Zrýchlenie výpočtu splajnových povrchov pomocou redukovaného algoritmu

Zoltán Szoplák

3Ib, 2017-2018

Abstrakt

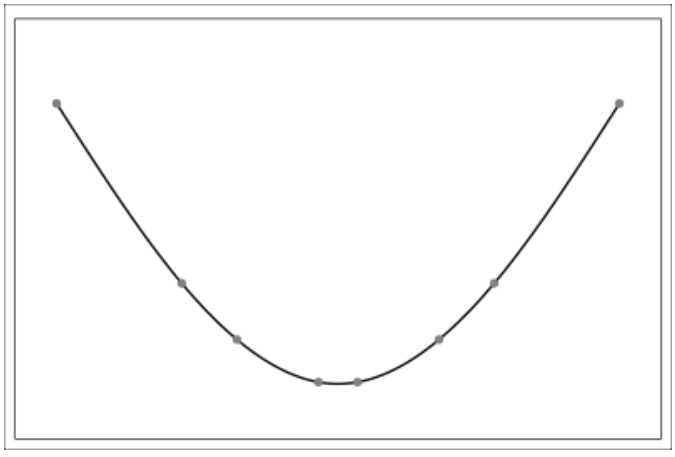
Splajny a splajnové povrchy sa v informatike mnohokrát používajú na modelovanie či aproximáciu dát a taktiež sú všadeprítomné v počítačovej grafike, hlavne v CAD systémoch. Z tohoto dôvodu sa usilujeme nájsť algoritmus, ktorý by dokázal riešiť trojdiagonálne sústavy rovníc potrebných na ich interpoláciu čo najrýchlejšie. Naše riešenie spočíva vo využití viacerých algoritmov, vrátane nedávno navrhnutej metódy, ktorá umožňuje explicitne vypočítať polovicu neznámych koefficientov.

Kľúčové slová: splajny, povrchy, interpolácia, trojdiagonálna sústava

Úvod

1.1 Teória splajnov a povrchov

Splajny sú špeciálne typy funkcií, ktoré sú definované polynomiálmi jednotlivo po častiach. Môžu byť jeden- alebo viacrozmerné.



Obrázok 1: Splajnová krivka

V informatike sú často využívané pri modelovaní dát, hlavne v oblasti počítačovej grafiky ako reprezentácia kriviek kvôli jednoduchosti ich konštrukcie, presnosti a rýchlosti ich vyhodnocovania, ako aj schopnosti vytvárania komplexnejších útvarov pomocou interaktívnych nástrojov. V našej práci sa budeme hlavne zaoberať splajnovými povrchmi, teda trojrozmernými splajnami.

Interpolácia je proces, ktorého cieľom je odhadnúť dáta na určitom intervale na základe už existujúcich dát. Existuje viacero metód interpolácie, jednou z najpoužívanejších je splajnová interpolácia, ktorá oproti polynomiálnej interpolácii produkuje malé chyby aj pri používaní polynómov nízkeho rádu, ktoré významne znižujú dobu výpočtu a taktiež sa vyhnú problému oscilácie nastávajúcej pri použití polynómov vyššieho rádu.

1.2 Splajnová interpolácia

Splajnová interpolácia pozostáva z rozdelenia danej funkcie na intervaly, kde hodnoty krajných bodov sú vopred známe, a nájdenie polynómov korešpondujúcich s jednotlivými intervalmi pre každý z nich. Našim cieľom je teda interpolovať splajnový povrch, o ktorom platí niekoľko pravidiel. Z dôvodu zachovania hladkosti funkcie sa požaduje, aby sa hodnota derivácie každého polynómu v krajnom bode zhodovala s hodnotou derivácie polynómu v susednom intervale v rovnakom krajnom bode. Úroveň hladkosti (kontinuity) hovorí o tom, koľkokrát je možné funkcie differencovať, aby sa spomínaný vzťah zachoval. Budeme pracovať s bikubickým povrchom, teda povrchom s úrovňou hladkosti 2, ktorý je navyše uniformný, čo znamená že deliace body sú od seba ekvidištančne položené. Uvažujme uniformnú mriežku definovanú deliacimi bodmi

