.2.2.P: Vstup: Gradient + Noise45,815%, Směr: LeftRighTopBottom

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	0,10	0,10	0,10	0,10	0,05	58,25	58,29	58,30	58,28	61,51	2	3 453
r=2	0,16	0,16	0,16	0,16	0,09	55,97	56,02	55,99	55,99	58,40	2	17 578
r=3	0,20	0,19	0,19	0,19	0,12	55,22	55,26	55,24	55,24	57,46	2	67 015
r=4	0,12	0,12	0,12	0,12	0,06	57,29	57,40	57,35	57,34	60,20	1	205 422

Tabulka 3.1.6.2.2.Q: Vstup: Gradient + Noise95,503%, Směr: LeftRight

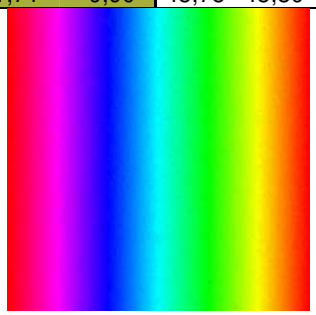
Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	263,60	163,78	261,28	229,55	57,70	23,92	25,99	23,96	24,62	30,52	112	6 568
r=2	23,03	14,27	24,18	20,50	6,79	34,51	36,59	34,30	35,13	39,81	112	23 449
r=3	5,82	4,77	6,11	5,57	2,43	40,48	41,34	40,27	40,70	44,28	111	86 674
r=4	2,56	2,49	2,38	2,48	1,28	44,04	44,18	44,36	44,19	47,07	111	268 012



r=1 *r=2* *r=3* *r=4*

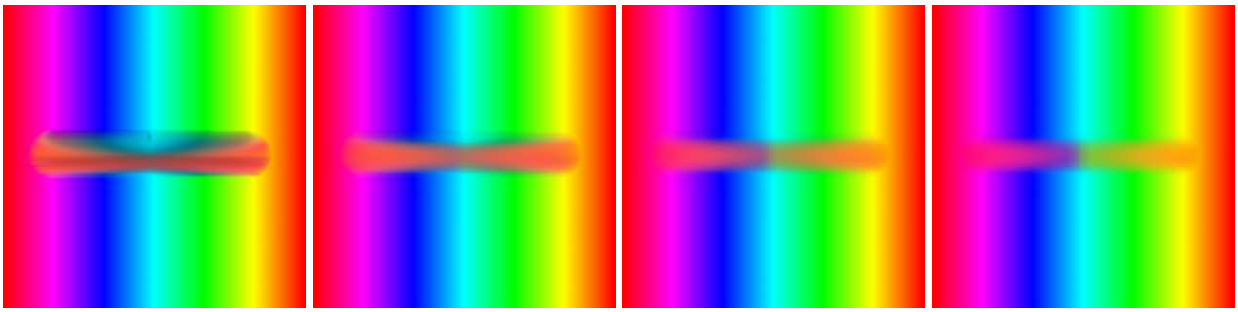
Tabulka 3.1.6.2.2.R: Vstup: Gradient + Noise95,503%, Směr: leftRighTopBottom

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	14,53	11,73	14,42	13,56	4,74	36,51	37,44	36,54	36,83	41,38	9	5 906
r=2	5,52	5,00	5,61	5,38	2,32	40,71	41,14	40,64	40,83	44,48	9	23 096
r=3	2,97	2,76	2,93	2,88	1,39	43,40	43,72	43,47	43,53	46,69	8	82 624
r=4	1,73	1,71	1,69	1,71	0,90	45,75	45,80	45,85	45,80	48,59	8	254 078



Tabulka 3.1.6.2.2.S: Vstup: Gradient +Wide, Směr: LeftRight

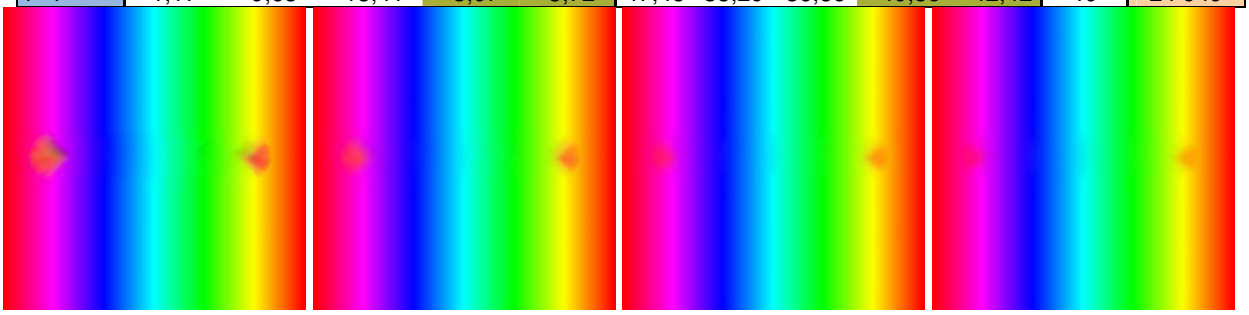
Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	736,05	1358,87	1330,68	1141,87	280,49	19,46	16,80	16,89	17,72	23,65	206	2 452
r=2	1257,00	893,66	981,44	1044,03	175,36	17,14	18,62	18,21	17,99	25,69	206	3 984
r=3	1080,64	460,29	607,17	716,03	91,81	17,79	21,50	20,30	19,86	28,50	205	9 844
r=4	847,77	232,30	362,34	480,80	48,36	18,85	24,47	22,54	21,95	31,29	205	24 532



r=1 *r=2* *r=3* *r=4*

Tabulka 3.1.6.2.2.T: Vstup: Gradient +Wide, Směr: LeftRighTopBottom

Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	9,25	118,82	152,64	93,57	27,19	38,47	27,38	26,29	30,72	33,79	41	1 391
r=2	1,07	41,28	51,32	31,22	10,42	47,83	31,97	31,03	36,94	37,95	41	4 028
r=3	0,99	16,00	21,80	12,93	5,02	48,18	36,09	34,75	39,67	41,12	41	8 953
r=4	1,17	9,63	13,41	8,07	3,72	47,45	38,29	36,86	40,86	42,42	40	24 949



r=1 *r=2* *r=3* *r=4*

C. Testovaci obraz Lena

Tabulka 3.1.6.2.2.U: Vstup: Lena + Fun, Směr: LeftRight

Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	5,40	9,06	4,62	6,36	6,68	40,81	38,56	41,48	40,28	39,89	31	343
r=2	2,95	6,87	4,16	4,66	4,72	43,43	39,76	41,94	41,71	41,39	31	933
r=3	2,67	6,44	4,11	4,41	4,41	43,86	40,04	42,00	41,97	41,68	30	2 988
r=4	2,43	6,26	4,08	4,26	4,22	44,27	40,17	42,03	42,15	41,88	30	9 578

