

Univerzita Pavla Jozefa Šafárika v Košiciach
Prírodovedecká fakulta

ZRÝCHLENIE VÝPOČTU SPLAJN POVRCHOV

DIPLOMOVÁ PRÁCA 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 záverečnej 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 odbornú pomoc počas tvorby aplikáčnej časti tejto práce.

Zadanie práce

Abstrakt

Splajny sú dôležitá súčasť počítačovej grafiky. Jedná sa o matematický model plochy, ktorá slúži na čo „najlepšie spojenie“ konečnej množiny bodov. Termín „najlepšie spojenie“ v našom prípade znamená hladká, matematicky ľahko vyjadriteľná plocha 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 the surface, which is 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 new algorithm for counting and generating splines bicubic clamped interpolation in three-dimensional space.

Obsah

1	De Boorov algoritmus a návrh zrýchlenia	7
1.1	Polynómy	7
1.2	Splajny	8
1.3	LU dekompozícia	12
1.4	De Boorova interpolácia	12
1.5	Počítanie derivácií redukovanou sústavou	15
1.6	Konštrukcia splajnu	19
2	Zrýchlenie	20
2.1	Procesorová architektúra	20
2.1.1	Kešovanie	21
2.1.2	Inštrukčný paralelizmus	23
2.2	Počty operácií	24
2.2.1	Počty pre plný algoritmus	25
2.2.2	Počty pre redukovaný algoritmus	26
2.2.3	Zhrnutie	28
2.3	Zrýchlenie v praxi	30
3	Implementácia a užívateľská príručka	32
3.0.1	Vizualizácia	34
3.0.2	Výkonnostný tester	36

Úvod

Témou mojej diplomovej práce sú priestorové splajn povrchy, kde naším cieľom je preskúmať nové poznatky o splajnoch, na ktorých istý čas pracuje vedúci tejto práce doc. RNDr. Csaba Török, CSc. Výsledkom tejto práce je návrh novej metódy výpočtu, jej porovnanie s de Boorovým postupom z ktorého vychádzame a implementácia aplikácií na vizuálne a výkonnostne porovnávanie oboch metód. Technológia v ktorej tieto poznatky implementujeme je Microsoft Silverlight. Jedná sa o veľmi schopný nástroj na tvorbu webových aplikácií s plnou podporou hardvérovej akcelerácie užívateľského prostredia a možnosťou tvorby rýchlej trojrozmernej grafiky. Na výkonnostné porovnanie máme implementovanú samostatnú aplikáciu. Výhodou tohto frameworku je, keďže beží na platforme Microsoft .NET, možnosť jednoduchej portácie na desktopovú prípadne mobilnú aplikáciu. Na výkonnostné porovnanie nášho algoritmu s existujúcim však implementujeme samostatnú aplikáciu nad C++.

Štruktúra práce je nasledovná.

- **De Boorov algoritmus a návrh zrýchlenia**

- **Splajn plochy**

- Definícia pojmu splajn.

- **De Boorova interpolácia**

- V tejto časti si vysvetlíme bikubickú splajn interpoláciu podľa Carla de Boora.

- **Počítanie derivácií redukovanou sústavou**

- Ukážka modifikovaného postupu pre kubické splajny a jeho rozšírenie pre bikubické splajny.

- **Zrýchlenie**

- Očakávané zrýchlenie výpočtov novým algoritmom.

- **Implementácia**

- Podrobnosti implementácie v MS Silverlight a C++. Namerané výsledky.

- **Záver**

Formálne tézy diplomovej práce sú:

- Analýza modelov interpolačných splajnov.
- Redukovaný algoritmus výpočtu koeficientov splajn povrchov.

- Testovanie faktorov vplývajúcich na rýchlosť výpočtu splajn koeficientov
 - Klasický a Hermitov splajn
 - De Boorov a redukovaný algoritmus
 - Vplyv latencie a priepustnosti procesorových inštrukcií a vyrovnávacích pamätí.

Kapitola 1

De Boorov algoritmus a návrh zrýchlenia

V tejto kapitole si postupne zadefinujeme pojem splajnu s ktorým budeme pracovať.

1.1 Polynómy

Pre úplnosť skôr než si povieme o splajnoch, predstavíme si pojem polynomickej funkcie ktorý neoddeliteľnou súčasťou pri definícii splajnu.

Definícia 1.1 Nech n je z $\mathbb{N} \cup \{0\}$ a pre každé i z $\{0, \dots, n\}$ je a_i z \mathbb{R} , pričom $a_n \neq 0$. Funkciu $p: \mathbb{R} \rightarrow \mathbb{R}$ tvaru

$$p(x) = \sum_{i=0}^n a_i x^i$$

nazveme *polynomická funkcia jednej premennej stupňa n* .

Označenie 1.2 Polynomicke funkcie jednej premennej budeme skrátené nazývať *polynomicke funkcie*. Špeciálne polynomicke funkcie stupňa 3 nazveme *kubické funkcie*.

V práci však budeme pracovať najmä s funkciami dvoch premenných. Analogicky si pre dve premenné zadefinujeme aj polynomicke funkcie.

Definícia 1.3 Nech n a m sú z $\mathbb{N} \cup \{0\}$ a pre každé i z $\{0, \dots, n\}$ a j z $\{0, \dots, m\}$ je a_{ij} z \mathbb{R} , pričom $a_{nm} \neq 0$. Funkciu $p: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ tvaru

$$p(x, y) = \sum_{i=0}^n \sum_{j=0}^m a_{ij} x^i y^j$$

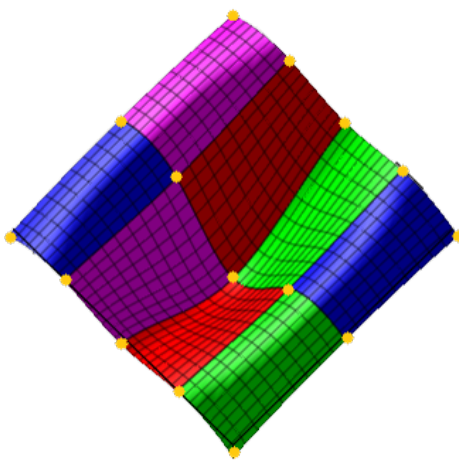
nazveme *polynomická funkcia dvoch premenných stupňov n a m*

Označenie 1.4 Polynomické funkcie dvoch premenných budeme skrátené nazývať *bipolynomické funkcie*. Bipolynomické funkcie stupňov 3 a 3 nazveme *bikubické funkcie*.

1.2 Splajny

V našej práci pracujeme s hermitovskými splajnami, ktoré sú štandardne triedy C^1 , teda splajnami, ktorých prvé derivácie v uzloch sa rovnajú. Pre naše potreby budeme uvažovať iba hermitovské splajny triedy C^2 , teda tie pri ktorých v uzloch máme zaručenú rovnosť aj derivácií druhého rádu. Všeobecne sa splajny triedy C^2 nazývajú *naturálne splajny*[6].

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 zdefinovať splajn všeobecne, ale rozdelenie definície špeciálne pre krivkové splajny v rovine a povrchové v priestore bude čitateľnejšie.



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

Formálnejšie si popíšme základné pojmy s ktorými budem v tomto článku pracovať.

Označenie 1.5 Postupnosť (a_0, a_1, \dots) nazveme *rovnomere 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 $a_{j+1} - a_j = a_{i+1} - a_i$.

Formálna definícia krivkového splajnu v rovine je nasledovná.

