

Univerzita Pavla Jozefa Šafárika v Košiciach  
Prírodovedecká fakulta

# ZRÝCHLENIE VÝPOČTU SPLAJN POVRCHOV

ŠVK PRÁCA

Študijný odbor:	Informatika
Školiace pracovisko:	Ústav informatiky
Vedúci záverečnej práce:	doc. RNDr. Csaba Török, CSc.
Konzultant:	RNDr. Lukáš Miňo

Košice 2016

Bc. Viliam Kačala

## **Podakovanie**

Rád by som poďakoval vedúcemu práce doc. RNDr. Csabovi Törökovi, CSc. za cenné pripomienky, odborné vedenie a obetavosť počas tvorby práce. Taktiež by som sa rád poďakoval RNDr. Lukášovi Miňovi za cenné rady a pomoc počas tvorby aplikačnej časti tejto práce.

Zadanie práce

## Abstrakt

Splajny sú dôležitá súčasť počítačovej grafiky. Jedná sa o matematický model krivky a plochy slúžiaci na čo „najlepšie spojenie“ konečnej množiny bodov. Termín „najlepšie spojenie“ v našom prípade znamená hladkú, matematicky ľahko vyjadriteľnú plochu s čo najmenším zakrivením. Využitie splajnov v grafike je veľmi široké od rôznych CAD aplikácií, v štatistike, alebo v analýze dát. Splajny existujú v mnohých formách, či už vo forme krivky v rovine, rôznych trojrozmerných telies, atď.. Táto práca si dáva za cieľ navrhnúť, analyzovať a implementovať nový algoritmus pre bikubickú interpoláciu v trojrozmernom priestore.

## Abstract

Splines are important part of computer graphics. It is a mathematical model of curve and surface for the "best connection" of any finite set of points. The term "best connection" in this case means smooth, easily calculable mathematical surface with minimal curvature. Use of splines in graphics varies from large variety of CAD applications, statistics or in data analysing. Splines exist in many forms, whether in the form of curves in the plane, a variety of three-dimensional bodies, etc.. This work aims to design, analyze and implement a new algorithm for counting and generating splines bicubic clamped interpolation in three-dimensional space.

# Obsah

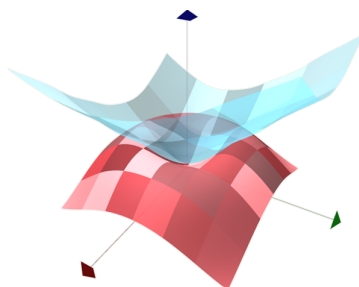
<b>1</b>	<b>De Boorov model a redukovaný algoritmus</b>	<b>6</b>
1.1	Trojdiagonálna LU dekompozícia . . . . .	11
1.2	De Boorov výpočet derivácií . . . . .	14
1.3	Počítanie derivácií redukovanou sústavou . . . . .	16
1.4	Konštrukcia hermitovho splajnu . . . . .	20
<b>2</b>	<b>Zrýchlenie</b>	<b>22</b>
2.1	Procesorová architektúra . . . . .	23
2.1.1	Rýchlosť aritmetických operácií . . . . .	25
2.2	Teoretické zrýchlenie . . . . .	28
2.2.1	Cena plného algoritmu . . . . .	30
2.2.2	Počty pre redukovaný algoritmus . . . . .	32
2.2.3	Pamäťové nároky . . . . .	35
2.2.4	Zhrnutie . . . . .	36
2.3	Merané zrýchlenie . . . . .	37

# Úvod

Témou mojej diplomovej práce sú priestorové splajn povrchy, pričom naším cieľom je preskúmať nové poznatky o splajnoch, na ktorých istý čas pracuje vedúci tejto práce doc. RNDr. Csaba Török, CSc.

Výsledok tejto práce je návrh novej metódy výpočtu derivácií splajnu v jeho uzloch, jej porovnanie s de Boorovým postupom z ktorého vychádzame a vysvetlenie zrýchlenia našej metódy.

Prínos zrýchlenia výpočtu splajnov spočíva v lepších možnostiach modelovania ľubovoľných trojrozmerných útvarov v podobe polynomických funkcií. Akékoľvek zrýchlenie totiž znamená možnosť v reálnom čase modelovať zložitejšie objekty alebo fyzikálne dáta, čo je užitočné nielen vzhľadom na vizuálnu reprezentáciu, ale aj na skúmanie fyzikálnych vlastností ako napríklad aerodynamické vlastnosti lietadiel a podobne.



# Kapitola 1

## De Boorov model a redukovaný algoritmus

V našej práci pracujeme s hermitovskými splajnami, ktoré sú štandardne triedy  $C^1$ , teda splajnami, ktorých prvé derivácie v uzloch sa rovnajú.

Najbežnejšie kubické splajny triedy  $C^2$ , teda tie pri ktorých v uzloch máme zaručenú rovnosť aj derivácií druhého rádu, sú naturálne a clamped splajny. Pre naše potreby budeme uvažovať iba druhé menované pričom ich budeme konštruovať použitím hermitovských báзовých funkcií (pozri časť 1.4).

Keďže naším cieľom je zrýchlený algoritmus počítania povrchových splajnov, ktorý vznikol zovšeobecnením algoritmu pre krivkové splajny tak si zaviedieme pojem splajnu pre krivky aj plochy. Iste by bolo možno elegantnejšie zadať splajn všeobecne, ale rozdelenie definície špeciálne pre krivkové splajny v rovine a povrchové v priestore bude čitateľnejšie.

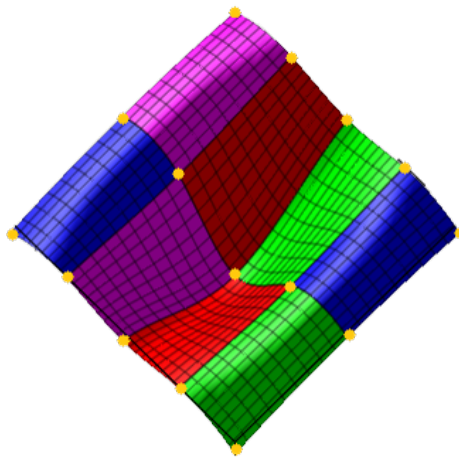
Formálna definícia krivkového splajnu v rovine je nasledovná.

**Definícia 1.1** Nech  $I \geq 0$  je prirodzené číslo,  $(u_0, \dots, u_{I-1})$  je rastúca postupnosť. Nech pre každé  $i$  z  $\{0, 1, \dots, I-2\}$  funkcie  $S_i$  sú polynomicke funkcie, ktoré spĺňajú

- $S_i(u_{i+1}) = S_{i+1}(u_{i+1})$ ,
- $\frac{dS_i}{dx}(u_{i+1}) = \frac{dS_{i+1}}{dx}(u_{i+1})$

Funkciu  $S$  z intervalu  $[u_0, u_{I-1}]$  do  $\mathbb{R}$  pre ktorú platí:

$$S(x) = \begin{cases} S_0(x, y) & \text{pre } x \in [u_0, u_1], \\ S_1(x, y) & \text{pre } x \in [u_1, u_2], \\ \vdots & \\ S_{I-1}(x, y) & \text{pre } x \in [u_{I-2}, u_{I-1}], \end{cases} \quad (1.1)$$



Obr. 1.1: Funkcia  $\sin(\sqrt{x^2 + y^2})$  interpolovaná bikubickým splajnom.

nazveme *krivkový splajn na uzloch*  $(u_0, \dots, u_{I-1})$ .

**Označenie 1.2** Pri označeniach z predchádzajúcej definície označme:

- Funkcie  $S_i$  nazveme *segmenty splajnu*.
- Hodnoty  $u_i$  nazveme *uzly*.
- Uzly  $u_0$  a  $u_{I-1}$  nazveme *krajné uzly*.
- $z_i = S(u_i)$  nazveme *funkčné hodnoty splajnu v uzloch*.
- $d_i^x = \frac{dS(u_i)}{dx}$  nazveme *derivácie v uzloch*.
- $h = u_i - u_{i-1}$  pre ľubovoľné  $i \in \{1, \dots, I-1\}$ .

Na to aby sme zostrojili segmenty splajnu potrebujeme mať dané uzly, funkčné hodnoty v uzloch aj derivácie v uzloch. Je však možné zostrojiť splajn aj keď okrem uzlov a funkčných hodnôt máme známe len dve krajné derivácie  $d_0$  a  $d_{I-1}$ . Zvyšné hodnoty derivácií totiž môžeme s malou odchýlkou vypočítať napríklad De Boorovou interpoláciou. Túto techniku, ale pre splajnové povrchy, si rozoberieme v nasledujúcich častiach kapitoly. Teraz rozšírime túto definíciu na povrchový splajn v priestore.



**Definícia 1.3** Nech  $I \geq 0$  a  $J \geq 0$  sú prirodzené čísla,  $(u_0, \dots, u_{I-1})$  a  $(v_0, \dots, v_{J-1})$  sú rovnomerne rastúce postupnosti. Nech pre každé  $i$  z  $\{0, 1, \dots, I-2\}$  a  $j$  z  $\{0, 1, \dots, J-2\}$  funkcie  $S_{i,j}$  sú bipolynomicke funkcie, ktoré spĺňajú

- $S_{i,j}(u_{i+1}, y) = S_{i+1,j}(u_{i+1}, y),$
- $S_{i,j+1}(x, v_{j+1}) = S_{i,j+1}(x, v_{j+1}),$
- $\frac{\partial S_{i,j}}{\partial x}(u_{i+1}, y) = \frac{\partial S_{i+1,j}}{\partial x}(u_{i+1}, y)$
- $\frac{\partial S_{i,j+1}}{\partial y}(x, v_{j+1}) = \frac{\partial S_{i,j}}{\partial y}(x, v_{j+1}),$

Funkciu  $S$  z intervalu  $[u_0, u_{I-1}] \times [v_0, v_{J-1}]$  do  $\mathbb{R}$  pre ktorú platí:

$$S(x, y) = \begin{cases} S_{0,0}(x, y) & \text{pre } (x, y) \in [u_0, u_1] \times [v_0, v_1], \\ S_{0,1}(x, y) & \text{pre } (x, y) \in [u_0, u_1] \times [v_1, v_2], \\ \vdots & \\ S_{0,J-1}(x, y) & \text{pre } (x, y) \in [u_0, u_1] \times [v_{J-2}, v_{J-1}], \\ S_{1,0}(x, y) & \text{pre } (x, y) \in [u_1, u_2] \times [v_0, v_1], \\ \vdots & \\ S_{1,J-1}(x, y) & \text{pre } (x, y) \in [u_1, u_2] \times [v_{J-2}, v_{J-1}], \\ \vdots & \\ S_{I-1,0}(x, y) & \text{pre } (x, y) \in [u_{I-2}, u_{I-1}] \times [v_0, v_1], \\ \vdots & \\ S_{I-1,J-1}(x, y) & \text{pre } (x, y) \in [u_{I-2}, u_{I-1}] \times [v_{J-2}, v_{J-1}], \end{cases} \quad (1.2)$$

nazveme *splajn na uzloch*  $(u_0, \dots, u_{I-1})$  a  $(v_0, \dots, v_{J-1})$ .

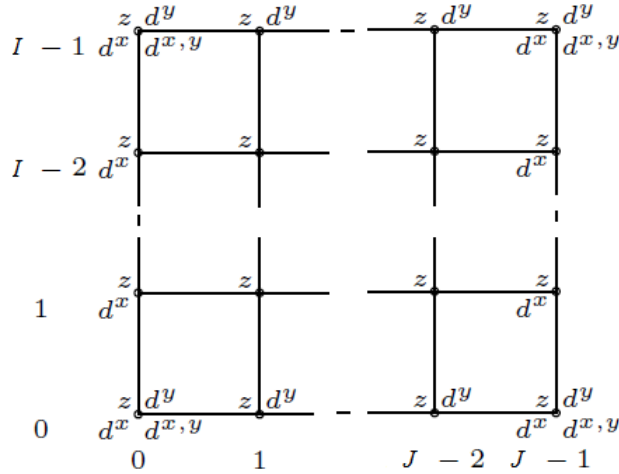
**Označenie 1.4** Pre úplnosť analogicky ako pri krivkách označme:

- Funkcie  $S_{i,j}$  nazveme *segmenty splajnu*.
- Dvojice  $\langle u_i, v_j \rangle$  nazveme *uzly*.
- Uzly  $\langle u_0, v_0 \rangle, \langle u_{I-1}, v_0 \rangle, \langle u_0, v_{J-1} \rangle$  a  $\langle u_{I-1}, v_{J-1} \rangle$  nazveme *rohoové uzly*.
- $z_{i,j} = S(u_i, v_j)$  nazveme *funkčné hodnoty splajnu v uzloch*.
- $d_{i,j}^x = \frac{\partial S(u_i, v_j)}{\partial x}$  nazveme *(smerové) x-ové derivácie v uzloch*.
- $d_{i,j}^y = \frac{\partial S(u_i, v_j)}{\partial y}$  nazveme *(smerové) y-ové derivácie v uzloch*.