Tabulka 3.1.6.2.2.V: Vstup: Lena + Fun, Směr: LeftRighTopBottom

Poloměr okolí	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS		
r=1	4,53	8,73	4,47	5,91	6,27	41,57	38,72	41,63	40,64	40,16	8	281
r=2	2,89	6,83	4,17	4,63	4,70	43,52	39,79	41,93	41,75	41,41	8	845
r=3	2,49	6,22	4,01	4,24	4,23	44,17	40,19	42,10	42,15	41,87	8	2 983
r=4	2,37	6,08	3,98	4,14	4,11	44,38	40,29	42,13	42,27	42,00	7	9 359

Tabulka 3.1.6.2.2.W: Vstup: Lena + Grid, Směr: LeftRight

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	22,66	45,70	54,30	40,89	31,88	34,58	31,53	30,78	32,30	33,10	511	8 787
r=2	20,95	42,93	52,77	38,89	29,69	34,92	31,80	30,91	32,54	33,40	509	24 051
r=3	19,84	41,21	52,49	37,85	28,29	35,15	31,98	30,93	32,69	33,61	507	86 891
r=4	19,85	41,25	52,45	37,85	28,31	35,15	31,98	30,93	32,69	33,61	505	219 915

Tabulka 3.1.6.2.2.X: Vstup: Lena + Grid, Směr: LeftRighTopBottom

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	22,66	45,70	54,30	40,89	31,88	34,58	31,53	30,78	32,30	33,10	1	5 500
r=2	20,95	42,93	52,77	38,88	29,69	34,92	31,80	30,91	32,54	33,40	1	20 484
r=3	19,84	41,21	52,49	37,85	28,29	35,15	31,98	30,93	32,69	33,61	1	86 391
r=4	19,85	41,25	52,45	37,85	28,31	35,15	31,98	30,93	32,69	33,61	1	230 062

Tabulka 3.1.6.2.2.Y: Vstup: Lena + Noise45,815%, Směr: LeftRight

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	13,39	26,88	31,03	23,77	18,74	36,86	33,84	33,21	34,64	35,40	8	3 671
r=2	11,13	23,16	29,27	21,19	15,79	37,67	34,48	33,47	35,21	36,15	8	19 111
r=3	10,88	22,86	29,28	21,01	15,54	37,77	34,54	33,47	35,26	36,22	8	75 998
r=4	10,87	22,87	29,28	21,01	15,54	37,77	34,54	33,47	35,26	36,22	7	225 719

Tabulka 3.1.6.2.2.Z: Vstup: Lena + Noise45,815%, Směr: LeftRighTopBottom

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	13,39	26,92	31,07	23,79	18,76	36,86	33,83	33,21	34,63	35,40	2	3 500
r=2	11,12	23,15	29,27	21,18	15,78	37,67	34,49	33,47	35,21	36,15	2	18 047
r=3	10,88	22,86	29,28	21,01	15,54	37,77	34,54	33,47	35,26	36,22	2	72 843
r=4	10,87	22,87	29,28	21,01	15,55	37,77	34,54	33,47	35,26	36,21	1	223 969

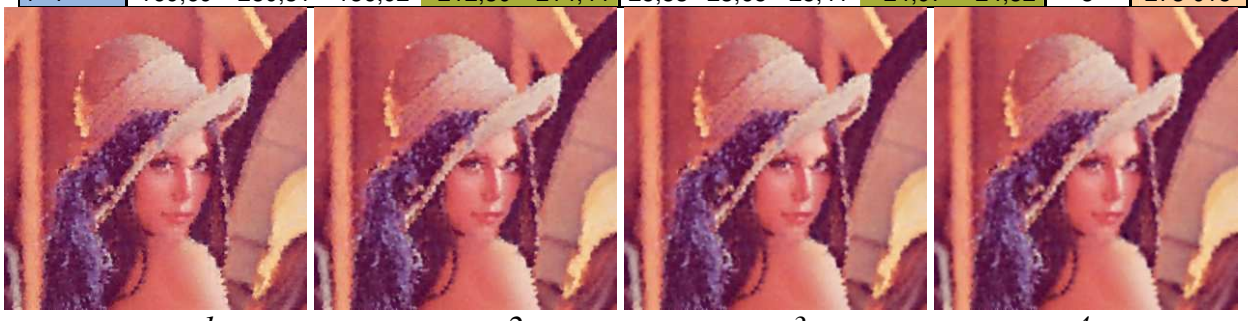
Tabulka 3.1.6.2.2.AA: Vstup: Lena + Noise95,503%, Směr: LeftRight