Definícia 1.6 Nech $I \geq 0$ je prirodzené číslo, (u_0, \dots, u_{I-1}) je rovnomerne 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:

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

nazveme *krivkový splajn*.

Označenie 1.7 Pri označeniach z predchádzajúcej definície označme:

- Funkcie S_i nazveme *segmenty*.
- 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 $i \in \{1, \dots, I-1\}$.

Definíciu krivkového splajnu následne rozšírime na povrchový splajn.

Definícia 1.8 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*.

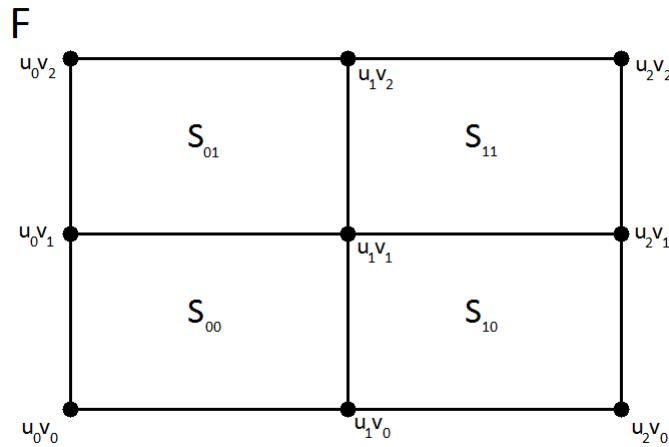
Označenie 1.9 Pre úplnosť analogicky ako pri krivkách označme:

- Funkcie S_{ij} nazveme *segmenty splajnu*.
- Dvojice (u_i, v_j) nazveme *uzly*.
- Uzly (u_0, v_0) , (u_{I-1}, v_0) , (u_0, v_{J-1}) a (u_{I-1}, v_{J-1}) nazveme *rohové 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 *x-ové derivácie v uzloch*.
- $d_{i,j}^y = \frac{\partial S(u_i, v_j)}{\partial y}$ nazveme *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. $j \in \{1, \dots, J-1\}$.

Prirodzene vidieť, že takto definovaný krivkový a 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. Ak uvažujeme ľubovoľnú funkciu tak pomocou nej vieme nájsť žiadané hodnoty derivácií. Navyše takto

získaný splajn bude danú funkciu interpolovať, teda ju „napodobniť“. 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. Navyše sa dá ukázať, že interpolácia funkcie splajnom je jednoznačná. Témou tejto práce sú práve metódy výpočtu prvých derivácií, ktorými je možné dosiahnuť práve spomenutý cieľ.

Témou následovných častí práce je úloha na základe vstupných uzlov a funkcie $d : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ nájsť *hladkú*, po častiach definovanú funkciu $S : [u_0, u_{I+1}] \times [v_0, v_{J+1}] \rightarrow \mathbb{R}$ z 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 $z_{i,j} = f(u_i, v_j)$ a hodnoty derivácií S v uzloch sú čo najbližšie hodnotám derivácií f . Funkciu S nazývame splajn (konkrétne povrchový splajn), pričom jednotlivé časti nazveme *segmenty*.



Obr. 1.2: Ukážka uzlov pre štvorsegmentový splajn.

Ak i je z $\{0, \dots, I-2\}$ a j je z $\{0, \dots, J-2\}$, tak štvorica 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 $[u_i, u_{i+1}] \times [v_j, v_{j+1}]$ 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 , d^x , d^y a d^{xy} . Tieto hodnoty vieme získať buď priamo z interpolovanej funkcie f , alebo ich vypočítať iným spôsobom, napríklad De Boorovou interpoláciou.

1.3 LU dekompozícia

Definícia 1.10 Nech $n \geq 3$ je z $\mathbb{N} \cup \{0\}$. Sústavu rovníc tvaru

$$\begin{pmatrix} b_1 & a_1 & 0 & \cdots & 0 & 0 \\ c_1 & b_2 & a_2 & \cdots & 0 & 0 \\ 0 & c_2 & b & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & b & a_{n-1} \\ 0 & 0 & 0 & \cdots & c_{n-1} & b_n \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_{n-1} \\ r_n \end{pmatrix} \quad (1.3)$$

nazveme *trojdiagonálna sústava lineárnych rovníc*.

Základ oboch postupov tkvie v opakovanom počítaní systémov trojdiagonálnych lineárnych rovníc. Jeden so spôsobov riešenia týchto rovníc spočíva v LU dekompozícii $A\mathbf{x} = L \underbrace{U\mathbf{x}}_{\mathbf{y}} = \mathbf{r}$:

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ \lambda_2 & 1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_3 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & \lambda_n & 1 \end{pmatrix} \cdot \begin{pmatrix} v_1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & v_1 & 1 & \cdots & 0 & 0 \\ 0 & 0 & v_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & v_{n-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & v_n \end{pmatrix}, \quad (1.4)$$

Pre k z $\{2, \dots, n\}$ sú hodnoty v_k a λ_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).

1.4 De Boorova interpolácia

Nech sú dané hodnoty (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 $[u_0, u_{I-1}] \times [v_0, v_{J-1}]$. Výsledný splajn bude

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 priamo z f môže byť v praxi na počítači nákladné najmä v prípade, keď pracujeme so symbolicky zapísanou funkciou vo forme textového reťazca, ktorú je potrebné interpretovať.

Algoritmus nájdený Carlom de Boorom umožňuje s malou odchýlkou odhadnúť hodnoty derivácií v uzloch na základe týchto vstupných hodnôt, ktoré potrebujeme získať 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 1.11 Na príklade uzlov z obrázka 1.2 potrebujeme poznať hodnoty nasledovne:

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

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 funkciu jednoznačne interpolovať splajnom.

Poznámka 1.12 V aplikácií na vizualizáciu používame práve Hermitove splajny.

Poznámka 1.13 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 sme v tejto časti popísali len špeciálny prípad interpolácie pre takto určené uzly.

1.5 Počítanie derivácií redukovanou sústavou

V rámci svojej bakalárskej práce som popisoval nový postup pre kubické splajnové krivky triedy C2, 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 $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$.

Označenie 1.14 Teraz popíšeme tento rozšírený algoritmus, ktorý pracovne označíme ako *redukovaný algoritmus*. Ďalej budeme pôvodný de Boorov postup, označovať pojmom *plný algoritmus*.

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 $d^{x,y}$

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

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

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

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

Následne vypočítame zvyšné derivácie $d^{x,y}$. Pre $i \in \{1, 3, \dots, I_l\}$ a $j \in \{1, 3, \dots, J_l\}$ platí

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

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 1.15

- 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:

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

kde

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

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} 1.19.

Označenie 1.16

- 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*.

1.6 Konštrukcia splajnu

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} , funkciu dvoch premenných $f : [u_0, u_{I-1}] \times [v_0, v_{J-1}] \rightarrow \mathbb{R}$. 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 inicializujeme ich podľa 1.4.

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 1.8. Pre každé i z $\{0, \dots, I-2\}$ a j z $\{0, \dots, J-2\}$ položme 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

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

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

Kapitola 2

Zrýchlenie

Časová zložitosť oboch algoritmov je rovnaká a síce $O(I \cdot J)$. Redukovaný algoritmus je však teoreticky, a aj prakticky (2.3) rýchlejší, navyše s menšími pamäťovými nárokmi. Keďže porovnávame asymptoticky rovnako rýchle algoritmy, musíme brať do úvahy vplyv „nízkoúrovňových“ operácií, konkrétne rýchlosti výpočtu sčítania, násobenia, delenia a na koniec aj samotný vplyv pamäťových operácií čítania a zapisovania. V tejto kapitole si spočítame počty týchto operácií.