- $d_{i,j}^{xy} = \frac{\partial^2 S(u_i, v_j)}{\partial x \partial y}$  nazveme *zmiešané derivácie v uzloch*.
- $h_x = u_i - u_{i-1}$  a  $h_y = v_j - v_{j-1}$  pre ľubovoľné  $i \in \{1, \dots, I-1\}$ , resp.  $j \in \{1, \dots, J-1\}$ .

Na to aby sme zostrojili splajnové segmenty pochopiteľne potrebujeme mať dané všetky uzly aj funkčné hodnoty v uzloch a derivácie v uzloch. Analogicky ako pri krivkách, ak nemáme známe všetky derivácie vieme napriek tomu zvyšné derivácie dopočítať De Boorovou interpoláciou s veľmi malou odchýlkou aby splajn bol spojitý a hladký. Na to aby sme zostrojili splajn pomocou De Boorovho modelu interpolácie potrebujeme mať známe

- postupnosti uzlov  $u_0, \dots, u_{I-1}$  a  $(v_0, \dots, v_{J-1})$ ,
- funkčné hodnoty  $\{z_{0,0}, \dots, z_{I-1,0}, \dots, z_{0,J-1}, \dots, z_{I-1,J-1}\}$ ,
- smerové derivácie  $\{d_{0,0}^x, \dots, d_{I-1,0}^x, d_{0,J-1}^x, \dots, d_{I-1,J-1}^x\}$ ,
- smerové derivácie  $\{d_{0,0}^y, \dots, d_{0,J-1}^y, d_{I-1,0}^y, \dots, d_{I-1,J-1}^y\}$ ,
- zmiešané derivácie  $\{d_{0,0}^{xy}, d_{I-1,0}^{xy}, d_{0,J-1}^{xy}, d_{I-1,J-1}^{xy}\}$ .



Obr. 1.2: De Boorov model vstupných hodnôt pre splajnový povrch.

Kým v prípade splajnových kriviek bolo evidentné hneď od začiatku ich výskumu a aplikácie, že okrem uzlov a funkčných hodnôt  $(z_0, \dots, z_{I-1})$  v uzloch je potrebné zadať ešte dve podmienky, v našom prípade sú to dve hodnoty  $d_0$  a  $d_{I-1}$ . V prípade interpolačných splajnových povrchov nebolo

zrejme ktoré smerové a zmiešané derivácie je potrebné zadať. De Boor navrhol vhodný model, ktorý okrem hodnôt  $z_{i,j}$  na mriežke veľkosti  $I \times J$  vyžaduje zadanie smerových derivácií na okrajoch mriežky uzlov a štyri zmiešané derivácie v rohoch.

Prirodzene vidieť, že takto definovaný krivkový aj povrchový splajn je triedy  $C^1$ . Pri vhodne zvolených hodnotách derivácií v uzloch vieme ale zaručiť rovnosť aj druhých derivácií, čím dostaneme splajn z triedy  $C^2$  čo nám zaručí hladkosť *spojenia* jednotlivých segmentov. Splajny sa taktiež dajú použiť na interpoláciu matematických funkcií.

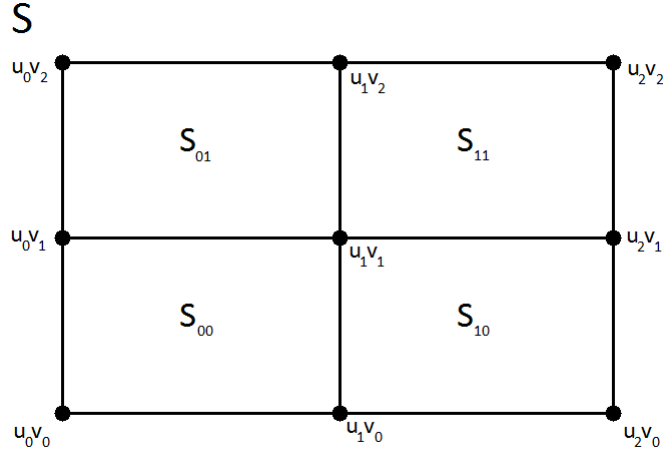
**Definícia 1.5** Nech  $S$  je splajn na uzloch  $(u_0, \dots, u_{I-1})$  a  $(v_0, \dots, v_{J-1})$  a  $f : [u_0, u_{I-1}] \times [v_0, v_{J-1}] \rightarrow \mathbb{R}$  je spojitá funkcia. Hovoríme, že splajn  $S$  interpoluje funkciu  $f$  ak platia tieto podmienky

- $z_{i,j} = f(u_i, v_j)$ , pre každé  $i$  z  $\{0, \dots, I-1\}$  a pre každé  $j$  z  $\{0, \dots, J-1\}$ .
- $d_{i,j}^x = \frac{\partial f(u_i, v_j)}{\partial x}$ , pre každé  $i$  z  $\{0, \dots, I-1\}$  a pre každé  $j$  z  $\{0, \dots, J-1\}$ .
- $d_{i,j}^y = \frac{\partial f(u_i, v_j)}{\partial y}$ , pre každé  $i$  z  $\{0, \dots, I-1\}$  a pre každé  $j$  z  $\{0, \dots, J-1\}$ .
- $d_{i,j}^{xy} = \frac{\partial^2 f(u_i, v_j)}{\partial x \partial y}$ , pre každé  $i$  z  $\{0, \dots, I-1\}$  a pre každé  $j$  z  $\{0, \dots, J-1\}$ .

Na splajn interpolujúcu funkciu  $f$  môžeme aplikovať De Boorovu interpoláciu a zo získaných derivácií už vieme zostrojiť jeho segmenty. Takto získaný splajn bude danú funkciu interpolovať, teda ju *napodobní*. To je žiadané napríklad v grafickom modelovaní, pretože pre počítač je jednoduchšie a rýchlejšie pracovať s polynómom interpolujúcim napríklad goniometrickú funkciu ako priamo s ňou. Obecne splajny minimalizujú integrál druhej derivácie funkcie (napr. zakrivenie, energie, ...). Poskytujú pružný nástroj na modelovanie reálnych situácií na lokálne požiadavky, pričom ich výpočet je rýchly a stabilný. Navyše sa dá ukázať, že interpolácia funkcie kubickým splajnom je jednoznačná. Témou tejto práce sú práve metódy výpočtu prvých derivácií, ktorými je možné dosiahnuť práve spomenutý cieľ.

Priebežné zhrnutie práce je úloha na základe vstupných uzlov  $u_0, \dots, u_{I-1}$ ,  $v_0, \dots, v_{J-1}$  a funkčných hodnôt  $z_{0,0}, \dots, z_{I,J}$  nájsť *hladkú*, po častiach definovanú funkciu  $S : [u_0, u_{I-1}] \times [v_0, v_{J-1}] \rightarrow \mathbb{R}$  so spojitými deriváciami prvého aj druhého rádu takú, že pre každé  $i \in \{0, \dots, I-1\}$  a  $j \in \{0, \dots, J-1\}$  platí  $z_{i,j} = S(u_i, v_j)$ . Funkciu  $S$  nazývame splajn (konkrétne povrchový splajn), pričom jednotlivé časti nazveme *segmenty*.

Ak  $i$  je z  $\{0, \dots, I-2\}$  a  $j$  je z  $\{0, \dots, J-2\}$ , tak štvorice uzlov  $(u_i, v_j)$ ,  $(u_i, v_{j+1})$ ,  $(u_{i+1}, v_j)$  a  $(u_{i+1}, v_{j+1})$  tvoria obdĺžnikový úsek nad ktorým sa nachádza splajnový segment. Každý segment  $S_{i,j}$  je bikubická funkcia z  $[u_i, u_{i+1}] \times$



Obr. 1.3: Ukážka uzlov pre štvorsegmentový splajn.

$[v_j, v_{j+1}]$  do  $\mathbb{R}$ . Výsledná funkcia  $S$  teda vznikne zjednotením segmentov  $S_{i,j}$ . Na vypočítanie každého segmentu potrebujeme štyri uzly, a pre každý uzol príslušne hodnoty  $z_{i,j}$ ,  $d_{i,j}^x$ ,  $d_{i,j}^y$  a  $d_{i,j}^{xy}$ . Hodnoty  $z_{i,j}$  máme vopred dané. Derivácie  $d_{i,j}^x$ ,  $d_{i,j}^y$  a  $d_{i,j}^{xy}$  buď môžu byť dané všetky, alebo iba niektoré a z nich vieme s malou odchýlkou odhadnúť hodnoty ostatných. Jeden z týchto spôsobov spočíva v De Boorovej interpolácii ktorú si v ďalších dvoch častiach objasníme. Potom predstavíme modifikovaný spôsob interpolácie špeciálne pre splajny s rovnomerne rastúcimi uzlami.

## 1.1 Trojdiagonálna LU dekompozícia

Základ De Boorovej interpolácie tkvie v opakovanom počítaní systémov trojdiagonálnych lineárnych rovníc.

**Definícia 1.6** Nech  $n \geq 3$  je z  $\mathbb{N} \cup \{0\}$ . Sústavu rovníc tvaru

$$\begin{pmatrix} b_0 & c_0 & 0 & \cdots & 0 & 0 \\ a_0 & b_1 & c_1 & \cdots & 0 & 0 \\ 0 & a_1 & b & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & b & c_{n-2} \\ 0 & 0 & 0 & \cdots & a_{n-2} & b_{n-1} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_{n-1} \\ r_n \end{pmatrix} \quad (1.3)$$

nazveme *trojdiagonálna sústava lineárnych rovníc*.

Jeden z asymptoticky efektívnych spôsobov riešenia týchto rovníc spočíva v  $LU$  dekompozícii  $A\mathbf{x} = L \underbrace{U\mathbf{x}}_{\mathbf{y}} = \mathbf{r}$ :

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ l_1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & l_2 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & l_{n-1} & 1 \end{pmatrix} \cdot \begin{pmatrix} u_0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & u_1 & 1 & \cdots & 0 & 0 \\ 0 & 0 & u_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & v_{n-2} & 0 \\ 0 & 0 & 0 & \cdots & 0 & v_{n-1} \end{pmatrix}, \quad (1.4)$$

Pre  $k$  z  $\{1, \dots, n\}$  sú hodnoty  $v_k$  a  $\lambda_k$  určené takto:

$$v_i = b, \left\{ \lambda_i = \frac{1}{v_{i-1}}, v = b - \lambda_i \right\}, i \in \{2, \dots, n\}. \quad (1.5)$$

Pre priamy a spätný chod máme

$$\text{Priamy: } L\mathbf{y} = \mathbf{r}, y_1 = r_1, \{y_i = r_i - \lambda_i\}, i \in \{2, \dots, n\}, \quad (1.6)$$

$$\text{Spätný: } U\mathbf{d} = \mathbf{y}, d_i = \frac{y_i}{u_i}, \left\{ d_i = \frac{1}{u_i}(y_i - d_{i+1}) \right\}, i \in \{n-1, \dots, 1\}. \quad (1.7)$$

$LU$  dekompozíciou sa rieši ako de Boorova sústava (1.12), tak aj naša redukovaná (1.23). Nižšie si ukážeme optimalizovaný pseudokód tohto algoritmu prevzatý z [8]. Procedúra algoritmu je priestorovo optimalizovaná kde  $LU$  rozklad robíme priamo na vstupných vektoroch predstavujúce diagonály a výsledok ukladáme do vektora obsahujúci pravú stranu. V prípade počítania derivácií je sústava rovníc v špeciálnom tvare podľa 1.12 pre plný alebo 1.23 pre redukovaný algoritmus. Algoritmus môžeme teda ešte ďalej priestorovo a navyše aj výpočtovo optimalizovať, pretože oboch prípadoch všetky čísla na hornej aj spodnej diagonále majú hodnotu 1. Na hlavnej diagonále sú to hodnoty 4 pri plnom algoritme a v prípade redukovaného postupu sú to hodnoty  $-14$ . Posledný prvok diagonály tiež môže obsahovať hodnotu  $-15$  ak počet rovníc je párny. Teda pri implementácii si nemusíme pamätať vektory, stačí nám zapamätať si iba štyri hodnoty. Upravený algoritmus, ktorý rieši  $LU$  dekompozíciu špeciálne pre plný a redukovaný algoritmus je nižšie.