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	423,15	592,59	311,15	442,30	471,91	21,87	20,40	23,20	21,82	21,39	112	8 195
r=2	261,97	402,74	237,54	300,75	313,72	23,95	22,08	24,37	23,47	23,17	112	25 425
r=3	206,61	330,52	204,81	247,31	254,59	24,98	22,94	25,02	24,31	24,07	111	96 560
r=4	180,63	294,23	190,67	221,84	225,62	25,56	23,44	25,33	24,78	24,60	111	289 944

r=1 *r=2* *r=3* *r=4*

Tabulka 3.1.6.2.2.AB: Vstup: Lena + Noise95,503%, Směr: LeftRighTopBottom

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	208,17	328,21	208,67	248,35	253,87	24,95	22,97	24,94	24,28	24,08	9	6 015
r=2	194,95	315,65	203,45	238,02	242,64	25,23	23,14	25,05	24,47	24,28	9	25 141
r=3	178,99	292,82	192,71	221,51	224,46	25,60	23,46	25,28	24,78	24,62	8	90 484
r=4	169,69	280,31	186,92	212,30	214,44	25,83	23,65	25,41	24,97	24,82	8	273 015



r=1 *r=2* *r=3* *r=4*

Tabulka 3.1.6.2.2.AC: Vstup: Lena + Wide, Směr: LeftRight

Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	298,72	326,90	135,55	253,72	274,51	23,38	22,99	26,81	24,39	23,75	206	2 601
r=2	305,67	355,14	126,28	262,36	295,94	23,28	22,63	27,12	24,34	23,42	206	4 327
r=3	275,28	327,43	120,97	241,22	271,91	23,73	22,98	27,30	24,67	23,79	205	10 648
r=4	251,40	300,17	116,10	222,56	249,43	24,13	23,36	27,48	24,99	24,16	205	28 176

r=1 *r=2* *r=3* *r=4*

Tabulka 3.1.6.2.2.AD: Vstup: Lena + Wide, Směr: LeftRighTopBottom

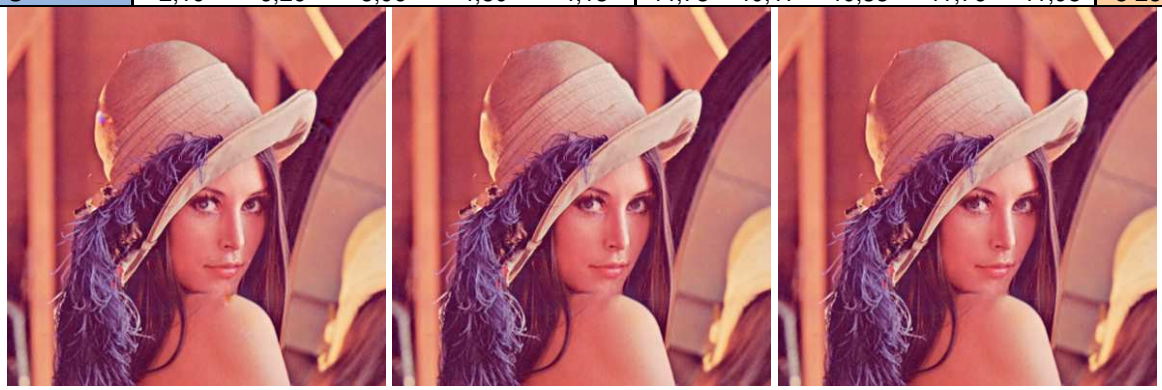
Poloměr	MSE					PSNR [dB]						Čas
okolí	R	G	B	RGB / 3	GS	R	G	B	RGB / 3	GS	Iterací	[ms]
r=1	161,19	217,31	106,81	161,77	174,21	26,06	24,76	27,84	26,22	25,72	41	1 518
r=2	165,70	212,02	104,73	160,82	173,31	25,94	24,87	27,93	26,24	25,74	41	4 455
r=3	164,11	206,20	102,73	157,68	169,45	25,98	24,99	28,01	26,33	25,84	41	10 160
r=4	157,68	200,66	101,49	153,28	164,55	26,15	25,11	28,07	26,44	25,97	40	30 144

r=1 *r=2* *r=3* *r=4*

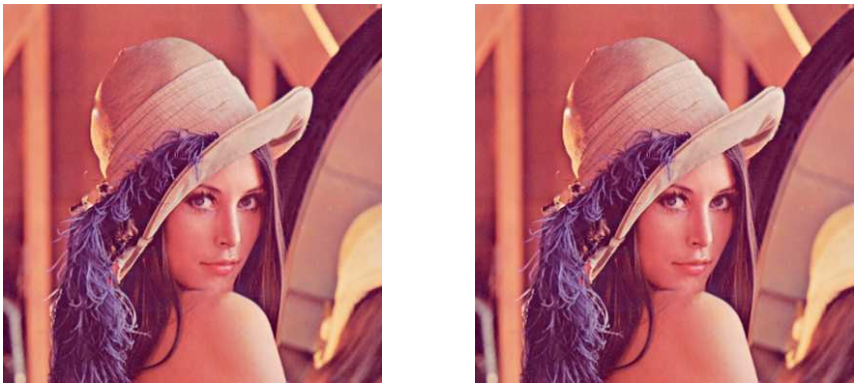
Volba radiální bázové funkce (kapitola 3.1.6.3, strana 53)

Tabulka 3.1.6.3.A: Vstup: Lena + Fun, Výběr okolí: AddWeightedAveragePoints

Bázová	MSE					PSNR [dB]					Čas
funkce	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	[ms]
Cubic	3,56	8,02	11,27	7,62	5,07	42,61	39,09	37,61	39,77	41,08	2 998
Gauss	4,42	8,85	4,60	5,96	6,37	41,67	38,66	41,51	40,61	40,09	3 081
Linear	2,49	6,22	4,01	4,24	4,23	44,17	40,19	42,10	42,15	41,87	2 860
Multiquadric	2,24	5,76	4,84	4,28	3,80	44,62	40,53	41,28	42,14	42,33	2 968
TPS	2,19	6,26	5,96	4,80	4,13	44,73	40,17	40,38	41,76	41,98	3 250