Na moderných procesoroch v prípade matematických operácií s plávajúcou desatinnou čiarkou platí, že sčítanie a násobenie sú rovnako rýchle, pričom delenie je asi desaťnásobne pomalšie. Pamäťové operácie sú pritom v závislosti od konkrétneho typu operačnej pamäte je desaťnásobne až stonásobne pomalšie ako sčítanie, resp. násobenie.¹

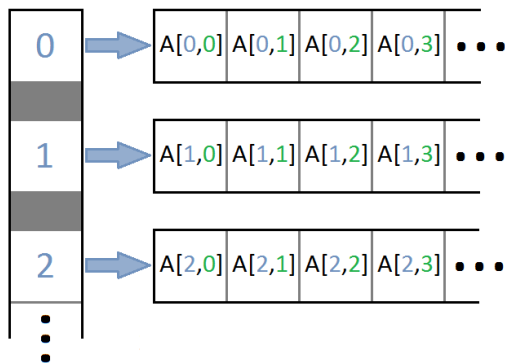
2.1 Procesorová architektúra

Predtým ako začneme s počítaním operácií je nutné objasniť si ako moderné procesory narábajú s dátami a ako funguje aplikovanie výpočtov. Architektúry CPU prešli za posledné desaťročia značným vývojom. V dnešnej dobe už nemožno zvyšovať výpočtovú rýchlosť hrubou silou a zvyšovaním frekvencie. Moderné procesory často používajú rôzne optimalizačné „triky a finty“ ako dosiahnuť zrýchlenie a ktoré majú značný vplyv na reálne rýchlosti algoritmov. Dvomi takými optimalizáciami sú takzvané kešovanie a inštrukčný paralelizmus.

¹V praxi sa aj výpočty rádovo miliónov uzlov často zmestia do vyrovnávacej pamäte procesora (*cache*). V tom prípade sú pamäťové operácie len 1×-10× pomalšie ako sčítanie.

2.1.1 Kešovanie

Pri návrhu algoritmu je vhodné si uvedomiť tvar dátových štruktúr do ktorých pri reálnej implementácii budeme ukladať funkčné hodnoty, uzly a derivácie. Naša implementácia v prípade kriviek používa klasické polia teda vektory. Pri povrchoch sa hodnoty reprezentujú maticami. Maticu je možné interpretovať buď ako „pole polí“ respektíve „vektor vektorov“ (anglicky „jagged array“), alebo ako jedno spojité pole, kde prvok na i -tom riadku a j -tom stĺpci má v takejto reprezentácii index $n * i + j$, kde n je počet stĺpcov. My budeme v meraniach a výpočtoch uvažovať práve prvú reprezentáciu znázorненú na obrázku nižšie.



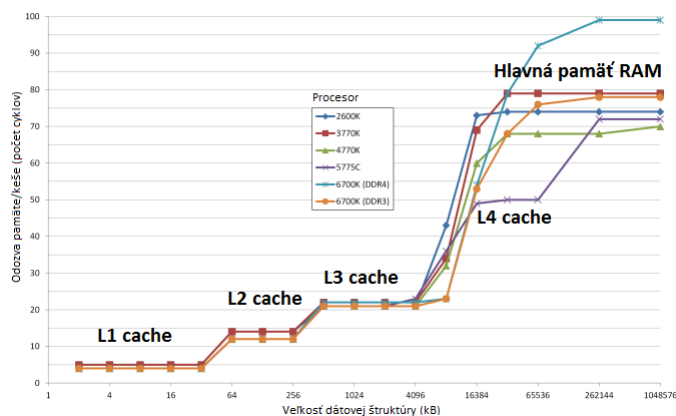
Obr. 2.3: Vizuálna reprezentácia dátovej štruktúry „pole polí“.

Operačná pamäť v dnešných počítačoch je rádovo pomalšia ako procesor, ktorý musí čakať desiatky až stovky strojových cyklov ak chce načítať hodnotu z pamäte. Moderné procesory preto používajú systém malých, ale rýchlych vyrovnávacích pamätí[12] tzv. *cache* do ktorých sa prednačítavajú dáta a strojové inštrukcie programu z operačnej pamäte, v ideálnom prípade skôr ako ich procesor bude potrebovať. Toto kešovanie je plne automatický proces riadený samotným CPU. Programátor má len nepriame možnosti ovplyvnenia tohto procesu, napríklad vhodným výberom dátových štruktúr a podobne.

Uvažujme maticu $m \times n$ typu pole polí ako z obrázka vyššie. Ak z matice načítame prvok $a_{0,0}$ tak je vysoko pravdepodobné, že procesor následne bude mať kešovaný aj prvky $a_{0,1}$, $a_{0,2}$ pretože sú z jedného poľa a teda sú v pamäti „vedľa seba“ (CPU kešuje pamäť po spojitých blokoch), ale už nemusí mať kešovaný prvok $a_{1,0}$ lebo tento je už z iného poľa a teda z úplne inej oblasti pamäte. Z toho vyplýva, že výraz $a_{0,0} + a_{0,1}$ procesor vyhodnotí rýchlejšie ako výraz $a_{0,0} + a_{1,0}$.

Pozorný čitateľ si môže všimnúť, že v prípade použitia druhej reprezentácie matice, teda jedného veľkého spojitého poľa dĺžky $m \cdot n$, pravdepodobnosť kešovania aj prvku $a_{1,0}$ by sa zvýšila, keďže je v tomto prípade sa bude nachádzať v tom istom bloku pamäte ako prvok $a_{0,0}$ iba o n prvkov (t.j. jeden celý stĺpec) ďalej. Toto je pravda, všeobecne sa druhá menovaná reprezentácia považuje za efektívnejšiu vzhľadom na procesor a pamäť počítača, ale kešovanie bude efektívne len pri malých maticiach najmä s malým počtom stĺpcov n .

V aplikácii sme totiž vyskúšali obe varianty reprezentácie a pre matice s rádom 1000×1000 prvkami sa ako efektívnejšia ukazuje práve prvá reprezentácia, teda „pole polí“. Totiž ak jeden prvok matice má 8 bajtov (veľkosť číselného typu *double* vo väčšine jazykoch), tak matica zaberá v pamäti približne 7,63 megabajtov. Pri matici typu pole polí máme 1000 polí každé zaberajúce práve 7,81 kilobajtov ($1 \text{ kB} = 1024 \text{ B}$, $1 \text{ MB} = 1024 \text{ kB}$, atď.). Pre CPU je kešovanie takýchto polí podstatne jednoduchšie ako v prípade matice reprezentovanej jedným poľom, ktoré samotné má veľkosť práve 7,63 MB, čo presahuje kapacitu väčšiny vyrovnávacích pamätí. Obrázok nižšie názorne ukazuje vzťah medzi veľkosťou dátovej štruktúry v kB a počtom strojových cyklov, ktoré procesor musí čakať pre prístup k jej prvkom. Ako vidieť do pri dosiahnutí veľkosti 32 kB, dáta sú namiesto veľmi rýchlej vyrovnávacej pamäte L1 kešované do trojnásobne pomalšej vyrovnávacej pamäte L2 atď..



Obr. 2.4: Graf rýchlosti prístupu k dátam rôznych veľkostí na moderných CPU[8].