---

**Algoritmus 1** Špeciálna LU dekompozícia

---

```
1: procedúra VYRIEŠLU
2:   vstup:
3:
4:    $b$  hodnota prvkov hlavnej diagonály
5:    $b_l$  hodnota posledného prvku hlavnej diagonály
6:    $r[0..n-1]$  pravá strana
7:   výstup:  $r[0..n-1]$  pravá strana obsahujúca výsledok
8:
9:    $p \leftarrow [0..n-2]$   $\triangleright$  Pomocný vektor nahradzujúci hornú diagonálu
10:   $m \leftarrow 1/b$ 
11:   $p[0] \leftarrow q$ 
12:   $r[0] \leftarrow q \cdot r[0]$ 
13:  pre  $i$  od 1 až  $n-2$ 
14:     $q \leftarrow 1/b - p[i-1]$ 
15:     $p[i] \leftarrow m$ 
16:     $r[i] \leftarrow m \cdot (r[i] - r[i-1])$ 
17:   $m \leftarrow 1/b_l - p[n-1]$ 
18:   $p[i] \leftarrow m$ 
19:   $r[i] \leftarrow m \cdot (r[n-1] - r[n-2])$ 
20:  pre  $i$  od  $n-2$  až 0
21:     $r[i] \leftarrow r[i] - p[i] \cdot r[i+1]$ 
```

---

**Lemma 1.7** *Nech  $n$  je počet rovníc trojdiagonálnej lineárnej sústavy, všetky prvky na spodnej a hornej diagonále majú hodnotu 1 a pre hodnoty  $b_0, b_1, \dots, b_{n-1}$  na hlavnej diagonále platí  $b_0 = b_1 = \dots = b_{n-2}$ . Algoritmus 1 nájde riešenie v lineárnom čase, pričom vyžaduje  $2n$  pamäte.*

Tento modifikovaný algoritmus má menej než polovičné pamäťové nároky než v prípade všeobecných trojdiagonálnych sústava čo nám v praxi to umožňuje riešiť väčšie úlohy. Navyše pri reálnej implementácii nemusíme vektory  $r$  a  $p$  inicializovať pri každom volaní procedúry *VyriešLU* ale namiesto toho môžeme použiť prednačítané vyrovnávacie vektory, tzv. *buffer*. To nám ušetrí mnoho strojových cyklov, ktoré procesor strávi neustálym alokovaním a mazaním potrebnej operačnej pamäte.

## 1.2 De Boorov výpočet derivácií

Nech sú dané uzly  $(u_0, \dots, u_{I-1})$  a  $(v_0, \dots, v_{J-1})$ , kde  $I, J$  sú z  $\mathbb{N} \cup \{0\}$ , pričom chceme interpolovať funkčné hodnoty  $(z_{0,0}, \dots, z_{0,J-1}, \dots, z_{I-1,0}, \dots, z_{I-1,J-1})$ . Výsledný splajn bude teda tvorený  $(I-1) \cdot (J-1)$  segmentami. Ako bolo spomenuté, každý segment splajnu potrebuje štyri uzly a hodnoty  $z$ ,  $d^x$ ,  $d^y$  a  $d^{xy}$ , ktoré potrebujeme získať z funkcie  $f$ . Získanie derivácií môže byť niekedy nákladné. Príkladom môže byť v prípade, keď funkčné hodnoty  $z_{i,j}$  sú získané vyhodnotením nejakej funkcie  $f$ . To môže byť v praxi na počítači značne nákladné najmä v prípade, ak pracujeme so symbolicky zapísanou funkciou vo forme textového reťazca, ktorú je potrebné dynamicky interpretovať počas behu programu.

Algoritmus nájdený Carlom de Boorom [5] umožňuje s malou odchýlkou vypočítať hodnoty derivácií v uzloch na základe nasledujúcich vstupných hodnôt, ktoré máme dané, napríklad vyhodnotením funkcie  $f$ :

- $z_{i,j} = f(u_i, v_j)$  pre  $i \in \{0, \dots, I-1\}$ ,  $j \in \{0, \dots, J-1\}$ .
- $d_{i,j}^x = \frac{\partial f}{\partial x}(u_i, v_j)$  pre  $i \in \{0, I-1\}$ ,  $j \in \{0, \dots, J-1\}$ .
- $d_{i,j}^y = \frac{\partial f}{\partial y}(u_i, v_j)$  pre  $i \in \{0, \dots, I-1\}$ ,  $j \in \{0, J-1\}$ .
- $d_{i,j}^{xy} = \frac{\partial^2 f}{\partial x \partial y}(u_i, v_j)$  pre  $i \in \{0, I-1\}$ ,  $j \in \{0, J-1\}$ .

**Príklad 1.8** Na príklade uzlov z obrázka 1.3 potrebujeme poznať hodnoty nasledovne:

- $z_{0,2}, z_{1,2}, z_{2,2},$   
 $z_{0,1}, z_{1,1}, z_{2,1},$   
 $z_{0,0}, z_{1,0}, z_{2,0},$
- $d_{0,2}^x, \quad , d_{2,2}^x,$   
 $d_{0,1}^x, \quad , d_{2,1}^x,$   
 $d_{0,0}^x, \quad , d_{2,0}^x,$
- $d_{0,2}^y, d_{1,2}^y, d_{2,2}^y,$   
 $, \quad , \quad ,$   
 $d_{0,0}^y, d_{1,0}^y, d_{2,0}^y,$

- $d_{0,2}^{xy}, \dots, d_{2,2}^{xy},$   
 $d_{0,0}^{xy}, \dots, d_{2,0}^{xy},$

Zvyšné derivácie  $d^x$ ,  $d^y$  a  $d^{xy}$  vieme jednoznačne vypočítať pomocou  $2(I) + J + 5$  lineárnych sústav s celkovo  $3IJ + I + J + 2$  rovnicami: Nižšie uvádzame modelové rovnice, pomocou ktorých sú zostrojené tieto sústavy lineárnych rovníc.

Pre  $j \in \{0, \dots, J-1\}$ , teda pre každý stĺpec  $j$  vypočítame parciálne derivácie  $d^x$

$$d_{i+1,j}^x + 4d_{i,j}^x + d_{i-1,j}^x = \frac{3}{h_x}(z_{i+1,j} - z_{i-1,j}), \quad (1.8)$$

$$i \in \{1, \dots, I-2\}$$

Pre  $j \in \{0, J-1\}$ , teda pre prvý a posledný stĺpec vypočítame parciálne derivácie  $d^{x,y}$

$$d_{i+1,j}^{xy} + 4d_{i,j}^{xy} + d_{i-1,j}^{xy} = \frac{3}{h_x}(d_{i+1,j}^y - d_{i-1,j}^y), \quad (1.9)$$

$$i \in \{1, \dots, I-2\}$$

Pre  $i \in \{0, \dots, I-1\}$ , teda pre každý riadok  $i$  vypočítame parciálne derivácie  $d^y$

$$d_{i,j+1}^y + 4d_{i,j}^y + d_{i,j-1}^y = \frac{3}{h_y}(z_{i,j+1} - z_{i,j-1}), \quad (1.10)$$

$$j \in \{1, \dots, J-2\}$$

Pre  $i \in \{0, \dots, I-1\}$ , teda pre každý riadok  $j$  dopočítame parciálne derivácie  $d^{x,y}$

$$d_{i,j+1}^{xy} + 4d_{i,j}^{xy} + d_{i,j-1}^{xy} = \frac{3}{h_y}(d_{i,j+1}^x - d_{i,j-1}^x), \quad (1.11)$$

$$j \in \{1, \dots, J-2\}$$

Každá z týchto sústav má takýto maticový tvar:

$$\begin{pmatrix} 4 & 1 & 0 & \cdots & 0 & 0 \\ 1 & 4 & 1 & \cdots & 0 & 0 \\ 0 & 1 & 4 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 4 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 4 \end{pmatrix} \cdot \begin{pmatrix} D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_{N-3} \\ D_{N-2} \end{pmatrix} = \begin{pmatrix} \frac{3}{\Delta}(Y_2 - Y_0) - D_0 \\ \frac{3}{\Delta}(Y_3 - Y_1) \\ \frac{3}{\Delta}(Y_4 - Y_2) \\ \vdots \\ \frac{3}{\Delta}(Y_{N-2} - Y_{N-4}) \\ \frac{3}{\Delta}(Y_{N-1} - Y_{N-3}) - D_{N-1} \end{pmatrix}, \quad (1.12)$$



kde podľa toho o ktorú z modelových rovníc sa jedná, hodnoty  $N$ ,  $D$  a  $Y$  zadávame následovne. Nech  $k$  z  $1, \dots, K-1$ :

- $N = I$ ,  $h = h_x$ ,  $D_k = d_{k,j}^x$  a  $Y_k = z_{k,j}$ , pre rovnicu 1.8.
- $N = I$ ,  $h = h_x$ ,  $D_k = d_{k,j}^{xy}$  a  $Y_k = d_{k,j}^y$ , pre rovnicu 1.9.
- $N = J$ ,  $h = h_y$ ,  $D_k = d_{i,k}^y$  a  $Y_k = z_{i,k}$ , pre rovnicu 1.10.
- $N = J$ ,  $h = h_y$ ,  $D_k = d_{i,k}^{xy}$  a  $Y_k = d_{i,k}^x$ , pre rovnicu 1.10.

Po vypočítaní všetkých derivácií môžeme funkčné hodnoty jednoznačne interpolovať splajnom.

### 1.3 Počítanie derivácií redukovanou sústavou

V rámci svojej bakalárskej práce som popisoval nový postup pre kubické splajnové krivky triedy  $C^2$ , teda splajny, kde interpolovaná funkcia  $f$  je typu  $\mathbb{R} \rightarrow \mathbb{R}$ . Cieľom tejto práce je postup zovšeobecniť pre bikubické splajny, teda pre splajny, kde interpolovaná funkcia  $f$  je typu  $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ . Podstatou je výsledok výskumu [2] doc. Töröka a RNDr. Szaba o vzťahu medzi bikvartickými polynómami a bikubickými splajnami a odvodenie sústav na základe tohto výsledku [3].

**Označenie 1.9** Teraz popíšeme tento nový algoritmus, ktorý pracovne označíme ako *redukovaný algoritmus*. Ďalej budeme pôvodný de Boorov postup, označovať pojmom *plný algoritmus*.

**Označenie 1.10** Postupnosť  $(a_0, a_1, \dots)$  nazveme *rovnomerne rastúcou* ak pre ľubovoľné  $i$  z  $\{0, 1, \dots\}$  platí  $a_{i+1} > a_i$  a pre ľubovoľné  $i, j$  z  $\{0, 1, \dots\}$  platí  $a_{j+1} - a_j = a_{i+1} - a_i$ .

**Poznámka 1.11** De Boorova interpolácia vo všeobecnosti nepredpokladá len rovnomerne rastúce postupnosti uzlov  $(u_0, \dots, u_I)$  a  $(v_0, \dots, v_J)$ . Náš postup v ďalšej časti článku ale funguje len s takýmito postupnosťami. Preto v tejto časti budeme naďalej uvažovať iba rovnomerne rastúce postupnosti uzlov.