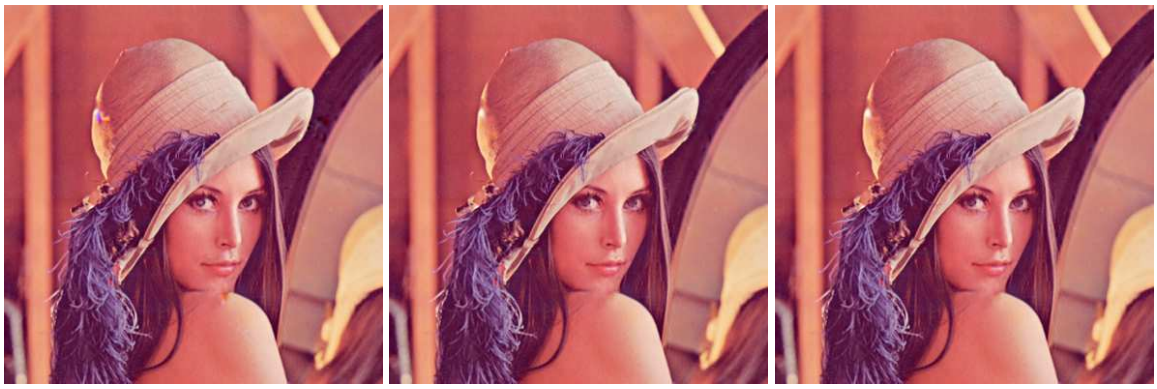
Cubic *Gauss* *Linear*



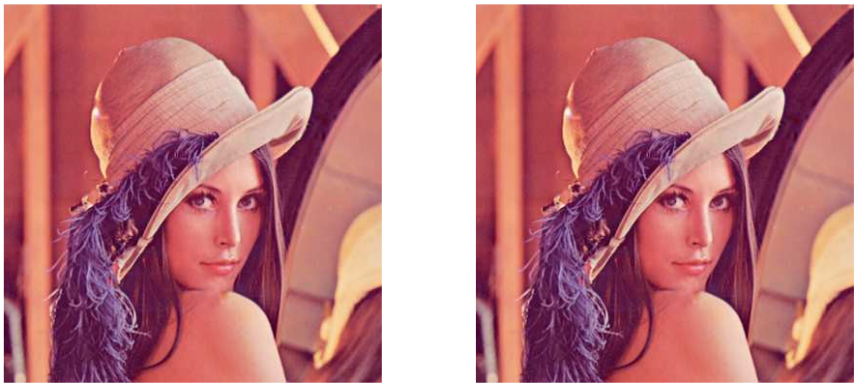
Multiquadric *TPS*

Tabulka 3.1.6.3.B: Vstup: Lena + Fun, Výběr okolí: NoAddingPoints

Bázová	MSE					PSNR [dB]					Čas
funkce	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	[ms]
Cubic	3,86	8,77	12,22	8,28	5,51	42,27	38,70	37,26	39,41	40,72	2 937
Gauss	3,26	7,66	5,71	5,55	5,37	42,99	39,29	40,56	40,95	40,83	3 094
Linear	2,35	6,85	6,11	5,10	4,60	44,42	39,78	40,27	41,49	41,50	2 797
Multiquadric	2,34	6,48	7,59	5,47	4,34	44,44	40,02	39,33	41,26	41,76	2 955
TPS	2,35	6,74	7,64	5,58	4,52	44,41	39,85	39,30	41,19	41,58	3 187



Cubic *Gauss* *Linear*



Multiquadric *TPS*

Tabulka 3.1.6.3.C: Vstup: Lena + Grid, Výběr okolí: AddWeightedAveragePoints

Bázová funkce	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Cubic	18,24	38,63	56,46	37,77	25,81	35,52	32,26	30,61	32,80	34,01	90 843
Gauss	30,79	58,23	58,04	49,02	41,91	33,25	30,48	30,49	31,41	31,91	93 125
Linear	19,84	41,21	52,49	37,85	28,29	35,15	31,98	30,93	32,69	33,61	84 140
Multiquadric	18,02	38,16	55,04	37,07	25,54	35,57	32,32	30,72	32,87	34,06	89 187
TPS	18,04	38,12	54,12	36,76	25,60	35,57	32,32	30,80	32,90	34,05	95 781

Tabulka 3.1.6.3.D: Vstup: Lena + Grid, Výběr okolí: NoAddingPoints

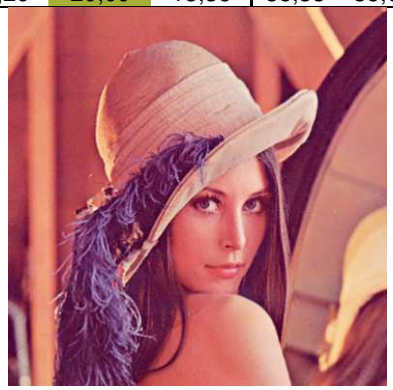
Bázová funkce	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Cubic	18,24	38,63	56,46	37,78	25,81	35,52	32,26	30,61	32,80	34,01	88 375
Gauss	30,70	58,18	58,07	48,98	41,85	33,26	30,48	30,49	31,41	31,91	92 032
Linear	19,84	41,22	52,52	37,86	28,30	35,15	31,98	30,93	32,69	33,61	84 578
Multiquadric	18,03	38,16	55,06	37,08	25,55	35,57	32,31	30,72	32,87	34,06	89 312
TPS	18,04	38,12	54,13	36,76	25,61	35,57	32,32	30,80	32,89	34,05	95 515

Tabulka 3.1.6.3.E: Vstup: Lena + Noise45,815%, Výběr okolí: AddWeightedAveragePoints

Bázová funkce	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Cubic	9,87	20,99	33,32	21,39	13,46	38,19	34,91	32,90	35,33	36,84	68 281
Gauss	16,14	30,25	31,98	26,12	21,48	36,05	33,32	33,08	34,15	34,81	73 625
Linear	10,88	22,86	29,28	21,01	15,54	37,77	34,54	33,47	35,26	36,22	68 578
Multiquadric	9,71	20,61	32,29	20,87	13,25	38,26	34,99	33,04	35,43	36,91	72 266
TPS	9,55	20,42	30,29	20,09	13,35	38,33	35,03	33,32	35,56	36,88	78 344