2.1.2 Inštrukčný paralelizmus

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 a/alebo L2 cache. Každé jadro môže spracovávať na sebe nezávislé procesy alebo jeden proces môže byť rozdelený do takzvaných vlákien, kde každé môže byť spracovávané iným jadrom. V tom prípade hovoríme o takzvanom *vláknovom paralelizme* s ktorým sa stretávame najmä pri programovaní vo vyšších jazykoch. V našej implementácii testujeme vláknovo paralelizované aj klasické sériové verzie algoritmov.

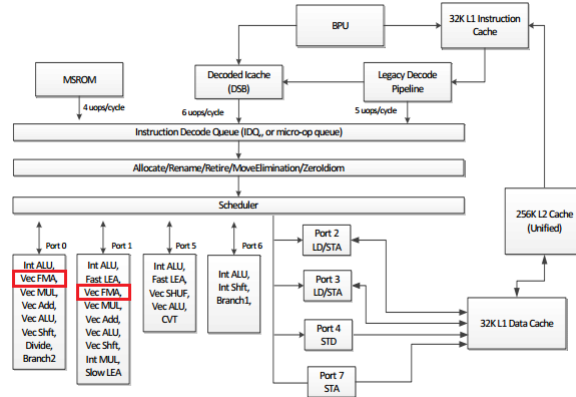
Ďalšie úrovne paralelizácie majú spoločné označenie *inštrukčný paralelizmus*. Pod týmto pojmom rozumieme *superskalárnosť*, *pipelining* a *vektORIZÁCIA*. Pri tejto úrovni paralelizmu má programátor len obmedzené možnosti jeho ovplyvnenia, všetku ťažkú prácu obstará prekladač a potom inštrukčný plánovač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

Hlavný vplyv má totiž superskalárnosť. Najprv si vysvetlíme čo tento pojem znamená. Jadro je tvorené niekoľkými až desiatkami špecializovanými jednotkami ako sú aritmeticko logické jednotky, bitové posuvníky, numerické koprocessory a podobne. Rozdiel oproti vláknovému paralelizmu spočíva, že tieto jednotky vo všeobecnosti nedokážu fungovať samostatne a síce simultánne spracovávať viacero vlákien alebo procesov². Vedia iba v rámci jedného vlákna spracovávať niekoľko inštrukcií naraz za predpokladu ak sú na sebe nezávislé, t.j. výsledok jednej inštrukcie nezávisí na výsledku predchádzajúcej.

Tu ale paralelizácia nekončí. Aj samotné jednotky totiž dokážu simultánne spracovávať viac než jeden dátový prúd (tzv. *data stream*). Na tejto úrovni rozlišujeme medzi vektorizáciou, a pipeliningom. Prvá menovaná technika umožňuje jednu inštrukciu aplikovať na celé vektory, resp. polia. Z povahy výpočtov v našej práci nás ale vektorizácia nemusí zaujímať. Pipelining je založený na myšlienke, že princíp fungovania výpočtových jednotiek možno prirovnať k výrobnéj linke vo fabrike, kde nový výrobok na linku vstúpi skôr ako je predchádzajúci dokončený. Teda aj väčšina výpočtových jednotiek dokáže s novým výpočtom začať ešte pred dokončením práve prebiehajúcej operácie.

Spomenuli sme majoritný vplyv superskalárnosti. Na obrázku 2.5 vidíme schému jedného jadra procesorovej mikroarchitektúry Intel Skylake, ktorá

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



Obr. 2.5: Schéma výpočtového jadra mikroarchitektúry Intel Skylake[9].

je ku dňu písania práce najmodernejšou a na ktorej testujeme náš zrýchlenie redukovaného algoritmu. Jadrá (nielen) tejto architektúry sú vybavené dvomi sčítačko-násobičkami čísiel s plávajúcou desatinnou čiarkou, ktoré sú na obrázku zvýraznené červenou farbou. Pointa spočíva v možnosti výpočet výrazov typu (napr.) $a_0 + a_1 + \dots + a_n$, 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.

2.2 Počty operácií

Spočítajme počet elementárnych matematických operácií sčítania, násobenia a delenia pre oba algoritmy. To budeme počítat tak, že si zadefinujeme „procedúry“ ktoré predstavujú jednotlivé časti algoritmu. Potom z nich zostavíme pseudokód pre oba algoritmy. Začnime teda počítat počty operácií. Keďže všetky procesory použité pri meraní obsahujú dve jednotky pre operácie s plávajúcou desatinnou čiarkou tak pri počítaní budeme uvažovať, že pri výrazoch s viacerými operáciami vezmeme do úvahy len polovičný počet operácií. Najprv položíme procedúry predstavujúce LU dekompozíciu spoločne pre oba postupy.

- Procedúra *InicializujLU* inicializuje hodnoty r_0, \dots, r_{K-1} a 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 *InicalizujLU*.

2.2.1 Počty pre plný algoritmus

Počty základných aritmetických a pamäťových operácií pre vyššie uvedené procedúry môžeme zhrnúť do tabuľky, pričom uvažujeme počty v našej ukážkovej implementácii [7]. Riadky tabuľky predstavujú jednotlivé procedúry a stĺpce udávajú počet vykonaných operácií, pre základné matematické operácie, kde K je počet neznámych. Podotýkam, že operáciu odčítania budeme uvažovať ako sčítanie. Operácie MEM predstavuje kopírovanie vypočítaných derivácií do výslednej matice uzlov. Ďalej musíme odlíšiť operácie vykonané na spojitých a nespojitých dátach podľa sekcie 2.1.1 o kešovaní. V prípade nespojitých dát má značný vplyv na meranie práve prístup do nekešovaných častí pamäte. To či matematické operácie v nasledujúcich procedúrach budú kešované alebo nie záleží od kontextu v akom sú volané.

Procedúra \ Operácia	+	×	÷	M
<i>InicalizujLU</i>	K	K	0	0
<i>VyriešLU</i>	$4K$	$4,5K$	K	K

Uvažujme procedúry predstavujúce implementáciu plného algoritmu:

- Procedúra *VypočítajDx* vypočíta parciálne derivácie d^x pomocou procedúry *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 procedúry *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 procedúry *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 procedúry *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 1.4 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 počty operácií nad kešovanými dátami

Procedúra \ Operácia	+	\times	\div	M
<i>VypočítajDx</i>	$3IJ$	$3,5IJ$	0	0
<i>VypočítajDxy</i>	$6I$	$7I$	$2I$	0
<i>VypočítajDy</i>	$4IJ$	$4,5IJ$	0	IJ
<i>VypočítajDyx</i>	$4IJ$	$4,5IJ$	0	IJ
<i>VypočítajPlný</i>	$11IJ$	$12,5IJ$	$2IJ$	$2IJ$

Pre operácie nad nekešovanými dátami dostávame

Procedúra \ Operácia	+	\times	\div	M
<i>VypočítajDx</i>	IJ	IJ	IJ	IJ
<i>VypočítajDxy</i>	$2I$	$7I$	$2I$	m
<i>VypočítajDy</i>	0	0	0	0
<i>VypočítajDyx</i>	0	0	0	0
<i>VypočítajPlný</i>	IJ	IJ	IJ	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*.