Vstupné hodnoty sú identické ako pri pôvodnom algoritme. Máme dané  $(u_0, \dots, u_{I-1})$  a  $(v_0, \dots, v_{J-1})$ , kde  $I, J$  sú z  $\mathbb{N} \cup \{0\}$ , pričom chceme interpolovať funkciu  $f$  na  $[u_0, u_{I-1}] \times [v_0, v_{J-1}]$ . Pre pripomenutie potrebujeme z funkcie  $f$  priamo získať tieto hodnoty.

- $z_{i,j} = f(u_i, v_j)$ , pre  $i \in \{0, \dots, I-1\}$ ,  $j \in \{0, \dots, J-1\}$ .

- $d_{i,j}^x$  pre  $i \in \{0, I-1\}$ ,  $j \in \{0, \dots, J-1\}$ .
- $d_{i,j}^y$  pre  $i \in \{0, \dots, I-1\}$ ,  $j \in \{0, J-1\}$ .
- $d_{i,j}^{xy}$  pre  $i \in \{0, I-1\}$ ,  $j \in \{0, J-1\}$ .

Zvyšné derivácie  $d^x$ ,  $d^y$  a  $d^{xy}$  vieme jednoznačne vypočítať pomocou  $2(I) + J + 5$  lineárnych sústav s celkovo  $3IJ + I + J + 2$  rovnicami: Nižšie uvádzame modelové rovnice, pomocou ktorých sú zostrojené tieto sústavy lineárnych rovníc. Označme  $I_l$  a  $J_l$  indexy po ktoré budeme iterovať. Platí:

$$I_l = \begin{cases} I-2 & \text{ak } I \text{ je nepárne,} \\ I-3 & \text{ak } I \text{ je párne,} \end{cases}$$

$$J_l = \begin{cases} J-2 & \text{ak } J \text{ je nepárne,} \\ J-3 & \text{ak } J \text{ je párne,} \end{cases}$$

Pre  $j \in \{0, \dots, J-1\}$ , teda pre každý stĺpec  $j$  vypočítame parciálne derivácie  $d^x$

$$d_{i+2,j}^x - 14d_{i,j}^x + d_{i-2,j}^x = \frac{3}{h_x}(z_{i+2,j} - z_{i-2,j}) - \frac{12}{h_x}(z_{i+1,j} - z_{i-1,j}), \quad (1.13)$$

$$i \in \{2, 4, \dots, I_l\}$$

Rovnica je podobná ako plnom algoritme 1.8. Všimnime si, že sústavu rovníc teraz budujeme len pre párne indexy  $i$ , teda vyriešením tejto sústavy získame iba polovicu žiadaných hodnôt  $d^x$ . Pre  $i \in \{1, 3, \dots, I_l\}$  a  $j \in \{0, \dots, J-1\}$  zvyšné derivácie  $d^x$  vypočítame ako

$$d_{i,j}^x = \frac{3}{4h_x}(z_{i+1,j} - z_{i-1,j}) - \frac{1}{4}(d_{i+1,j}^x - d_{i-1,j}^x) \quad (1.14)$$

Pre  $i \in \{0, \dots, I-1\}$ , teda pre každý riadok  $i$  analogicky vypočítame parciálne derivácie  $d^y$

$$d_{i,j+2}^y - 14d_{i,j}^y + d_{i,j-2}^y = \frac{3}{h_y}(z_{i,j+2} - z_{i,j-2}) - \frac{12}{h_y}(z_{i,j+1} - z_{i,j-1}), \quad (1.15)$$

$$i \in \{2, 4, \dots, I_l\}$$

Následne analogicky pre  $i \in \{1, 2, \dots, I-1\}$  a  $j \in \{1, 3, \dots, J_l\}$  zvyšné derivácie  $d^y$  vypočítame ako

$$d_{i,j}^y = \frac{3}{4h_y}(z_{i,j+1} - z_{i,j-1}) - \frac{1}{4}(d_{i,j+1}^y - d_{i,j-1}^y) \quad (1.16)$$

Pre  $j \in \{0, J-1\}$ , teda pre prvý a posledný stĺpec vypočítame parciálne derivácie  $d^{x,y}$  rovnako ako pri plnom algoritme.

$$d_{i+1,j}^{xy} + 4d_{i,j}^{xy} + d_{i-1,j}^{xy} = \frac{3}{h_x}(d_{i+1,j}^y - d_{i-1,j}^y), \quad (1.17)$$

$$i \in \{1, \dots, I-2\}$$

Pre  $i \in \{0, I-1\}$ , teda pre prvý a posledný riadok analogicky vypočítame parciálne derivácie  $d^{x,y}$

$$d_{i,j+1}^{xy} + 4d_{i,j}^{xy} + d_{i,j-1}^{xy} = \frac{3}{h_y}(d_{i,j+1}^x - d_{i,j-1}^x), \quad (1.18)$$

$$j \in \{1, \dots, J-2\}$$

Pre  $i \in \{2, 4, \dots, I_l\}$ , teda pre každý stĺpec  $i$  dopočítame parciálne derivácie  $d^{x,y}$

$$\begin{aligned} d_{i,j+2}^{xy} + 4d_{i,j}^{xy} + d_{i,j-2}^{xy} = & \\ & \frac{1}{7}(d_{i-2,j+2}^{xy} - d_{i-2,j-2}^{xy}) - 2d_{i-2,j}^{xy} \\ & + \frac{3}{7h_x}(d_{i-2,j+2}^y - d_{i-2,j-2}^y) + \frac{3}{7h_y}(-d_{i-2,j+2}^x - d_{i-2,j-2}^x) \\ & + \frac{9}{7h_x}(d_{i,j+2}^y - d_{i,j-2}^y) + \frac{9}{7h_x h_y}(-z_{i-2,j+2} + z_{i-2,j-2}) \\ & + \frac{12}{7h_x}(-d_{i-1,j+2}^y - d_{i-1,j-2}^y) + \frac{12}{7h_y}(d_{i-2,j+1}^x - d_{i-2,j-1}^x) \\ & + \frac{3}{7h_y}(d_{i,j+2}^x - d_{i,j-2}^x) + \frac{27}{7h_x h_y}(-z_{i,j+2} + z_{i,j-2}) \\ & + \frac{36}{7h_x h_y}(z_{i-1,j+2} - z_{i-1,j-2} + z_{i-2,j+1} - z_{i-2,j-1}) \\ & - \frac{6}{h_x}d_{i-2,j}^y + \frac{144}{7h_x h_y}(-z_{i-1,j+1} + z_{i-1,j-1}) + \frac{24}{h_x}d_{i-1,j}^y, \end{aligned} \quad (1.19)$$

$$j \in \{4, 6, \dots, J_l-2\}$$

Následne vypočítame zvyšné derivácie  $d^{x,y}$ . Najprv pre  $i \in \{1, 3, \dots, I_l\}$  a

$j \in \{1, 3, \dots, J_l\}$  platí

$$\begin{aligned}
d_{i,j}^{xy} = & \frac{1}{16}(d_{i+1,j+1}^{xy} + d_{i+1,j-1}^{xy} + d_{i-1,j+1}^{xy} + d_{i-1,j-1}^{xy}) \\
& - \frac{3}{16h_y}(d_{i+1,j+1}^x - d_{i+1,j-1}^x + d_{i-1,j+1}^x - d_{i-1,j-1}^x) \\
& - \frac{3}{16h_x}(d_{i+1,j+1}^y + d_{i+1,j-1}^y - d_{i-1,j+1}^y - d_{i-1,j-1}^y) \\
& + \frac{9}{16h_x h_y}(z_{i+1,j+1} - z_{i+1,j-1} - z_{i-1,j+1} + z_{i-1,j-1}).
\end{aligned} \tag{1.20}$$

Nakoniec pre  $i \in \{1, 3, \dots, I_l + 1\}$  a  $j \in \{2, 4, \dots, J_l\}$

$$d_{i,j}^{xy} = \frac{3}{4h_y}(d_{i,j+1}^x - z d_{i,j-1}^x) - \frac{1}{4}(d_{i,j+1}^{xy} - d_{i,j-1}^{xy}) \tag{1.21}$$

a pre  $i \in \{2, 4, \dots, I_l\}$  a  $j \in \{1, 3, \dots, J_l + 1\}$

$$d_{i,j}^{xy} = \frac{3}{4h_y}(d_{i,j+1}^x - z d_{i,j-1}^x) - \frac{1}{4}(d_{i,j+1}^{xy} - d_{i,j-1}^{xy}) \tag{1.22}$$

**Označenie 1.12** Zavedme dve označenia.

- Rovnice 1.14, 1.16 budeme súhrne označovať pojmom *resty*.
- Rovnice 1.20, 1.21, 1.22 budeme súhrne označovať pojmom *zmiešané resty*.

Modelové sústavy rovníc 1.13 a 1.15 majú takýto maticový tvar:

$$\begin{pmatrix} -14 & 1 & 0 & \cdots & 0 & 0 \\ 1 & -14 & 1 & \cdots & 0 & 0 \\ 0 & 1 & -14 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -14 & 1 \\ 0 & 0 & 0 & \cdots & 1 & \mu \end{pmatrix} \cdot \begin{pmatrix} D_2 \\ D_4 \\ D_6 \\ \vdots \\ D_{v-2} \\ D_v \end{pmatrix} = \begin{pmatrix} \frac{3}{h}(Y_4 - Y_0) - \frac{12}{h}(Y_3 - Y_1) - D_0 \\ \frac{3}{h}(Y_6 - Y_2) - \frac{12}{h}(Y_5 - Y_3) \\ \frac{3}{h}(Y_8 - Y_4) - \frac{12}{h}(Y_7 - Y_5) \\ \vdots \\ \frac{3}{h}(Y_\nu - Y_{\nu-4}) - \frac{12}{h}(Y_{\nu-3} - Y_{\nu-5}) \\ \frac{3}{h}(Y_{\nu+\tau} - Y_{\nu-2}) - \frac{12}{h}(Y_{\nu-1} - Y_{\nu-3} - \theta D_{K+1}) \end{pmatrix}, \tag{1.23}$$

kde

$$\begin{aligned} \mu = -15, \tau = 0, \theta = -4, \text{ a } \nu = N, & \quad \text{ak } K \text{ je párne,} \\ \mu = -14, \tau = 2, \theta = 1, \text{ a } \nu = N - 1, & \quad \text{ak } N \text{ je nepárne,} \end{aligned} \quad (1.24)$$

a podľa toho o ktorú z modelových rovníc sa jedná, hodnoty  $N$ ,  $D$  a  $Y$  zadávame následovne. Nech  $k$  z  $1, \dots, K - 1$ :

- $N = I$ ,  $h = h_x$ ,  $D_k = d_{k,j}^x$  a  $Y_k = z_{k,j}$ , pre rovnicu 1.13..
- $N = J$ ,  $h = h_y$ ,  $D_k = d_{i,k}^y$  a  $Y_k = z_{i,k}$ , pre rovnicu 1.15.

Analogicky vieme zostrojiť maticový tvar aj pre modelovú sústavu pre derivácie  $d^{xy}$  podľa 1.19.

**Označenie 1.13** Špeciálne zavedme označenia pre tieto hodnoty:

- Parciálne derivácie  $d^x$  a  $d^y$  počítané rovnicami 1.14 a 1.16 budeme nazývať *zbytkové derivácie*.
- Zmiešané parciálne derivácie  $d^{xy}$  počítané rovnicami 1.20, 1.21 a 1.22 budeme nazývať *zmiešané zbytkové derivácie*.

## 1.4 Konštrukcia hermitovho splajnu

Teraz si ukážeme ako podľa zadaných uzlov a vypočítaných derivácií zostrojiť segmenty splajnu ako funkcie dvoch premenných v priestore. Predpokladajme  $I \geq 3$  a  $J \geq 3$  z  $\mathbb{N} \cup \{0\}$ , rovnomerné uzly  $u_0, \dots, u_{I-1}$  a  $v_0, \dots, v_{J-1}$ . Úlohou je zostrojiť splajn  $S$  triedy  $C^2$  interpolujúci funkciu  $f$ . Položme

- funkčné hodnoty splajnu  $z_{0,0}, \dots, z_{I-1,0}, \dots, z_{0,J-1}, \dots, z_{I-1,J-1}$ ,
- $x$ -ové derivácie splajnu  $d_{0,0}^x, \dots, d_{I-1,0}^x, \dots, d_{0,J-1}^x, \dots, d_{I-1,J-1}^x$ ,
- $y$ -ové derivácie splajnu  $d_{0,0}^y, \dots, d_{I-1,0}^y, \dots, d_{0,J-1}^y, \dots, d_{I-1,J-1}^y$ ,
- $x$ -ové derivácie splajnu  $d_{0,0}^{xy}, \dots, d_{I-1,0}^{xy}, \dots, d_{0,J-1}^{xy}, \dots, d_{I-1,J-1}^{xy}$ ,

a inicializujme ich podľa 1.2.