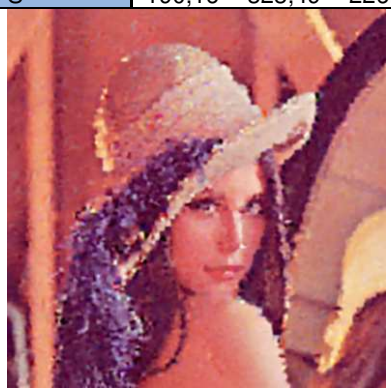
Tabulka 3.1.6.3.F: Vstup: Lena + Noise45,815%, Výběr okolí: NoAddingPoints

Bázová funkce	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Cubic	9,87	20,99	33,32	21,39	13,46	38,19	34,91	32,90	35,33	36,84	71 328
Gauss	16,14	30,25	31,98	26,12	21,48	36,05	33,32	33,08	34,15	34,81	74 062
Linear	10,88	22,86	29,28	21,01	15,54	37,76	34,54	33,47	35,26	36,21	68 204
Multiquadric	9,71	20,61	32,29	20,87	13,25	38,26	34,99	33,04	35,43	36,91	72 250
TPS	9,55	20,42	30,29	20,09	13,35	38,33	35,03	33,32	35,56	36,88	78 937

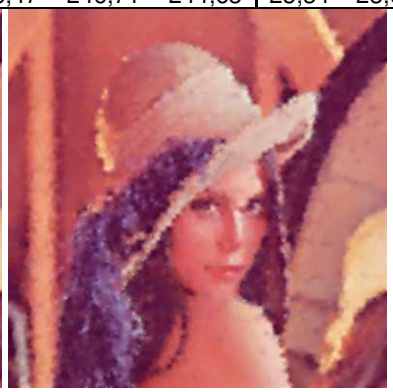


Tabulka 3.1.6.3.G: Vstup: Lena + Noise95,503%, Výběr okolí: AddWeightedAveragePoints

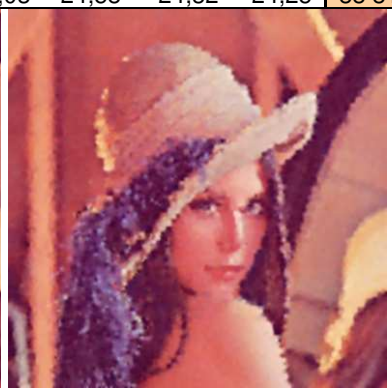
Bázová funkce	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Cubic	237,75	397,26	295,21	310,07	298,49	24,37	22,14	23,43	23,31	23,38	81 828
Gauss	215,43	327,93	199,89	247,75	256,20	24,80	22,97	25,12	24,30	24,04	83 501
Linear	178,99	292,82	192,71	221,51	224,46	25,60	23,46	25,28	24,78	24,62	76 594
Multiquadric	181,03	302,92	205,44	229,80	230,67	25,55	23,32	25,00	24,62	24,50	81 628
TPS	190,16	323,49	226,47	246,71	244,63	25,34	23,03	24,58	24,32	24,25	88 812



Cubic



Gauss



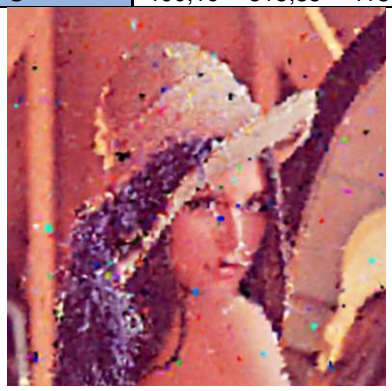
Linear

Multiquadric

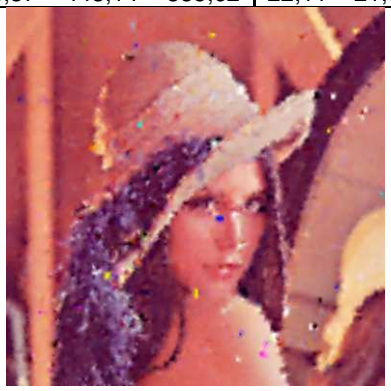
TPS

Tabulka 3.1.6.3.H: Vstup: Lena + Noise95,503%, Výběr okolí: NoAddingPoints

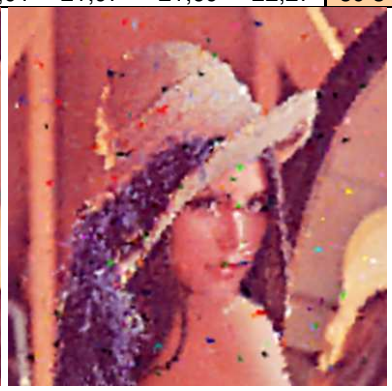
Bázová funkce	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Cubic	485,56	588,12	497,18	523,62	442,57	21,27	20,44	21,17	20,96	21,67	81 251
Gauss	252,49	401,91	292,23	315,54	300,80	24,11	22,09	23,47	23,22	23,35	81 657
Linear	352,42	447,09	364,48	388,00	345,02	22,66	21,63	22,51	22,27	22,75	74 905
Multiquadric	399,05	490,22	422,09	437,12	374,57	22,12	21,23	21,88	21,74	22,40	86 876
TPS	400,19	515,85	413,37	443,14	385,62	22,11	21,01	21,97	21,69	22,27	89 343