Procedúra \ Operácia	+	\times	\div	M
<i>InicalizujLU</i>	$1,5K$	K	0	0
<i>VyriešLU</i>	$4,5K$	$4,5K$	K	K

Podobne ako pri plnom algoritme, to či operácie budú vykonané na kešovaných dátach závisí od toho z ktorej procedúry sú volané. Pri procedúrach *InicalizujZmiešLU* a *VyriešZmiešLU*, ktoré sú volané len v jednom kontexte ale vieme exaktne určiť ktoré operácie budú vykonané na kešovaných dátach a ktoré nie.

Procedúra \ Operácia	+	×	÷	M
<i>InicalizujZmiešLU</i>	0	0	0	0
<i>VyriešZmiešLU</i>	3K	3,5K	K	0

Procedúra \ Operácia	+	×	÷	M
<i>InicalizujZmiešLU</i>	16,5K	8,5K	0	0
<i>VyriešZmiešLU</i>	16,5K	8,K	0	K

Analogicky položíme procedúry predstavujúce implementáciu redukovaného algoritmu:

- Procedúra *VypočítajDxResty* vypočíta zvyšné parciálne derivácie d^x podľa 1.14.
- Procedúra *VypočítajDx* vypočíta parciálne derivácie d^x pomocou procedúry *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 procedúry *VyriešLU*. Vstupné hodnoty vezmeme z 1.15. Jedno 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 procedúry *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 procedúry *VyriešLU*. Vstupné hodnoty vezmeme z 1.19, pričom na ich inicializovanie do LU dekompozície použijeme 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 1.4 postupným vykonaním predchádzajúcich siedmich procedúr.

Najprv si spočítajme procedúry pre resty. Všimnime si, že všetky výpočty v *VypočítajDyResty* sú kešované ale výpočty v *VypočítajDxResty* nie.

Procedúra \ Operácia	+	×	÷	M
<i>VypočítajDyResty</i>	$1,5IJ$	IJ	0	$\sqrt[5]{2}IJ$
<i>VypočítajZmiešResty</i>	$7IJ$	$2,5IJ$	0	0

Pri nekešované výpočty v *VypočítajDxResty* dostávame

Procedúra \ Operácia	+	×	÷	M
<i>VypočítajDxResty</i>	$1,5IJ$	IJ	0	$\sqrt[5]{2}IJ$

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

Procedúra \ Operácia	+	×	÷	M
<i>VypočítajDx</i>	$1,5IJ$	$1,75IJ$	0	0
<i>VypočítajDy</i>	$3IJ$	$2,75IJ$	$0,5IJ$	$0,5IJ$
<i>VypočítajDxy</i>	$6I + 8J$	$7I + 9J$	$2I$	I
<i>VypočítajDyx</i>	$2,875IJ$	$1,75IJ$	$0,25IJ$	$0,25IJ$
<i>VypočítajRedukovaný</i>	$7,375IJ$	$6,25IJ$	$0,75IJ$	$0,75IJ$

Pre operácie nad nekešovanými dátami dostávame

Procedúra \ Operácia	+	×	÷	M
<i>VypočítajDx</i>	$1,5IJ$	IJ	$0,5IJ$	$0,5IJ$
<i>VypočítajDy</i>	0	0	0	0
<i>VypočítajDxy</i>	$2I$	$2I$	$2J$	J
<i>VypočítajDyx</i>	$4,175IJ$	$2,125IJ$	0	0
<i>VypočítajRedukovaný</i>	$2,675IJ$	$3,215IJ$	$0,5IJ$	$0,5IJ$

2.2.3 Zhrnutie

Teraz máme spočítané jednotlivé matematické a pamäťové operácie. V záujme prehľadnosti v rámci tejto podsekcie zavedme tieto označenia

- \succ označuje spojité dátové štruktúry ako napríklad vektory,

- \therefore označuje nespojité dátové štruktúry ako napríklad vektory referencií na nejaké dáta alebo spájané zoznamy.

Nech $t_{\text{operácia}}^{\text{dáta}}$ udáva dobu trvania nejakej operácie na nejakých dátach. Napríklad $t_+^{\text{spojité}}$ udáva čas sčítania dvoch rovnako veľkých spojitých kešovaných vektorov. Na začiatku sekcie sme zvolili predpoklad, že pri číselných typoch s pohyblivou čiarkou sú operácie sčítania (odčítania) a násobenia rovnako rýchle. Na všetkých testovaných procesoroch je totožne rýchla aj pamäťová operácia kopírovania pri spojitých dátach. Teda $t_+ = t_- = t_{\text{MEM}_c}$. Položme si teraz pomery udávajúce pomery dôb trvania jednotlivých operácií.

- $\gamma = \frac{t_+^r}{t_+^r}$
- $\eta = \frac{t_M^r}{t_M^r}$
- $\iota = \frac{t_+^r}{t_+^r}$
- $\lambda = \frac{t_+^r}{t_+^r}$

Celková cena algoritmu je teda súčet počtu všetkých operácií z tabuliek 2.2.1 a 2.2.1 pre plný algoritmus, respektíve 2.2.2 a 2.2.2 pre redukovaný algoritmus z predchádzajúcich podsekcí, pričom počty operácií delenia na kešovaných dátach vynásobíme γ , počty nekešovaných pamäťových operácií vynásobíme η , počty operácií sčítania a násobenia na nekešovaných dátach vynásobíme ι a nakoniec počty operácií delenia na nekešovaných dátach vynásobíme λ (pretože λ udáva pomer delenia a sčítania na nekešovaných dátach a to ešte treba vynásobiť ι).

Pre plný algoritmus teda dostávame

$$25,5IJ + 2\gamma IJ + \eta IJ + 2\iota IJ + \lambda \iota IJ, \quad (2.1)$$

a pre redukovaný máme

$$14,375IJ + 0,75\gamma IJ + 0,5\eta IJ + 5,84\iota IJ + 0,5\lambda \iota IJ. \quad (2.2)$$

Ak chceme určiť zrýchlenie redukovaného algoritmu stačí vydeliť 2.1 a 2.2 čím dostaneme

$$\frac{25,5IJ + 2\gamma IJ + \eta IJ + 2\iota IJ + \lambda \iota IJ}{14,375IJ + 0,75\gamma IJ + 0,5\eta IJ + 5,84\iota IJ + 0,5\lambda \iota IJ}. \quad (2.3)$$

Definícia 2.1 Výraz 2.3 nazveme *pomer rýchlostí*.

Príklad 2.2 Ak budeme predpokladať $\gamma = 10$, to jest operácia delenia je desaťnásobne pomalšia ako operácia sčítania tak a budeme uvažovať maticu uzlov s počtom $I \times J = 1000 \times 1000$ tak pomer rýchlostí bude 1,187.

2.3 Zrýchlenie v praxi

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), pričom paralelný kód bol implementovaný pomocou rozhrania OpenMP.

Testy boli vykonané na troch rôznych počítačových zostavách, všetky so systémom Windows 10. Testovacie stroje obsahujú moderné viacvláknové procesory so vzájomne odlišnými architektúrami a hlavne spôsobmi vykonávania paralelizovaných procesov. Stĺpce 2-4 v tabuľke predstavujú časy trvania testov základných aritmetických operácií.

Stĺpce 5 a 6 predstavujú časy behov sekvenčných implementácií de Boorovho a redukovaného algoritmu, pričom stĺpec 7 ukazuje pomer týchto časov. Ostávajúce stĺpce analogicky znázorňujú časy a pomer paralelných verzií. Všetky uvedené časové údaje sú v milisekundách.