Ďalším krokom je vypočítanie zvyšných hodnôt derivácií buď plným alebo redukovaným algoritmom. Výstupom oboch sú rovnaké hodnoty derivácií, ktoré nám zaručia spojitosť a hladkosť medzi jednotlivými segmentami budovaného splajnu, čím podľa definície bude patriť práve do triedy  $C^2$ .

Teraz už máme všetky potrebné hodnoty aby sme mohli definovať segmenty splajnu  $S$  podľa definície 1.3. Pre každé  $i$  z  $\{0, \dots, I-2\}$  a  $j$  z  $\{0, \dots, J-2\}$  položíme segment  $S_{i,j} : [u_i, v_j] \times [u_{i+1}, v_{j+1}] \rightarrow \mathbb{R}$  vzťahom:

$$S_{i,j}(x, y) = \lambda^T(x, u_i, u_{i+1}) \cdot \Phi(u_i, u_{i+1}, v_j, v_{j+1}) \cdot \lambda(y, v_j, v_{j+1}), \quad (1.25)$$

kde

$$\lambda(t, t_0, t_1) = \left( \frac{(t-t_1)^2(1-2\frac{t-t_0}{t_0-t_1})}{(t_0-t_1)^2}, \frac{(t-t_0)^2(1-2\frac{t-t_1}{t_1-t_0})}{(t_1-t_0)^2}, \frac{(t-t_1)^2(t-t_0)}{(t_0-t_1)^2}, \frac{(t-t_0)^2(t-t_1)}{(t_1-t_0)^2} \right)^T, \quad (1.26)$$

$$\Phi(t_0, t_1, s_0, s_1) = \left( \frac{(t-t_1)^2(1-2\frac{t-t_0}{t_0-t_1})}{(t_0-t_1)^2}, \frac{(t-t_0)^2(1-2\frac{t-t_1}{t_1-t_0})}{(t_1-t_0)^2}, \frac{(t-t_1)^2(t-t_0)}{(t_0-t_1)^2}, \frac{(t-t_0)^2(t-t_1)}{(t_1-t_0)^2} \right)^T, \quad (1.27)$$

**Lemma 1.14** *Nech sú splnené podmienky na začiatku tejto časti (1.4. Splajn  $S$  na rovnomerne rastúcich uzloch s vyššie definovanými segmentami a deriváciami v uzloch vypočítaných podľa plného alebo podľa redukovaného algoritmu je triedy  $C^2$  a interpoluje funkčné hodnoty  $z_{0,0}, \dots, z_{I-1,0}, \dots, z_{0,J-1}, \dots, z_{I-1,J-1}$ .*

## Kapitola 2

### Zrýchlenie

V predchádzajúcej kapitole sme popísali dva postupy výpočtu neznámych derivácií pre splajnové povrchy. Časová zložitosť oboch algoritmov je rovnaká a síce  $O(I \cdot J)$ . Pri určení asymptotickej časovej zložitosti zvyčajne zanedbávame rýchlosti jednoduchých krokov algoritmu ako sú napríklad aritmetické operácie, porovnávanie veľkosti čísel, kopírovanie a podobne. Keď porovnáваме rýchlosť asymptoticky rovnako rýchlych postupoch musíme brať do úvahy vplyvy jednotlivých týchto triviálnych krokov. Pochopiteľne tieto operácie nemusia byť rovnako rýchle a môže nastať situácia kedy algoritmus s väčším počtom krokov bude rýchlejší ako iný s menším počtom krokov ak tieto kroky bude procesor počítača schopný vykonávať rýchlejšie.

To je práve aj náš prípad. Redukovaný algoritmus De Boorovej interpolácie obsahuje väčší počet aritmetických operácií ako sú sčítanie, násobenie, delenie. V časti 2.2 si ukážeme, že redukovaný algoritmus je skutočne rýchlejší. Tu iba poznamenajme, že zrýchlenie nastáva vďaka zredukovanému počtu delení. V tejto kapitole si spočítame počty týchto operácií, ale predtým popíšeme technické parametre procesorov v dôsledku ktorých dosahujeme zrýchlenie. Následne pri počítaní rýchlosti algoritmov budeme schopní k jednotlivým krokom algoritmu a aritmetickým operáciám v jednotlivých krokoch priradiť *váhy*. Potom už budeme schopní presne určiť skutočné zrýchlenie a porovnať ho so zrýchlením v konkrétnej implementácii.

Na moderných procesoroch v prípade matematických operácií s plávajúcou desatinnou čiarkou platí, že sčítanie a násobenie sú podobne rýchle, pričom delenie je niekoľkonásobne pomalšie. Pamäťové operácie sú pritom v závislosti od konkrétneho typu operačnej pamäte približne dvadsaťnásobne pomalšie ako sčítanie, resp. násobenie.<sup>1</sup>

---

<sup>1</sup>V praxi sa aj výpočty rádovo tisícov uzlov často zmestia do vyrovnávacej pamäte procesora (*cache*). V tom prípade sú pamäťové operácie len  $1 \times - 10 \times$  pomalšie ako sčítanie.

## 2.1 Procesorová architektúra

V posledných rokoch sa v počítačovej vede stále viac spomína pojem paralelizmu. Dnešné procesory architektúry x86 využívajú až štyri úrovne paralelizácie výpočtov. Na najvyššej úrovni hovoríme o návrhu kedy je procesor zložený z niekoľkých autonómne pracujúcich „podprocesorov“ nazývaných *jadrá*. Tie zdieľajú systémové zbernice a pamäť pričom majú väčšinou vlastnú L1 alebo L2 cache. Každé jadro môže spracovávať na sebe nezávislé procesy prípadne jeden proces môže byť rozdelený do takzvaných vlákien, kde každé môže byť spracovávané iným jadrom. V tomto prípade hovoríme o takzvanom *vláknovom paralelizme* s ktorým sa stretávame najmä pri programovaní vo vyšších jazykoch ako sú C, Java, Haskell a iné.

Ďalšie úrovne paralelizácie majú spoločné označenie *inštrukčný paralelizmus*. Pod týmto pojmom rozumieme *superskalárnosť*, *pipelining* a *vektorizácia*. Pri tejto úrovni paralelizmu má programátor len obmedzené možnosti jeho ovplyvnenia, všetku „ťažkú“ prácu obstará prekladač a pri behu aplikácie zasa inštrukčný plánovač v samotnom procesorovom jadre.

Hlavný vplyv na výkon plného a najmä redukovaného algoritmu má superskalárnosť. Najprv si vysvetlíme čo tento pojem znamená. Jadro je tvorené niekoľkými až desiatkami špecializovanými jednotkami ako sú aritmeticko-logické jednotky (ALU), bitové posuvníky (Shift), numerické koprocessory (FPU) a podobne. Narozdiel od celých jadier pri vláknovom paralelizme, tieto jednotky vo všeobecnosti nedokážu fungovať samostatne a simultánne spracovávať viacero vlákien alebo procesov<sup>2</sup>. Vedia iba v rámci jedného vlákna, za splnenia určitých podmienok, spracovávať niekoľko inštrukcií naraz. Nutnou podmienkou je napríklad vzájomná nezávislosť niekoľkých po sebe idúcich inštrukcií, teda keď výsledok jednej inštrukcie nezávisí na výsledku predchádzajúcej.

Tu ale paralelizácia nekončí. Aj samotné jednotky totiž dokážu simultánne spracovávať viac než jeden dátový vstup. Na tejto úrovni rozlišujeme medzi vektorizáciou, a pipeliningom. Prvá menovaná technika umožňuje jednu inštrukciu aplikovať na celé vektory, resp. polia. Vektorizácia funguje ak vykonávame operácie na vektore tak, že výpočet  $i$ -teho prvku nezávisí na výsledku výpočtu (napr.)  $(i - 1)$ -teho prvku. V algoritme 1 pre LU dekompozíciu môžeme vidieť, že toto neplatí. Vektorizácia nás teda nemusí zaujímať.

Pipelining je založený na myšlienke princípu fungovania výpočtových jednotiek podobne ako výrobná linka vo fabrike, kde nový výrobok na linku vstúpi skôr ako je predchádzajúci dokončený. Teda aj väčšina výpočtových

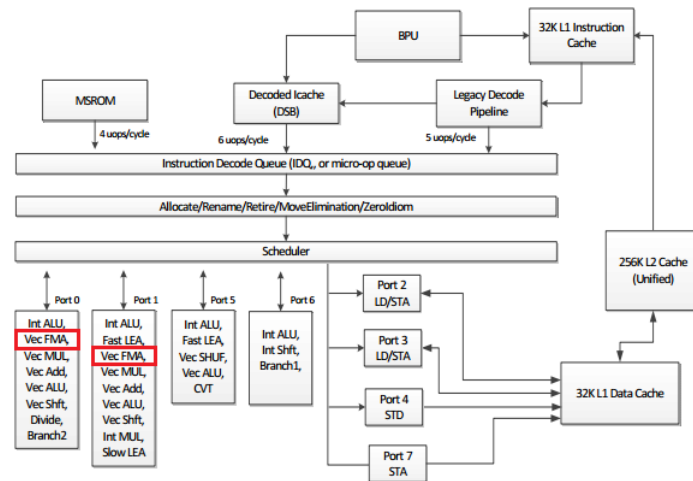
---

<sup>2</sup>Existujú technológie ako napríklad Hyper-threading umožňujúce za určitých podmienok uplatniť vláknový paralelizmus aj v rámci jediného superskalárneho jadra.



jednotiek dokáže s novým výpočtom začať ešte pred dokončením práve prebiehajúcej operácie. Táto technika sa prejavuje najviac ak máme veľký počet operácií rovnakého typu, ktoré sú navyše na sebe nezávislé podobne ako pri vektorizácii. To opäť nie je prípad našich algoritmov takže ho zanedbáme.

**Poznámka 2.1** Aby som neuviedol čitateľa v omyl, svoje závery ohľadom vektorizácie a pipeliningu upresním. Procesor pri vykonávaní algoritmu robí mnoho operácií nevýpočtového charakteru, ktoré sú programátorovi skryté a kde sa môžu (nielen) tieto techniky prejavovať. Ale aj po zanedbaní dostávame korešpondujúce výsledky meraní a vypočítaného zrýchlenia, ktoré sa líšia o najviac 10%. Uvažovaním vplyvu týchto techník pri počítaní zrýchlenia by sa dosiahnutý cieľ a síce vysvetlenie príčin urýchlenia redukovaného algoritmu nezmenil. Teda vzhľadom na množstvo potenciálne investovaného času by zanedbanie týchto techník bolo kontraproduktívne.



Obr. 2.4: Schéma výpočtového jadra mikroarchitektúry Intel Skylake[10].

Spomenuli sme majoritný vplyv superskalárnosti. Na obrázku 2.4 vidíme schému jedného jadra procesorovej mikroarchitektúry Intel Skylake, ktorá je ku dňu písania práce najmodernejšou bežne dostupnou procesorovou generáciou a na ktorej testujeme náš zrýchlenie redukovaného algoritmu. Jadrá (nielen) tejto architektúry sú vybavené dvomi sčítačko-násobičkami čísel s plávajúcou desatinnou čiarkou, ktoré sú na obrázku zvýraznené červenou farbou. Pointa superskalárnosti v tomto prípade spočíva v možnosti výpočet výrazov typu  $a_0 \circ a_1 \circ \dots \circ a_n$ , kde  $\circ \in \{+, \cdot\}$  rozložiť medzi dve jednotky a dosiahnuť dvojnásobného zrýchlenia. Toto má značný vplyv na rýchlosť najmä

redukovaného algoritmu pri počítaní pravých strán trojdiagonálnych rovníc, ktoré sú na prvý pohľad zložitejšie ako v prípade plného algoritmu. Keďže procesor má ale iba jednu deličku, pre výrazy typu  $a_0/a_1/\dots/a_n$  takýto trik fungovať nebude.