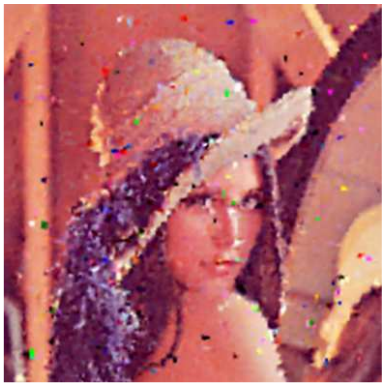
Cubic



Gauss



Linear



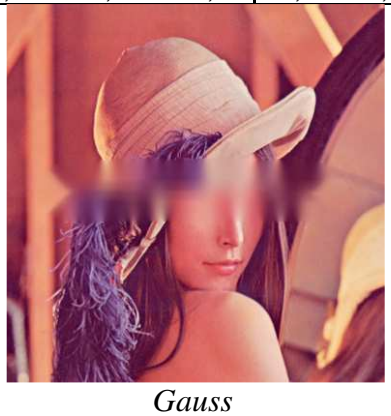
Multiquadric

TPS

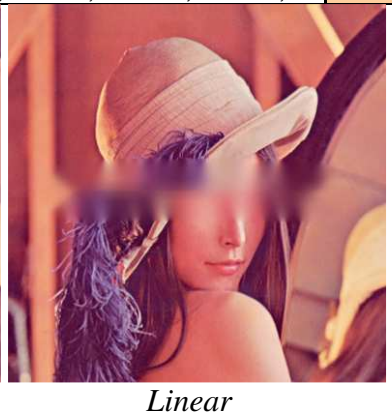
Tabulka 3.1.6.3.I: Vstup: Lena + Wide, Výběr okolí: AddWeightedAveragePoints

Bázová funkce	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Cubic	257,46	314,54	286,72	286,24	239,77	24,02	23,15	23,56	23,58	24,33	9 328
Gauss	172,29	219,82	106,50	166,20	179,79	25,77	24,71	27,86	26,11	25,58	9 746
Linear	164,11	206,20	102,73	157,68	169,45	25,98	24,99	28,01	26,33	25,84	8 924
Multiquadric	168,09	209,10	107,21	161,47	172,55	25,88	24,93	27,83	26,21	25,76	9 514
TPS	166,95	205,79	117,37	163,37	169,93	25,90	25,00	27,44	26,11	25,83	10 084

Cubic

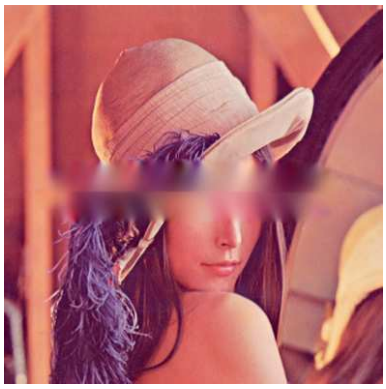


Gauss



Linear

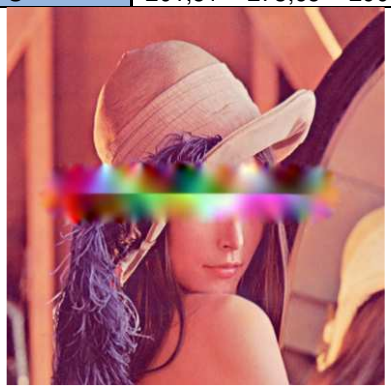
Multiquadric



TPS

Tabulka 3.1.6.3.J: Vstup: Lena + Wide, Výběr okolí: NoAddingPoints

Bázová funkce	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Cubic	381,11	583,24	525,20	496,52	354,70	22,32	20,47	20,93	21,24	22,63	9 799
Gauss	181,59	265,08	204,72	217,13	210,07	25,54	23,90	25,02	24,82	24,91	9 078
Linear	171,17	238,23	172,55	193,98	188,22	25,80	24,36	25,76	25,31	25,38	8 355
Multiquadric	194,57	285,71	288,87	256,38	219,66	25,24	23,57	23,52	24,11	24,71	8 912
TPS	201,51	273,65	290,71	255,29	212,39	25,09	23,76	23,50	24,11	24,86	9 515



Cubic

Gauss

Linear

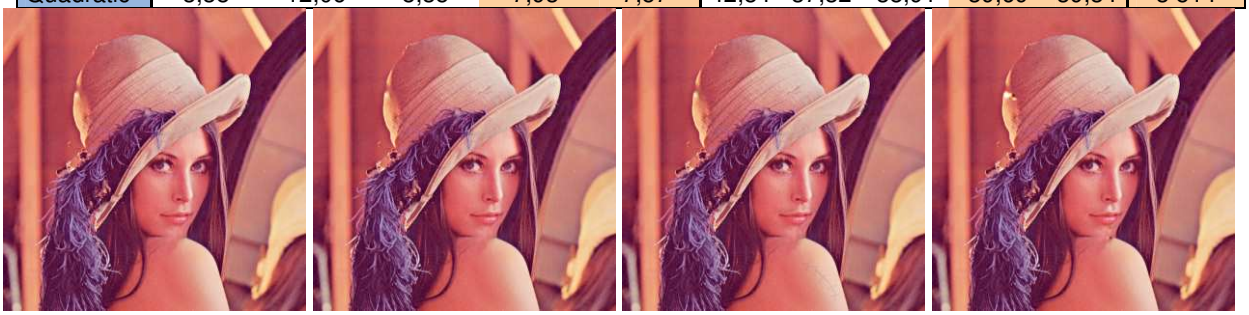
Multiquadric

TPS

Vliv rozšiřujícího elementu (kapitola 3.1.6.4, strana 54)

Tabulka 3.1.6.4.A: Vstup: Lena + Fun, Výběr okolí: AddWeightedAveragePoints

Rozšiřující element	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Constant	2,53	6,24	3,95	4,24	4,25	44,10	40,18	42,16	42,14	41,84	2 596
Linear	2,49	6,22	4,01	4,24	4,23	44,17	40,19	42,10	42,15	41,87	2 828
None	3,89	6,72	4,35	4,98	4,93	42,23	39,86	41,74	41,28	41,21	2 437
Quadratic	3,38	12,06	8,35	7,93	7,57	42,84	37,32	38,91	39,69	39,34	3 514



Constant *Linear* *None* *Quadratic*

Tabulka 3.1.6.4.B: Vstup: Lena + Fun, Výběr okolí: NoAddingPoints

Rozšiřující element	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Constant	2,51	6,36	4,03	4,30	4,30	44,13	40,10	42,08	42,10	41,79	2 594
Linear	2,35	6,85	6,11	5,10	4,60	44,42	39,78	40,27	41,49	41,50	2 828
None	11,47	23,13	25,52	20,04	16,72	37,54	34,49	34,06	35,36	35,90	2 406
Quadratic	11,40	14,76	20,37	15,51	10,22	37,56	36,44	35,04	36,35	38,04	3 471