CPU	Operácia			Sekvenčne			Paralelne		
	+	·	/	Plný	Red	F : R	Plný	Red	F : R
Core i5 4400	121	121	1087	128	101	1,27	55	56	0,98
Core i3 2350M	325	297	3151	252	202	1,25	142	145	0,98
FX-6300	133	136	529	180	146	1,23	78	79	0,99

Na každom z troch testovacích strojov sú rýchlosti sčítania a násobenia 64 bitových čísel s plávajúcou desatinnou čiarkou totožné. V prípade násobenia, respektíve sčítania a delenia vychádza pomer na procesoroch Intel Core i5 a i3 približne 9. Sekvenčná verzia redukovaného algoritmu je o 25% rýchlejšia ako pôvodný de Boorov postup čo súhlasí s odhadnutým zlepšením na konci kapitoly Zrýchlenie v príklade 2.2. Tam nám vyšlo 19% zrýchlenie, kde zbytok prisudzujem vplyvu pamäťových operácií, ktoré sme vtedy nebrali do úvahy.

V prípade procesora AMD FX si môžeme všimnúť anomáliu, kde nový algoritmus je výrazne rýchlejší než by mal byť podľa dosiahnutého pomeru rýchlosti násobenia a delenia. Moja terajšia hypotéza k tomuto javu je založená na pozorovaní, že AMD FX procesory majú výrazne pomalší pamäťový subsystém včítane systému rýchlych vyrovnávacích pamätí[11] (tzv. *cache*). Pamäťové operácie sú teda na AMD FX procesore výrazne drahšie ako na Intel Core. Redukovaný algoritmus má menšie priestorové nároky čo znamená

menej použitých pamäťových operácií. Teda na procesore AMD FX má síce zníženie počtu delení menší vplyv na efektivitu redukovaného algoritmu, ale to mohlo byť „vyvážené“ práve znížením počtu, na tomto CPU pomalých, pamäťových operácií.

Ďalej si môžeme všimnúť malé škálovanie výkonu paralelizovaného algoritmu. Prvou zmýšľanou príčinou môže byť fakt, že v plnom algoritme všetky parciálne derivácie podľa nejakej premennej vypočítame trojdiagonálnymi sústavami naraz v jednom cykle. V redukovanom ale cez trojdiagonálne sústavy vyrátame len polovicu deriváciu, pričom zvyšok dopočítame práve restandami. Toto rozdelenie výpočtu na dve subprocedúry teoreticky môže implikovať vysokú réžiu plánovača vlákien (tzv. *thread scheduler*). Táto príčina sa po prepracovaní implementácie ukázala bezpredmetná. Ďalšou príčinou môže byť fakt, testovanie prebieha na príliš malom počte uzlov (rádovo státisíce až milióny uzlov). Toto sa chystám prešetriť v priebehu februára.³

³Túto hypotézu vplyvu cache je potrebné ešte podrobnejšie otestovať, k čomu sa dostanem pravdepodobne v priebehu februára.

Kapitola 3

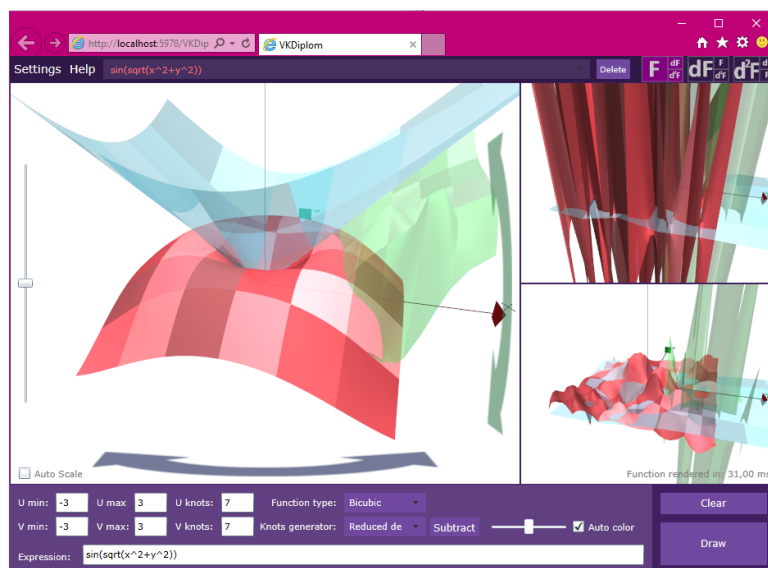
Implementácia a užívateľská príručka

Naším cieľom je vytvorenie aplikácií na vizuálne a výkonnostné porovnávanie splajnov, ktorých uzly sú počítané De Boorovým a naším algoritmom. Pôvodne sme vizuálnu časť implementovali v Microsoft Silverlight, čo je bezplatný nástroj na tvorbu webových aplikácií spustiteľných priamo vo webovom prehliadači. Samotný vývoj aplikácii môže prebiehať v ľubovoľnom programovacom jazyku bežiacom pod virtuálnym strojom Common Language Runtime. V našom prípade sme siahli po jazyku C#.



Silverlight je kompatibilný s väčšinou moderných webových prehliadačov a operačných systémov vrátane Microsoft Windows, Apple OS X a vďaka technológii Moonlight – open-source implementácii Silverlight-u aj na väčšine Linuxových distribúcií. Framework je možné použiť aj na vývoj off-line aplikácií v operačných systémoch Windows Phone, Windows 8 a Windows RT.

Poznámka 3.1 Naša aplikácia používa technológie, ktoré nie sú s Moonlight kompatibilné. V súčasnej dobe teda podporujeme iba MS Windows a Apple OS X. Prípadná portácia na GNU/Linux by musela byť vo forme off-line desktopovej aplikácie.



Obr. 3.6: Ukážka z aplikácie.

Pretože CLR znemožňuje korektne porovnávať procesorový čas a pamäťové nároky implementovaných algoritmov, rozhodli sme sa výkonnostný tester oddeliť do samostatnej aplikácie implementovanej v natívnom jazyku, konkrétne v jazyku C++.

```

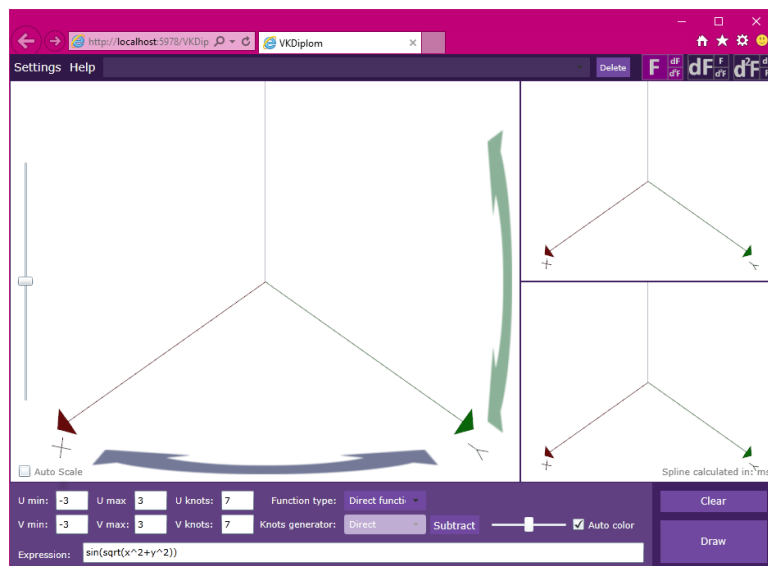
SplineKnots Benchmark 2
1: Multiplication vs division benchmark.
2: Spline curve benchmark.
3: Spline surface benchmark.
4: Spline surface benchmark (in parallel).
5: Compare Csaba T. vs. Vilo K. LU decomposition.
B: Disable/enable buffering in benchmarks.
Q: End program

```

Obr. 3.7: Výkonnostný tester.