Procesorové jadro na obrázku vyššie má svojich 29 jednotiek prístupných cez osem zberníc (porty 0 až 7). To konkrétne v prípade procesorov Skylake znamená, že dokážu naraz využívať maximálne osem svojich výpočtových jednotiek za splnenia určitých podmienok.

Málokedy ale proces alebo vlákno obsahuje vždy osmice vzájomne nezávislých inštrukcií. V situácií kedy jeden proces (vlákno) nemôže využiť všetky jednotky jadra, niektoré procesory umožňujú nevyužívané porty priradiť inému procesu (vláknu). Táto technika sa nazýva *Simultaneous Multi-Threading* (ďalej len SMT). V literatúre sa, v prípade procesoroch Intelu, často môžeme stretnúť s názvom *Hyper-Threading*. Procesory AMD majú implementovaný odlišný variant tejto techniky zvaný *Clustered Multi-Threading*, kde namiesto klasických jadier je procesor vybavený tzv. modulmi, kde každý modul má dve samostatné „oklieštené“ jadrá schopné spracovávať iba celočíselné operácie a jedinú FPU jednotku spoločnú pre oba jadrá. V praxi teda SMT predstavuje akýsi medzistupeň medzi vláknovým a inštrukčným paralelizmom, kedy procesor operačnému systému a bežiacim procesom hlási väčší počet jadier akým je v skutočnosti vybavený, pričom interne na úrovni mikroarchitektúry sa jedná iba o ďalšiu techniku superskalárnosti.

SMT má výrazne pozitívny vplyv na výkon procesora ak procesy (vlákna) zdieľajúce jedno jadro nepožadujú prístup k tým istým jednotkám. V opačnom prípade dochádza k javu kedy jeden proces (vlákno) musí čakať na uvoľnenie jednotky druhým procesom (vláknom) a vtedy môže dôjsť k zníženiu výkonu oproti identickému procesoru s neaktívnym SMT. V praktických meraniach má SMT na implementované algoritmy negatívny vplyv.

### 2.1.1 Rýchlosť aritmetických operácií

Procesory majú mnoho aritmetických jednotiek špecializovaných na určitý typ operácie. Je prirodzené predpokladať, že tieto jednotky budú pracovať navzájom rozličnými rýchlosťami. V prípade výpočtových algoritmov má pochopiteľne okrem výberu vhodných dátových štruktúr vplyv najmä rýchlosť vykonania základných matematických operácií a síce sčítania, odčítania, násobenia a delenia.

Najlepším zdrojom ako zistiť rýchlosť týchto operácií je dokumentácia inštrukčných sád procesorov. Inštrukčná sada x86 sa od svojho prvotného uvedenia v roku 1978 dočkala mnohých rozšírení. Moderné procesory obsahujú mnoho spôsobov ako napríklad vynásobiť dve čísla. Keďže nie je v silách

otestovať všetky možné rozšírenia sady, zvolili sme si jedno konkrétne rozšírenie a síce Streaming SIMD Extensions (skrátene SSE), konkrétne vo verzii 4 (SSE4). Sady z rodiny SSE sú v čase písania práce najpoužívanějšími sadami (najmä verzia SSE2) ktoré sú podporované prakticky všetkými procesormi od roku 2003.

V súčasnosti už existuje modernejšia náhrada tejto sady zvaná Advanced Vector Extensions (skratka AVX), ktorej hlavný prínos spočíva vo vylepšených vektorových operáciach. Vektorové operácie však vyžadujú aby výpočty využívajúce jednotlivé prvky vektora boli navzájom nezávislé. To nie je prípad ani jedného z testovaných algoritmov. Nakonfigurovaním prekladača na generovanie týchto inštrukcií by sme iba znemožnili beh aplikácie na polovici z testovaných procesorov. Praktické zrýchlenie sme testovali na šiestich procesoroch architektúry x86, pričom sme pokryli väčšinu mikroarchitektúr v rozmedzí rokov 2007 až 2015.

V nasledujúcej tabuľke [11] uvidíme rýchlosti štyroch základných matematických operácií v rámci šiestich testovaných mikroarchitektúr. Tabuľka obashaie tieto stĺpce.

- **Architektúra (rok)**

Testovaná mikroarchitektúra a rok jej uvedenia na trh. Architektúry sú zoradené abecedne podľa výrobcu a následne podľa roku vydania.

- **Odozva**

Počet strojových cyklov potrebných na vykonanie inštrukcie.

- **Inverzný prietok**

Počet strojových cyklov ktoré je nutné čakať kým je daná výpočtová jednotka schopná zopakovať inštrukciu. V prípade operácií sčítania, odčítania a násobenia je tento počet menší ako odozva. To znamená, že aritmetické sčítačky a násobičky sú schopné vďaka technike pipelining-u začať nový výpočet ešte pred dokončením aktuálneho výpočtu.

Podľa tabuliek vidno, že operácie sčítania a odčítania sú rovnako rýchle čo sa pochopiteľne dalo očakávať. Tieto dve operácie preto budeme spoločne označovať symbolom  $\pm$ . Od tejto chvíle ak spomenieme operáciu sčítania tak tým budeme súčasne myslieť aj operáciu odčítania. Ako vidieť zďaleka najpomalšie je práve delenie.

Pre zaujímavosť si môžeme ukázať praktické výsledky z testovacej aplikácie. Merali sme operácie na dvoch 512 prvkových poliach, pričom aby sme dostali „rozumne“ dlhé časy výpočty boli opakované 500000 krát. Ďalej

---

<sup>2</sup>Odozva je nadobúda menšie hodnoty ak je deliteľ celý.

Architektúra (rok)	Odozva				Inverzný prietok			
	+	−	·	÷	+	−	·	÷
AMD Piledriver (2012)	5-6	5-6	5-6	9-27	0,5	0,5	0,5	5-10
Intel Penryn (2007)	3	3	5	6-21 <sup>2</sup>	1	1	1	5-20 <sup>2</sup>
Intel Nehalem (2008)	3	3	5	7-22	1	1	1	7-22
Intel Sandy Bridge (2011)	3	3	5	10-22	1	1	1	10-22
Intel Haswell (2013)	3	3	5	10-20	1	1	0,5	8-14
Intel Skylake (2015)	4	4	4	13-14	0,5	0,5	0,5	4

Tabuľka 2.1: Tabuľka aritmetických operácií na rôznych mikroarchitektúrach.

Procesor	±	·	÷	⋈	⋈·
AMD FX-6300	176	173	673		
Intel Core 2 Duo E8200	namerat	u	c. toroka		
Intel Core i5 650	198	224	1758		
Intel Core i3 2350M	233	231	2462		
Intel Core i5 4440	120	119	1087		
Intel Core i7 6700K	99	99	257	3,8	59

Tabuľka 2.2: Rýchlosť aritmetických operácií na konkrétnych CPU.

budeme merať aj pamäťovú operáciu kopírovania, pričom rozlišujeme medzi kopírovaním spojitých a nespojitých dát.

**Označenie 2.2** Označme dve triedy dátových štruktúr symbolmi:

- $\succ$  označuje spojitú dátovú štruktúru ako napríklad vektory.
- $\therefore$  označuje nespojitú dátovú štruktúru ako napríklad spájané zoznamy alebo vektory referencií na konkrétne dáta. Taktiež označuje aj spojitú vektory ak operácia kopírovania z jedného vektora do druhého nie je vykonávaná spojitou (napr. kopírujeme iba prvky na nepárnych inde-  
xoch).

V tabuľke sú namiesto mikroarchitektúr uvedené konkrétne modely procesorov, pričom ich poradie zodpovedá poradiu z minulej tabuľky. Všetky údaje sú v milisekundách.

V praxi vidno, že sčítavanie a násobenie môžeme považovať za podobne rýchle operácie. Delenie býva až desať násobne pomalšie. To je spôsobné

veľmi vysokou hodnotou inverzného prietoku v prípade opakovane vykonávanom delení čo nastáva napríklad ak chceme vydeliť dva vektory. V prípade našich algoritmov ale dochádza len k jednému deleniu na mnoho sčítaní a násobení. Z toho dôvodu môžeme inverzný prietok v prípade delenia zanedbať a v kontexte implementovaných algoritmov delenie považovať podľa jeho odozvy za približne **trojnásobne** pomalšie ako násobenie.

## 2.2 Teoretické zrýchlenie

V tejto časti si spočítame počty operácií procedúr tvoriacich plný a redukovaný algoritmus. Na základe poznatkov z časti o procesorovej architektúre si formálne zdefinujeme pojmy pre počty a ceny aritmetických operácií ktoré budeme v nasledujúcich častiach.

**Označenie 2.3** Množinu všetkých matematických výrazov budeme označme symbolom  $\mathbb{V}$ .

Zdefinujeme si funkciu  $o$  ktorá vráti počet aritmetických operácií sčítania.

**Definícia 2.4** Nech  $\mathcal{V}$  je matematický výraz obsahujúci  $s$  sčítaní,  $n$  násobení a  $d$  delení. Definujeme funkciu  $o : \mathbb{V} \rightarrow \mathbb{N} \times \mathbb{N} \times \mathbb{N}$  vzťahom

$$o(\mathcal{V}) = \langle s, n, d \rangle.$$

Funkciu  $o$  budeme nazývať *počet operácií výrazu  $V$* .

**Označenie 2.5** Podľa časti 2.1 o inštrukčnom paralelizme položíme dve premenné, ktoré nám pomôžu definovať ceny aritmetických operácií.

- Hodnota  $\beta$  značí *faktor superskalárnosti* operácií s plávajúcou desatinnou čiarkou. Inými slovami hodnota  $\beta$  značí počet jednotiek jadra procesora schopných počítat desatinné čísla.
- Hodnota  $\gamma$  značí *pomer odozvy delenia a násobenia* pri operáciách s plávajúcou desatinnou čiarkou. Inými slovami hodnota  $\gamma$  značí koľkokrát je delenie pomalšie oproti sčítaniu v prípade, keď nedochádza k častému opakovanému násobeniu (resp. sčítaniu).

**Poznámka 2.6** Pre moderné procesory architektúry x86 budeme uvažovať  $\beta = 2$  a  $\gamma = 3$ .

**Poznámka 2.7** Faktor superskalárnosti  $\beta$  budeme uvažovať iba pre operácie sčítania a násobenia. Žiadny bežne dostupný procesor totiž nedokáže paralelizovať delenie.

Teraz si definujeme funkciu  $c$  ktorá vráti cenu aritmetických operácií pre nejakú matematickú operáciu berúc do úvahy hodnoty  $\beta$  a  $\gamma$ .

**Definícia 2.8** Nech  $s$  je počet sčítaní,  $n$  je počet násobení a  $d$  je počet delení. Definujeme funkcie cien operácií

- $c : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N} \times \mathbb{N}$  vzťahom

$$c(\langle s, n, d \rangle) = \left\langle \left\lceil \frac{s}{\beta} \right\rceil, \left\lceil \frac{n}{\beta} \right\rceil, \gamma d \right\rangle,$$

ktorú nazveme *cena operácií*.

- Nech  $V$  je matematický výraz a  $o(V) = \langle s, n, d \rangle$ . Potom *cena operácií výrazu*  $V$  je funkcia  $c : \mathbb{V} \rightarrow \mathbb{N} \times \mathbb{N} \times \mathbb{N}$  v tvare

$$c(\mathcal{V}) = c(o(\mathcal{V})).$$

V kontexte počítania operácií a cien budeme z dôvodu zjednodušenia procedúry považovať za množiny matematických výrazov, pričom algoritmy budeme považovať za množiny obsahujúce procedúry a výrazy.

**Označenie 2.9** Stručne doplníme počty aj pre procedúry a algoritmy.

- Nech  $\mathcal{P}$  je procedúra. Funkciu

$$o(\mathcal{P}) = \sum_{v \in \mathcal{P}} o(v)$$

nazveme *počet operácií procedúry*  $\mathcal{P}$ .

- Nech  $\mathcal{A}$  je procedúra. Funkciu

$$c(\mathcal{A}) = \sum_{x \in \mathcal{A}} c(x)$$

nazveme *počet operácií algoritmu*  $\mathcal{A}$ .