Constant *Linear* *None* *Quadratic*

Tabulka 3.1.6.4.C: Vstup: Lena + Grid, Výběr okolí: AddWeightedAveragePoints

Rozšiřující element	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Constant	19,85	41,22	52,49	37,85	28,30	35,15	31,98	30,93	32,69	33,61	80 562
Linear	19,84	41,21	52,49	37,85	28,29	35,15	31,98	30,93	32,69	33,61	86 328
None	25,78	43,59	55,19	41,52	31,58	34,02	31,74	30,71	32,16	33,14	78 500
Quadratic	19,59	41,04	53,01	37,88	28,03	35,21	32,00	30,89	32,70	33,65	105 047

Tabulka 3.1.6.4.D: Vstup: Lena + Grid, Výběr okolí: NoAddingPoints

Rozšiřující element	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Constant	19,85	41,22	52,49	37,86	28,30	35,15	31,98	30,93	32,69	33,61	78 812
Linear	19,84	41,22	52,52	37,86	28,30	35,15	31,98	30,93	32,69	33,61	82 516
None	26,83	44,05	55,55	42,14	32,14	33,84	31,69	30,68	32,07	33,06	79 938
Quadratic	19,60	41,06	53,06	37,91	28,05	35,21	32,00	30,88	32,70	33,65	125 046

Tabulka 3.1.6.4.E: Vstup: Lena + Noise45,815%, Výběr okolí: AddWeightedAveragePoints

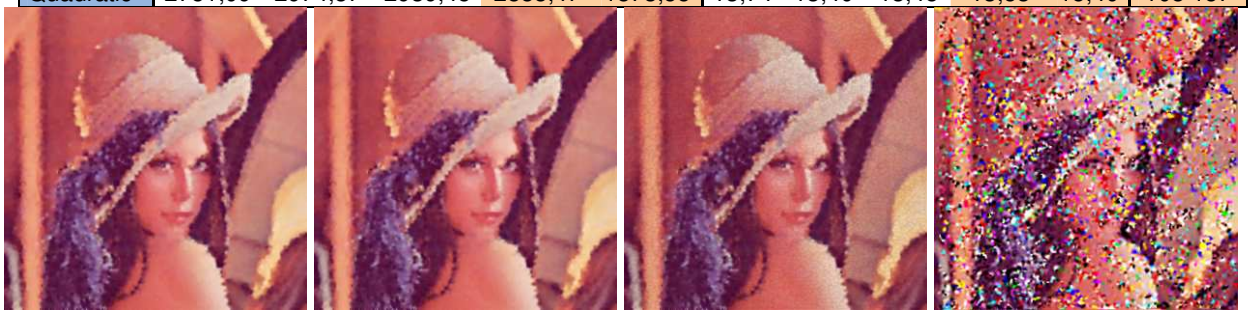
Rozšiřující	MSE					PSNR [dB]					Čas
element	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	[ms]
Constant	10,88	22,86	29,28	21,00	15,54	37,77	34,54	33,47	35,26	36,22	62 734
Linear	10,88	22,86	29,28	21,01	15,54	37,77	34,54	33,47	35,26	36,22	68 047
None	12,16	23,41	29,86	21,81	16,27	37,28	34,44	33,38	35,03	36,02	59 516
Quadratic	10,78	22,81	29,43	21,01	15,48	37,80	34,55	33,44	35,27	36,23	83 765

Tabulka 3.1.6.4.F: Vstup: Lena + Noise45,815%, Výběr okolí: NoAddingPoints

Rozšiřující	MSE					PSNR [dB]					Čas
element	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	[ms]
Constant	10,88	22,86	29,28	21,00	15,54	37,77	34,54	33,47	35,26	36,22	63 469
Linear	10,88	22,86	29,28	21,01	15,54	37,76	34,54	33,47	35,26	36,21	73 749
None	12,21	23,43	29,87	21,84	16,30	37,26	34,43	33,38	35,02	36,01	61 281
Quadratic	10,79	22,81	29,43	21,01	15,48	37,80	34,55	33,44	35,26	36,23	88 843

Tabulka 3.1.6.4.G: Vstup: Lena + Noise95,503%, Výběr okolí: AddWeightedAveragePoints

Rozšiřující	MSE					PSNR [dB]					Čas
element	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	[ms]
Constant	180,10	293,39	192,44	221,98	225,18	25,58	23,46	25,29	24,77	24,61	76 389
Linear	178,99	292,82	192,71	221,51	224,46	25,60	23,46	25,28	24,78	24,62	86 216
None	282,62	334,85	231,73	283,07	280,86	23,62	22,88	24,48	23,66	23,65	68 986
Quadratic	2751,09	2974,87	2939,45	2888,47	1873,55	13,74	13,40	13,45	13,53	15,40	108 157



Constant *Linear* *None* *Quadratic*

Tabulka 3.1.6.4.H: Vstup: Lena + Noise95,503%, Výběr okolí: NoAddingPoints

Rozšiřující	MSE					PSNR [dB]					Čas
element	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	[ms]
Constant	180,38	294,84	193,74	222,99	226,07	25,57	23,43	25,26	24,75	24,59	73 422
Linear	352,42	447,09	364,48	388,00	345,02	22,66	21,63	22,51	22,27	22,75	84 266
None	2262,78	3708,99	4481,31	3484,36	3022,50	14,58	12,44	11,62	12,88	13,33	70 232
Quadratic	4357,33	3923,57	3802,27	4027,73	2526,51	11,74	12,19	12,33	12,09	14,11	105 046

Constant *Linear* *None* *Quadratic*

Tabulka 3.1.6.4.I: Vstup: Lena + Wide, Výběr okolí: AddWeightedAveragePoints

Rozšiřující element	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Constant	175,49	214,65	102,75	164,30	178,04	25,69	24,81	28,01	26,17	25,63	8 263
Linear	164,11	206,20	102,73	157,68	169,45	25,98	24,99	28,01	26,33	25,84	9 677
None	322,98	244,27	155,21	240,82	237,93	23,04	24,25	26,22	24,50	24,37	8 351
Quadratic	239,71	295,89	181,07	238,89	241,23	24,33	23,42	25,55	24,44	24,31	13 002