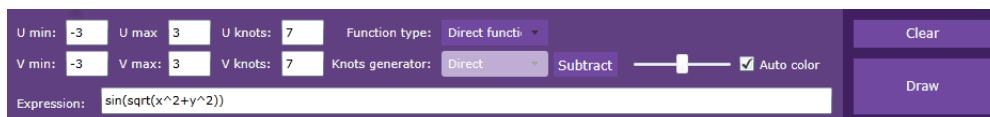
3.0.1 Vizualizácia

V tejto kapitole si ukážeme stručný návod na používanie a testovanie implementovaných aplikácií. Začnime aplikáciou na grafickú vizualizáciu.



Obr. 3.8: Aplikácia po spustení.

Okno aplikácie sa skladá z troch hlavných častí, ktoré si trochu netradične prejdeme zdola nahor. Na obrázku 3.9 vidíme panel, pomocou ktorého mô-

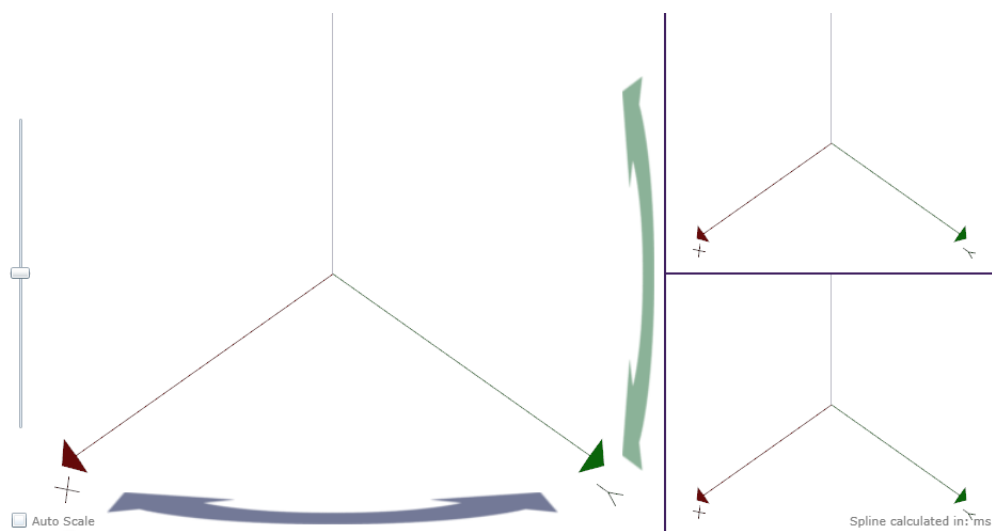


Obr. 3.9: Panel zadávania splajnu.

žeme na ľubovoľnom intervale a s ľubovoľným počtom uzlov (v rozumných medziach) interpolovať zapísanú funkciu dvoch premenných x a y . Popíšme si jednotlivé ovládacie prvky. Textové polia označené $U\ min$, $U\ max$ a $U\ knots$ ($V\ min$, $V\ max$ a $V\ knots$) značia interval a počty uzlov na osi x (osi y). Do spodného textového políčka označené *Expression* môžeme napísať takmer ľubovoľnú matematickú funkciu premenných x a y ktorú si želáme vykresliť. Vysúvateľný zoznam označený *Function type* umožňuje zvoliť typ interpolácie. Na výber máme ponúknuté tri možnosti akým spôsobom si želáme interpretovať matematický výraz v políčku *Expression*.

- *Direct function* vypočíta funkčné hodnoty potrebné pre grafickú vizualizáciu priamo z funkcie.
- *Bicubic* vypočíta z funkcie iba funkčné hodnoty v uzloch. Zvyšné potrebné body sú získané interpolačným bikubickým splajnom.
- *Biquartic* funguje analogicky ako predchádzajúca položka. Použije sa ale bikvartický splajn.

Vysúvateľný zoznam označený *Knots generator* nám, v prípade výberu vykreslenia funkcie splajnom, umožňuje zvoliť plný (*De Boor*) alebo redukovaný (*Reduced de Boor*) algoritmus. Tlačítko *Subtract* umožní vykresliť rozdiel medzi aktuálne zadanými hodnotami a nejakou už vykreslenou funkciou. Pomocou posuvníka označený *Auto color* môžeme namiesto automaticky vybranej farby vykresliť novú plochu vlastnou farbou. Nakoniec v pravej časti okna sa nachádzajú tlačidlá *Clear* a *Draw*, pričom prvé odstráni všetky vykreslené funkcie a druhé nám vypočíta a zobrazí funkciu podľa aktuálne zadaných hodnôt.



Obr. 3.10: Hlavné zobrazovacie okná.

Na obrázku 3.10 vidíme dominantnú časť aplikácie a síce zobrazovanie funkciových plôch. Obrazovka je rozdelená na tri časti, kde v najväčšia časť implicitne zobrazuje interpretovanú funkciu a zvyšné dve zobrazujú prvú, respektíve druhú zmiešanú deriváciu funkcie. V ľavej časti okna môžeme nájsť posuvník na škálovanie osi z . Na spodnej a pravej strane hlavnej časti okna sú tlačidlá reprezentované modrou horizontálnou, respektíve zelenou vertikálnou šípkou. Pri ich držaní myšou je možné rotovať obrazom v danej osi.

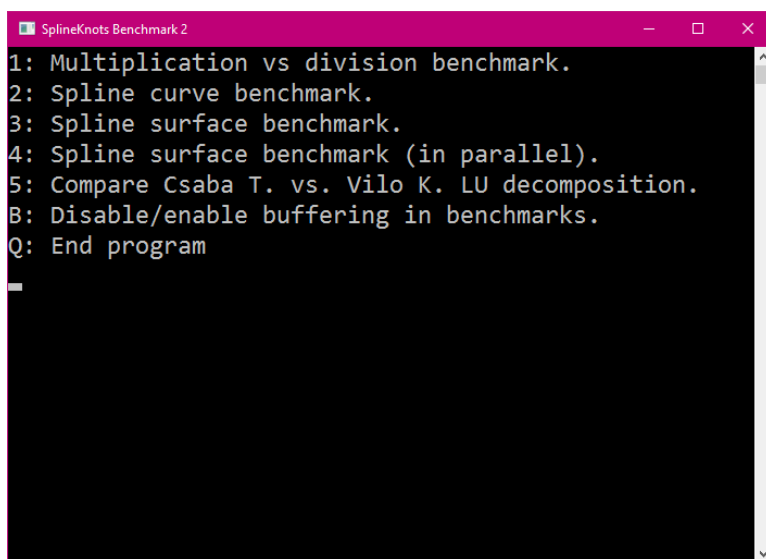


Obr. 3.11: Lišta nástrojov.

Položka *Settings* umožňuje prispôbiť grafické vykresľovanie splajnov. Tieto nastavenia je vhodné upraviť najmä pri starších počítačoch na dosiahnutie optimálnej plynulosti vykresľovania. V strednej časti panela máme k dispozícii vysúvateľný zoznam vykresľovaných plôch, ktorým ich môžeme zvýrazňovať. V pravej časti nájdeme tri tlačidlá, ktoré vycentrujú zobrazenie splajnu, prípadne jeho prvej alebo druhej zmiešanej derivácie.