**Označenie 2.10** Analogicky označme aj ceny procedúr a algoritmov.

- Nech  $\mathcal{P}$  je procedúra. Funkciu

$$c(\mathcal{P}) = \sum_{v \in \mathcal{P}} c(v)$$

nazveme *cena operácií procedúry*  $\mathcal{P}$ .

- Nech  $\mathcal{A}$  je procedúra. Funkciu

$$c(\mathcal{A}) = \sum_{x \in \mathcal{A}} c(x)$$

nazveme *cena operácií algoritmu*  $\mathcal{A}$ .

Dôvod, prečo je náš redukovaný algoritmus rýchlejší je práve fakt, že pri ňom dochádza k značne menšiemu počtu delení, ktoré je oproti ostatným trom operáciám výrazne pomalšie. V ďalšej sekcii si spočítame jednotlivé operácie a vypočítame teoretické zrýchlenie redukovaného algoritmu. To budeme počítat tak, že si zadefinujeme „procedúry“ ktoré predstavujú jednotlivé časti algoritmu podľa častí 1.1, 1.2 a 1.3.

Najprv položíme procedúry predstavujúce LU dekompozíciu spoločne pre oba postupy.

- Procedúra *InicializujLU* inicializuje hodnoty pravej strany sústavy rovníc  $r_0, \dots, r_{K-1}$  a hodnotu  $b$  z LU dekompozície 1.5 podľa 1.12 v prípade plného algoritmu, respektíve podľa 1.23 v prípade redukovaného algoritmu.
- Procedúra *VyriešLU* vypočíta sústavu rovníc na základe hodnôt pravej strany poskytnutými *InicializujLU*.

Každý algoritmus inicializuje pravé strany inak, takže pre oba si ceny uvedieme osobitne.

### 2.2.1 Cena plného algoritmu

Počty základných aritmetických a pamäťových operácií pre vyššie uvedené procedúry môžeme zhrnúť do tabuľky, pričom uvažujeme počty v našej ukážkovej implementácii<sup>3</sup>. Riadky tabuľky predstavujú jednotlivé procedúry a stĺpce udávajú počet vykonaných operácií, kde  $K$  je počet neznámych. Podotýkam, že operáciu odčítania budeme uvažovať ako sčítanie. Operácia  $\succ$  predstavuje kopírovanie vypočítaných derivácií do výslednej matice uzlov. Pri meraní sa ukázalo, že kopírovanie spojitých dát vieme má cenu 0,04 teda kopírovanie je 25 násobne rýchlejšie ako sčítanie. V tabuľkách budeme pre sčítanie, násobenie a delenie uvádzať ich ceny podľa definície 2.8. V prípade plného algoritmu dochádza len ku kopírovaniu spojitých dát, teda počet kopírovaní v tabuľkách vynásobíme 0,04.

<sup>3</sup>Repozitár so zdrojovými kódmi k aplikáciám možno nájsť na adrese <https://github.com/vildibald/VKDiplom-master>

Procedúra	$\pm$	$\cdot$	$\div$	$\succ$
<i>InicalizujLU</i>	$K$	$K$	0	0
<i>VyriešLU</i>	$3K$	$2K$	$\gamma K$	$1/25 K$

Tabuľka 2.3: Ceny operácií LU dekompozície pre plný algoritmus vzhľadom na počet uzlov  $I \cdot J$ .

Procedúry *VyriešLU* a *InicalizujLU* neobsahujú matematické výrazy s viac ako jedným sčítaním prípadne násobením. Superskalárnosť procesora sa teda v prípade plného algoritmu neprejaví. Uvažujme procedúry predstavujúce implementáciu plného algoritmu podľa časti 1.2:

- Procedúra *VypočítajDx* vypočíta parciálne derivácie  $d^x$  pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.8. Jedno volanie *VyriešLU* vypočíta derivácie  $d^x$  pre jeden stĺpec. Teda procedúra musí počítat LU pre každý stĺpec, ktorých je  $J$ .
- Procedúra *VypočítajDxy* vypočíta parciálne derivácie  $d^{xy}$  pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.9. Jedno volanie *VyriešLU* vypočíta derivácie  $d^{xy}$  pre jeden stĺpec. Procedúra musí počítat LU pre prvý a posledný stĺpec.
- Procedúra *VypočítajDy* vypočíta parciálne derivácie  $d^y$  pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.10. Jedno volanie *VyriešLU* vypočíta derivácie  $d^y$  pre jeden riadok. Teda procedúra musí počítat LU pre každý riadok, ktorých je  $I$ .
- Procedúra *VypočítajDyx* vypočíta parciálne derivácie  $d^{xy}$  pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.11. Jedno volanie *VyriešLU* vypočíta derivácie  $d^{xy}$  pre jeden riadok. Teda procedúra musí počítat LU pre každý riadok, ktorých je  $I$ .
- Procedúra *VypočítajPlný* vypočíta na základe vstupných hodnôt pre de Boorovu interpoláciu 1.2 zavolaním procedúr *VypočítajDx*, *VypočítajDxy*, *VypočítajDy* a *VypočítajDyx*.

Nech  $I$  značí počet uzlov na osi  $x$  a  $J$  značí počet uzlov na osi  $y$ . Všetky počty sú v tvare  $a \cdot IJ + b \cdot I + c \cdot J + d$ . Pre zjednodušenie budeme počty, tam kde je koeficient  $a$  nenulový, uvádzať v tvare  $a \cdot IJ$ . Keďže počty rastú kvadraticky tak pre veľké  $I$  a  $J$  bude odchýlka zanedbateľná. Pre plný algoritmus teda dostaneme tieto počty



Procedúra	$\pm$	$\cdot$	$\div$	$\gamma$
<i>VypočítajDx</i>	$4IJ$	$3IJ$	$\gamma IJ$	$1/_{25}IJ$
<i>VypočítajDxy</i>	$8I$	$6I$	$2\gamma I$	$1/_{25}I$
<i>VypočítajDy</i>	$4IJ$	$3IJ$	$\gamma IJ$	$1/_{25}IJ$
<i>VypočítajDyx</i>	$4IJ$	$3IJ$	$\gamma IJ$	$1/_{25}IJ$

Tabuľka 2.4: Ceny operácií plného algoritmu.

Sčítaním všetkých cien operácií v tabuľke dostaneme nasledujúci výsledok.

**Lemma 2.11** *Nech  $I$  a  $J$  označujú počty uzlov na osiach  $x$  a  $y$ . Potom sumárna cena operácií plného algoritmu je*

- $12IJ$  sčítaní,
- $9IJ$  násobení,
- $3\gamma IJ$  delení,
- $0^3/_{25}IJ$  kopírovaní.

Cena plného algoritmu teda je

$$^{528}/_{25}IJ + 3\gamma IJ.$$

## 2.2.2 Počty pre redukovaný algoritmus

Redukovaný spôsob na počítanie derivácií využíva iný tvar trojdiagonálnej sústavy rovníc. Zmeňme počty operácií procedúr *VyriešLU* a *InicalizujLU*. Ďalej položme dve nové pomocné procedúry *InicalizujZmiešLU*, ktorá inicializuje hodnoty  $r_0, \dots, r_{K-1}$  a  $b$  z LU dekompozície 1.5 pre zmiešané derivácie  $d^{xy}$  a k nej príslušnú *VyriešZmiešLU*.

V prípade redukovaného algoritmu nie je kopírovanie výsledku LU dekompozície spojené. Konkrétne to znamená, že potrebujeme do vektora dĺžky napríklad  $I$  na každú nepárnu pozíciu uložiť hodnotu z vektora  $I/2$ , pretože LU nám nájde iba polovicu hľadaných derivácií. Cena kopírovania bude podľa reálneho merania 0.6. Aby sme takéto kopírovanie vizuálne odlíšili od kopírovania v predchádzajúcej časti, budeme túto operáciu označovať symbolom  $\therefore$ .

V prípade inicializovania pravej strany pre LU dekompozíciu v procedúre *InicalizujLU* dochádza k viacerým sčítaniam a/alebo násobeniam v rámci

Procedúra	$\pm$	$\cdot$	$\div$	$\div$
<i>InicalizujLU</i>	$\frac{3}{\beta}K$	$\frac{2}{\beta}K$	0	0
<i>VyriešLU</i>	$3K$	$2K$	$\gamma K$	$\frac{3}{5}K$

Tabuľka 2.5: Ceny operácií LU dekompozície pre redukovaný algoritmus.

Procedúra	$\pm$	$\cdot$	$\div$	$\div$
<i>InicalizujZmiešLU</i>	$\frac{33}{2}K$	$\frac{17}{2}K$	0	0
<i>VyriešZmiešLU</i>	$3K$	$2K$	$\gamma K$	$\frac{3}{5}K$

Tabuľka 2.6: Ceny operácií LU dekompozície na spojitých dátach pre zmiešané zostatkové derivácie.

jedného výrazu. Tu sa následne prejaví superskalárnosť výpočtových jednotiek procesorového jadra podľa časti 2.1. Moderné procesory architektúry x86 obsahujú práve dve jednotky pre výpočty s pohyblivou desatinnou čiarkou čo znamená  $\beta = 2$ . To implikuje zaujímavý dôsledok, kedy algoritmus s väčším počtom matematických operácií je v praxi rýchlejší ako algoritmus s menším celkovým počtom operácií, ale s viacerými na sebe závislými výrazmi (t.j. prípad, keď výraz  $b$  musí byť vyhodnotený až po výraze  $a$ ).

V prípade zmiešaných derivácií používame odlišné pravé strany rovníc pre LU dekompozíciu. Osobitne si spočítajme ceny aj pre tieto procedúry.

Analogicky ako v predchádzajúcej sekcii položíme procedúry predstavujúce implementáciu redukovaného algoritmu podľa časti 1.3.:

- Procedúra *VypočítajDxResty* vypočíta zvyšné parciálne derivácie  $d^x$  podľa 1.14.
- Procedúra *VypočítajDx* vypočíta parciálne derivácie  $d^x$  pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.13. Jedno volanie *VyriešLU* vypočíta derivácie na párnych riadkoch  $d^x$  pre jeden stĺpec. Teda procedúra musí počítať LU pre každý stĺpec, ktorých je  $J$ . Zvyšné derivácie dopočítame procedúrou *VypočítajResty*.
- Procedúra *VypočítajDyResty* vypočíta zvyšné parciálne derivácie  $d^y$  podľa 1.16.
- Procedúra *VypočítajDy* vypočíta parciálne derivácie  $d^y$  pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.15. Jedno

Procedúra	$\pm$	$\cdot$	$\div$	$\div$
<i>VypočítajDx</i>	$3/2IJ + 3/2\beta IJ$	$1IJ + 1/\beta IJ$	$1/2\gamma IJ$	$3/10IJ$
<i>VypočítajDy</i>	$3/2IJ + 3/2\beta IJ$	$1IJ + 1/\beta IJ$	$1/2\gamma IJ$	$3/10IJ$
<i>VypočítajDxy</i>	$8I + 8J$	$6I + 6J$	$2I + 2J$	$I + J$
<i>VypočítajDyx</i>	$3/4IJ + 33/4\beta IJ$	$1/2IJ + 17/4\beta IJ$	$1/4\gamma IJ$	$3/25IJ$

Tabuľka 2.7: Ceny operácií na spojitých dátach redukovaného algoritmu.

volanie *VyriešLU* vypočíta derivácie na párnych stĺpcoch  $d^y$  pre jeden riadok. Teda procedúra musí počítať LU pre každý riadok, ktorých je  $I$ . Zvyšné derivácie dopočítame procedúrou *VypočítajResty*.