Constant *Linear* *None* *Quadratic*

Tabulka 3.1.6.4.J: Vstup: Lena + Wide, Výběr okolí: NoAddingPoints

Rozšiřující element	MSE					PSNR [dB]					Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS	
Constant	191,21	220,92	103,13	171,75	185,56	25,32	24,69	28,00	26,00	25,45	8 196
Linear	171,17	238,23	172,55	193,98	188,22	25,80	24,36	25,76	25,31	25,38	9 222
None	1161,61	2938,42	2481,64	2193,89	2209,21	17,48	13,45	14,18	15,04	14,69	7 813
Quadratic	863,48	1128,01	981,04	990,84	593,14	18,77	17,61	18,21	18,20	20,40	12 428

Constant *Linear* *None* *Quadratic*

Volba algoritmu (kapitola 3.1.6.5, strana 54)

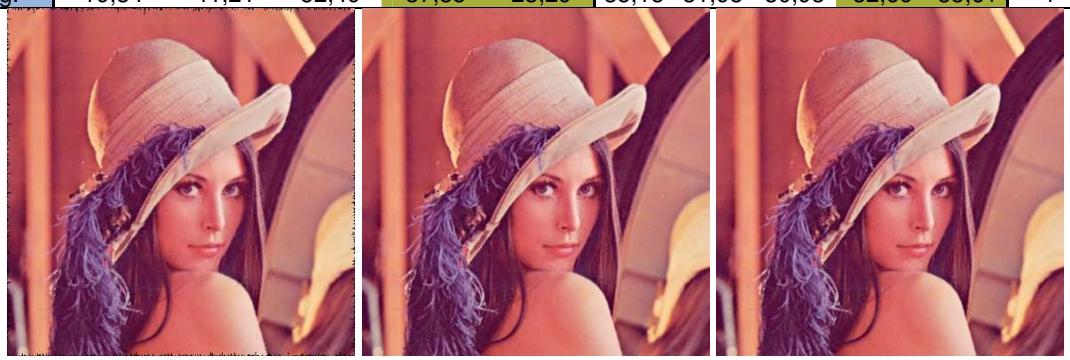
Tabulka 3.1.6.5.A: Vstup: Lena + Fun

Algoritmus	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS		
Lerp	4,27	8,75	4,97	6,00	6,26	41,83	38,71	41,17	40,57	40,17	60	299
Uhlíř	2,82	7,59	7,67	6,02	5,22	43,63	39,33	39,28	40,75	40,96	9	3 158
Náš alg.	2,49	6,22	4,01	4,24	4,23	44,17	40,19	42,10	42,15	41,87	8	3 218

Bilineární interpolace *Uhlíř* *Náš algoritmus*

Tabulka 3.1.6.5.B: Vstup: Lena + Noise45,815%

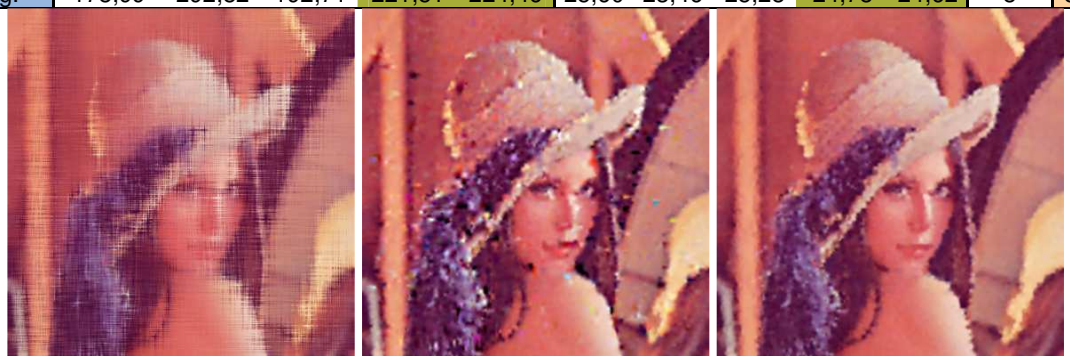
Algoritmus	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS		
Lerp	17377,92	6277,42	6111,96	9922,43	8803,68	5,73	10,15	10,27	8,72	8,68	256	304 311
Uhlíř	19,85	41,22	52,52	37,86	28,30	35,15	31,98	30,93	32,69	33,61	1	121 015
Náš alg.	19,84	41,21	52,49	37,85	28,29	35,15	31,98	30,93	32,69	33,61	1	87 781



Bilineární interpolace *Uhlíř* *Náš algoritmus*

Tabulka 3.1.6.5.C: Vstup: Lena + Noise95,503%

Algoritmus	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS		
Lerp	488,56	712,44	377,22	526,08	564,04	21,24	19,60	22,36	21,07	20,62	222	1 551
Uhlíř	341,88	494,94	375,38	404,07	383,85	22,79	21,19	22,39	22,12	22,29	152	77 329
Náš alg.	178,99	292,82	192,71	221,51	224,46	25,60	23,46	25,28	24,78	24,62	8	86 717



Bilineární interpolace *Uhlíř* *Náš algoritmus*

Tabulka 3.1.6.5.D: Vstup: Lena + Wide

Algoritmus	MSE					PSNR [dB]					Iterací	Čas [ms]
	R	G	B	RGB/3	GS	R	G	B	RGB/3	GS		
Lerp	186,70	247,50	118,25	184,15	200,59	25,42	24,20	27,40	25,67	25,11	410	2 230
Uhlíř	166,15	253,39	161,60	193,71	193,68	25,93	24,09	26,05	25,35	25,26	41	9 435
Náš alg.	164,11	206,20	102,73	157,68	169,45	25,98	24,99	28,01	26,33	25,84	41	10 782

Bilineární interpolace *Uhlíř* *Náš algoritmus*