3.0.2 Výkonnostný tester

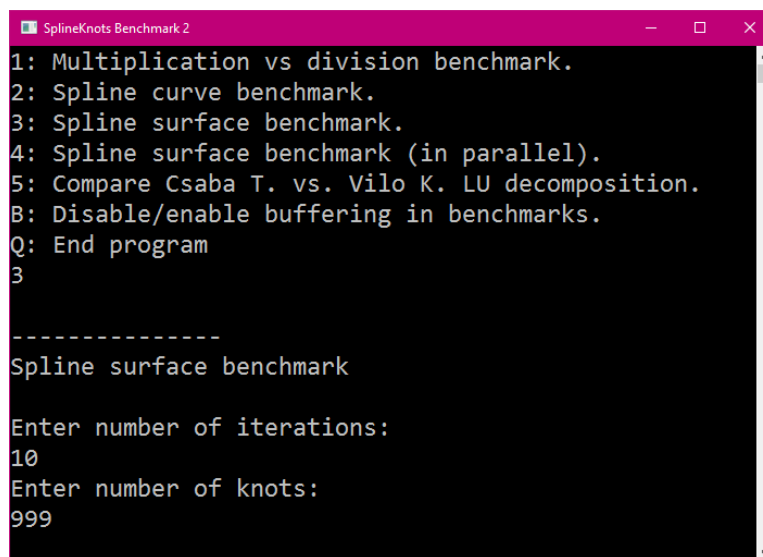
Teraz si popíšme druhú implementovanú aplikáciu zameranú nie na vykresľovanie, ale na ukážku či skutočne je redukovaný algoritmus rýchlejší. Na



Obr. 3.12: Výkonnostný tester.

obrázku 3.12 môžeme vidieť úvodné okno aplikácie, ktoré obsahuje sedem položiek. Ich význam si teraz vyjasníme.

1. *Multiplication vs division benchmark* spustí niekoľko variant testovania pomerov rýchlosti operácií sčítania, násobenia a delenia.
2. *Spline curve benchmark* pomerá časy výpočtov derivácií pre krivkový splajn interpolujúci preddefinovanú funkciu plným a redukovaným al-

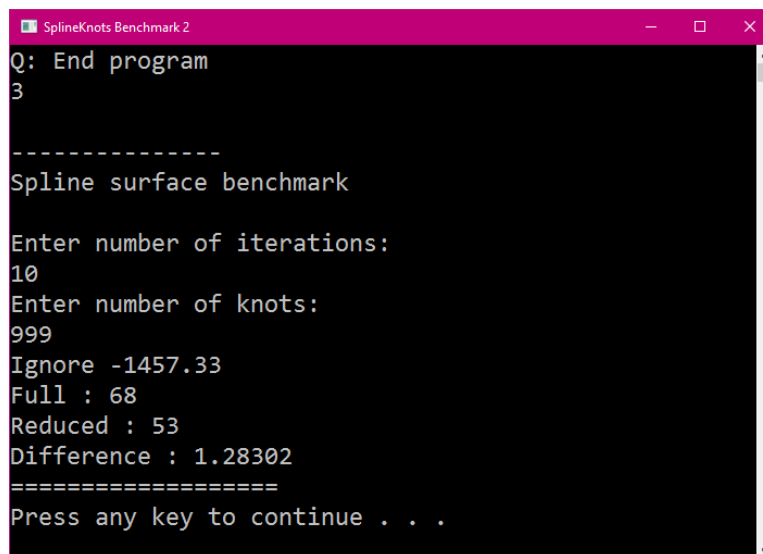


```
SplineKnots Benchmark 2
1: Multiplication vs division benchmark.
2: Spline curve benchmark.
3: Spline surface benchmark.
4: Spline surface benchmark (in parallel).
5: Compare Csaba T. vs. Vilo K. LU decomposition.
B: Disable/enable buffering in benchmarks.
Q: End program
3

-----
Spline surface benchmark

Enter number of iterations:
10
Enter number of knots:
999
```

Obr. 3.13: Zadanie údajov pre testovanie.



```
SplineKnots Benchmark 2
Q: End program
3

-----
Spline surface benchmark

Enter number of iterations:
10
Enter number of knots:
999
Ignore -1457.33
Full : 68
Reduced : 53
Difference : 1.28302
=====
Press any key to continue . . .
```

Obr. 3.14: Výsledky testu.

goritmom. Pred spustením aplikácie vyžaduje postupne zadanie dvoj údajov. Prvý udáva počet opakovaní testu pre každý algoritmus. Výsledný pomer vznikne vydelením aritmetických priemerov nameraných časov pre každý algoritmus. Čím je zadané číslo väčšie, tým je meranie menej náchylné na odchýlky. Odporúča sa zadať dvojcifernú hodnotu

a potvrdiť klávesou **Enter**. Druhé zadané číslo znamená počet uzlov splajnu. Odporúča sa zadať rádovo 10^5 až 10^6 uzlov. Menší počet dokáže väčšina moderných počítačov vypočítať príliš rýchlo aby ten čas dokázala aplikácia rozumne zmerať.

3. *Spline surface benchmark* analogicky ako predchádzajúci test testuje pomer časov výpočtu plného a redukovaného algoritmu pre plochy. Druhý zadany údaj tentoraz znamená počet uzlov pre jeden rozmer. Celkový počet uzlov je teda štvorec zadanej hodnoty. Pre väčšinu počítačov sa odporúča zadať rádovo 10^2 až 10^3 uzlov.
4. *Spline surface benchmark (in parallel)* paralelná verzia testu pre plochy využívajúca na výpočet jednej matice uzlov všetky dostupné logické procesory.
- B. *Disable/enable buffering* prepne medzi naivnou a optimalizovanou variantou algoritmu pre počítanie LU dekompozície v testoch 2 a 3. Test 4 toto nastavenie neovplyvňuje z dôvodu značných pamäťových nárokov naivnej verzie LU dekompozície.
- Q. *End program* ako názov napovedá, pri stlačení tlačidla **Q** dôjde k vypnutiu aplikácie.

Záver

Zoznam použitej literatúry

- [1] David Salomon, Curves and Surfaces for Computer Graphics, Springer, 2006
- [2] I. Szabó, L. Miño, C. Török, Biquartic polynomials in bicubicspline construction, PF UPJŠ, 2014
- [3] L. Miño, Parametrické modelovanie dát komplexnej štruktúry, PF UPJŠ, 2014
- [4] C. de Boor, Bicubic spline interpolation, Journal of Mathematics and Physics, 41(3),1962, 212-218.
- [5] J. Albahari, B.Albahari, C# 5.0 in a Nutshell, O'Reilly, 2012
- [6] E. Süli, D. Mayers, An Introduction to Numerical Analysis, Cambridge University Press, 2003, ISBN 0 521 00794 1
- [7] <https://github.com/vildibald/VKDiplom-master>, repozitár so zdrojovými kódmi k aplikáciám.
- [8] <http://www.anandtech.com/show/9483/intel-skylake-review-6700k-6600k-ddr4-ddr3-ipc-6th-generation/9>
- [9] <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf>
- [10] <http://www.lighterra.com/papers/modernmicroprocessors/>
- [11] <http://techreport.com/review/23750/amd-fx-8350-processor-reviewed/3>
- [12] TODO: Doplň články z internetu