- Procedúra *VypočítajDxy* vypočíta parciálne derivácie  $d^{xy}$  pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.17 a 1.18. Jedno volanie *VyriešLU* vypočíta na párnych riadkoch(stĺpcoch) derivácie  $d^{xy}$  pre jeden stĺpec(riadok). Procedúra musí počítať LU pre prvý a posledný stĺpec a pre prvý a posledný riadok.
- Procedúra *VypočítajZmiešResty* vypočíta zvyšné parciálne derivácie  $d^{xy}$  podľa 1.20, 1.21 a 1.22.
- Procedúra *VypočítajDyx* vypočíta parciálne derivácie  $d^{xy}$  pomocou procedúr *InicalizujZmiešLU* a *VyriešZmiešLU*. Vstupné hodnoty vezmeme z 1.19, pričom na ich inicializovanie do LU dekompozície použijeme práve procedúru *InicalizujZmiešLU*. Jedno volanie *VyriešLU* vypočíta na párnych stĺpcoch  $d^{xy}$  pre jeden riadok. Procedúra musí počítať LU pre každý párny riadok, ktorých je  $I$ . Zvyšné derivácie dopočítame procedúrou *VypočítajZmiešResty*.
- Procedúra *VypočítajRedukovaný* vypočíta na základe vstupných hodnôt pre de Boorovu interpoláciu 1.2 postupným vykonaním predchádzajúcich siedmich procedúr.

Opäť pre zjednodušenie budeme počty v tvare  $a \cdot IJ + b \cdot I + c \cdot J + d$  zanedbávať na  $a \cdot IJ$ . Ak hodnota  $I$  značí počet uzlov na osi  $x$  a hodnota  $J$  značí počet uzlov na osi  $y$  tak pre redukovaný algoritmus dostaneme nasledujúce počty nad kešovanými dátami

Najprv si spočítajme procedúry pre resty.

Sčítaním operácií v predchádzajúcich dvoch tabuľkách dostaneme nasledujúci výsledok.

Procedúra	$\pm$	$\cdot$	$\div$	$\div \cdot$
<i>VypočítajDxResty</i>	$3/_{2\beta}IJ$	$1/_{\beta}IJ$	0	0
<i>VypočítajDyResty</i>	$3/_{2\beta}IJ$	$1/_{\beta}IJ$	0	0
<i>VypočítajZmiešResty</i>	$17/_{4\beta}IJ$	$7/_{4\beta}IJ$	0	0

Tabuľka 2.8: Ceny operácií pre zmiešané zostatkové derivácie.

**Lemma 2.12** *Nech  $I$  a  $J$  označujú počty uzlov na osiach  $x$  a  $y$ . Potom sumárny počet operácií plného algoritmu je*

- $15/_{4}IJ + 37/_{2\beta}IJ$  sčítaní,
- $5/_{2}IJ + 10/_{\beta}IJ$  násobení,
- $5/_{4\gamma}IJ$  delení,
- $0,75IJ$  kopírovaní.

Cena redukovaného algoritmu teda je

$$7IJ + 57/_{2\beta}IJ + 5/_{4\gamma}IJ.$$

### 2.2.3 Pamäťové nároky

Redukovaný algoritmus by nemal byť lepší len v počte aritmetických operácií, ale taktiež priniesť menšie pamäťové nároky vyplývajúce z polovičnej veľkosti sústav rovníc riešených LU dekompozíciou. Uvažujme maticu uzlov veľkosti  $I$  a  $J$ . Potrebujeme si pamätať  $I \cdot J$  funkčných hodnôt v uzloch, derivácií podľa  $x$ ,  $y$  a zmiešaných derivácií. Samotné uzly si pamätať nemusíme lebo uvažujeme len uniformné splajny. Teda stačí si nám uložiť iba uzly  $u_0, u_{I-1}, v_0, v_{J-1}$ . Zvyšné uzly si vieme vypočítať vzorcom  $u_i = u_0 + |u_{I-1} - u_0| \cdot \frac{i}{I}$  pre ľubovoľné  $i$  z  $\{1, \dots, I-2\}$ , respektíve  $v_j = v_0 + |v_{J-1} - v_0| \cdot \frac{j}{J}$  pre ľubovoľné  $j$  z  $\{1, \dots, J-2\}$ . Potrebujeme si teda pamätať aspoň  $4 \cdot I \cdot J + 4$  čísel pre funkčné hodnoty uzlov.

Počítanie všeobecnej LU dekompozície podľa algoritmu ?? potrebuje päť vektorov, kde tri predstavujú trojdiagonálu, jeden pravú stranu, v ktorom na konci bude výsledok. Nám ale stačí uvažovať iba procedúru pre náš špeciálny tvar trojdiagonálnej sústavy podľa algoritmu 1. Pri použití tejto priestorovo optimalizovanej varianty výpočtu LU dekompozície potrebujeme na vypočítanie  $I \cdot J$  uzlov práve  $I \cdot J + 2I + 2J$  pamäte.

## 2.2.4 Zhrnutie

Po spočítaní jednotlivých matematických a pamäťových operácií v častiach 2.2.1 a 2.2.2 pristúpime k formulácii zrýchlenia.

**Veta 2.13** *Nech  $I$  a  $J$  označujú počty uzlov na osiach  $x$  a  $y$ . Potom zrýchlenie redukovaného algoritmu je*

$$\frac{528/25IJ + 3\gamma IJ}{7IJ + 57/2\beta IJ + 5/4\gamma IJ}.$$

Ukážme si dva príklady zrýchlenia aby sme názorne videli vplyv superskalárnosti na rýchlosť redukovaného algoritmu. V prvom príklade predpokladajme porovnanie algoritmov na primitívnom procesore.

**Príklad 2.14** Nech  $\beta = 1$ , teda procesorové jadro má iba jednu jednotku pre výpočty s pohyblivou desatinnou čiarkou a  $\gamma = 3$ , to jest operácia delenia je trojnásobne pomalšia ako operácia sčítania podľa tabuliek 2.1 a 2.2 operácií v časti 2.1.1. ??),  $\iota = 1$ ,  $\lambda = 1$ . Dosadením do vzťahu z predchádzajúcej vety dostaneme

$$\frac{30,12}{36,75} \approx 0,77.$$

Pomer je menší ako 1 čo znamená, že bez superskalárnosti ( $\beta = 1$ ) je redukovaný algoritmus pomalší ako plný.

**Príklad 2.15** Nech  $\beta = 2$ , teda procesorové jadro má práve dve jednotky pre výpočty s pohyblivou desatinnou čiarkou a  $\gamma = 3$ . Analogicky ako v predchádzajúcom príklade dosadením do vzťahu dostaneme

$$\frac{30,12}{25} \approx 1,2.$$

V prípade superskalárnosti s faktorom 2 je redukovaný algoritmus o 20% rýchlejší ako plný.

Podľa príkladov vidíme, že rýchlosť redukovaného algoritmu je závislá na schopnosti hardvérovej architektúry procesora paralelizovať vyhodnotenia aritmetických výrazov s viacerými operandami. Je otázne nakoľko by s rastúcim faktorom  $\beta$  rástlo zrýchlenie. Keďže nemáme k dispozícii procesor s vhodnou hardvérovou výbavou (ak vôbec taký existuje) je náročné predpokladať ako by si takýto stroj s redukovaným postupom poradil.

V nasledujúcej časti si ukážeme reálne výsledky a uvidíme či s nimi vy počítané zrýchlenie skutočne súhlasí.

Procesor	Krivka		Sériovo	
	Plný	Reduk.	Plný	Reduk.
FX-6300 <sub>3C/6T</sub>				
C2D E8200 <sub>2C/2T</sub>				
Ci5 650 <sub>2C/4T</sub>				
Ci3 2350M <sub>2C/4T</sub>	66	53	153	131
Ci5 4440 <sub>4C/4T</sub>				
Ci7 6700K <sub>4C/8T</sub>	29	25	55	49

Tabuľka 2.9: Reálne merania plného a redukovaného algoritmu na mriežke  $1000 \times 1000$  uzlov.

## 2.3 Merané zrýchlenie

V kapitole Zrýchlenie sme určili teoretické zrýchlenie počítania uzlov, dosiahnuteľné zredukovaním veľkosti trojdiagonálnych sústav. V tejto časti si ukážeme reálne výsledky dosiahnuté v ukážkovej implementácii.

Ako bolo spomenuté, testovacia aplikácia bola implementovaná v jazyku C++. Program obsahuje testy pre sekvenčné aj paralelné počítanie derivácií pre oba predmetné algoritmy. Použitý prekladač bol Intel C++ Compiler v 64 bitovej verzii nastavený na generovanie agresívne optimalizovaného binárneho kódu (-O2).

Testy boli vykonané na šiestich rôznych počítačových zostavách, všetky so systémom Windows 7 a 10. Testovacie stroje obsahujú rozličné multivláknové procesory od starého Penryn z roku 2007 až po najmodernejší Skylake z roku 2015 so vzájomne odlišnými architektúrami a hlavne spôsobmi vykonávania paralelizovaných procesov.

Stĺpec 1 obsahuje modely procesorov zoradených podľa mikroarchitektúry ako v tabuľke 2.1. Vedľa názvu modelu CPU sa nachádza údaj tvaru  $nC/mT$ , kde  $n$  je počet fyzických jadier procesora a  $m$  počet logických jadier. V prípade  $m > n$  je procesor vybavený určitou formou SMT. Stĺpce 2 a 3 predstavujú časy behov sekvenčných implementácií plného a redukovaného algoritmu v prípade krivkových splajnov. Stĺpce 4 a 5 predstavujú časy behov sériovej verzie pre povrchové splajnov.

Ak je táto hypotéza korektná tak by bolo potrebné upraviť resty, najmä pri zmiešaných deriváciách, aby sa ich výpočet dal vykonať paralelne v jednom kroku spolu s LU dekompozíciou

Táto kapitola sa venovala počítaniu zrýchlenia redukovaného algoritmu vzhľadom na plný algoritmus. Ako sme videli, teoretické zrýchlenie je v súlade

s nameraným zrychlením v závislosti od typu procesora.

## Záver

Podarilo sa nám v prípade rovnomerne rozložených uzlov urýchliť výpočet derivácií splajnov v uzloch a tiež zmenšiť pamäťovú náročnosť výpočtu na polovicu. Napriek tomu, že výsledný redukovaný algoritmus obsahuje viac aritmetických operácií je vďaka povahe týchto operácií rýchlejší o približne 20%. Prioritou teraz ostáva jednak zovšeobecniť redukovaný algoritmus aj pre splajny s nerovnomernými uzlami a upraviť zmiešané zbytkové derivácie aby sme mohli zefektívniť paralelizáciu. Paralelné počítanie uzlov redukovaným algoritmom nie je v súčasnosti rýchlejšie vo všeobecnosti, ale závisí na konkrétnej procesorovej architektúre. Výhodou ale ostávajú polovičné pamäťové nároky čo umožňuje riešiť väčšie úlohy a tak rozdeliť prácu medzi väčší počet logických procesorov. To by v budúcnosti umožnilo výpočty akcelerovať použitím masívne paralelizovaných čipoch ako sú napríklad grafické karty.



# Zoznam použitej literatúry

- [1] David Salomon, Curves and Surfaces for Computer Graphics, Springer, 2006
- [2] I. Szabó, L. Miño, C. Török, Biquartic polynomials in bicubic spline construction, PF UPJŠ, 2014
- [3] Lukáš Miño, Csaba Török, Fast algorithm for spline surfaces, PF UPJŠ, 2015
- [4] Lukáš Miño, Parametrické modelovanie dát komplexnej štruktúry, PF UPJŠ, 2014
- [5] C. de Boor, Bicubic spline interpolation, Journal of Mathematics and Physics, 41(3),1962, 212-218.
- [6] J. Albahari, B.Albahari, C# 5.0 in a Nutshell, O'Reilly, 2012
- [7] E. Süli, D. Mayers, An Introduction to Numerical Analysis, Cambridge University Press, 2003, ISBN 0 521 00794 1
- [8] [https://en.wikibooks.org/wiki/Algorithm\\_Implementation/Linear\\_Algebra/Tridiagonal\\_matrix\\_algorithm](https://en.wikibooks.org/wiki/Algorithm_Implementation/Linear_Algebra/Tridiagonal_matrix_algorithm)
- [9] <http://www.anandtech.com/show/9483/intel-skylake-review-6700k-6600k-ddr4-ddr3-ipc-6th-generation/9>
- [10] <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf>
- [11] [http://www.agner.org/optimize/instruction\\_tables.pdf](http://www.agner.org/optimize/instruction_tables.pdf)
- [12] <http://www.lighterra.com/papers/modernmicroprocessors/>
- [13] <http://techreport.com/review/23750/amd-fx-8350-processor-reviewed/3>