$$[u_0, u_1, \ldots, u_{I-1}] \times [v_0, v_1, \ldots, v_{J-1}], \tag{1}$$

kde

$$u_i = u_0 + ih_x, \quad i = 1, 2, \ldots, I-1, \quad I = 2m+1, m \in \mathbb{N},$$
$$v_j = v_0 + jh_y, \quad j = 1, 2, \ldots, J-1, \quad J = 2n+1, n \in \mathbb{N}.$$

a h_x, h_y predstavujú vzdialenosť dvoch kontrolných bodov na osi x resp. y. Splajnový povrch pozostáva zo štyroch komponentov:

hodnota(výška) v danom bode mriežky

$$z_{i,j}, \quad i = 0, 1, \ldots, I-1, \quad j = 0, 1, \ldots, J-1 \tag{2}$$

parciálne derivácie prvého rádu pozdlž osi x

$$d^x_{i,j}, \quad i = 0, I-1, \quad j = 0, 1, \ldots, J-1 \tag{3}$$

parciálne derivácie prvého rádu pozdlž osi y

$$d^y_{i,j}, \quad i = 0, 1, \ldots, I-1, \quad j = 0, J-1 \tag{4}$$

parciálne derivácie druhého rádu podľa x a y v krajných bodoch mriežky

$$d^{x,y}_{i,j}, \quad i = 0, I-1, \quad j = 0, J-1 \tag{5}$$

Úlohou je násť tieto štyri hodnoty pre každý bod mriežky(pod bodom mriežky rozumieme usporiadanú dvojicu deliacich bodov $[u_i,v_j]$). Nech $S(u_i,v_j)$ je funkčná hodnota v príslušnom bode mriežky, kde S získame použitím spomýnaných štyroch hodnôt.

$$\begin{aligned} S(u_i, v_j) &= z_{i,j}, & \frac{\partial S(u_i, v_j)}{\partial y} &= d^y_{i,j}, \\ \frac{\partial S(u_i, v_j)}{\partial x} &= d^x_{i,j}, & \frac{\partial^2 S(u_i, v_j)}{\partial x \partial y} &= d^{x,y}_{i,j}, \end{aligned} \tag{6}$$

Kedže hodnoty $z_{i,j}$ sú nám známe, potrebujeme vypočítať derivácie. Prislúchajúca sústava rovníc zapísaná do matice tvorí trojdiagonálnu sústavu.

1.3 Trojdiagonálne matice a sústavy

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}$$

Obrázok 2: Trojdiagonálna sústava

Trojdiagonálne sústavy rovníc v sebe obsahujú trojdiagonálne matice, ktoré sa vyznačujú tým, že iba prvky, ktoré sa nachádzajú na diagonále alebo riadok nad či pod ňou majú nenulové hodnoty. Zatiaľ čo štandardné systémy matíc sa väčšinou riešia Gaussovou eliminačnou metódou, čo je operácia so zložitosťou $O(n^3)$, pre trojdiagonálne systémy existujú algoritmy, ktoré ich riešia v lineárnom čase. Sú nimi napr. Thomasov algoritmus či LU dekompozícia. Avšak pri simuláciach a vykreslovaniach v reálnom čase môže byť aj sekvenčný algoritmus zložitosti O(n) časovo náročný. Väčšina metód zlepšenia dnes spočíva v zjednodušení, ako aj paralelizácii spomínaného výpočtu.

Riešenie pomocou redukovaného algoritmu

2.1 LU dekompozícia

LU dekompozícia je sekvenčný algoritmus na rozdelenie riešenia trojdiagonálnej sústavy na dve menšie podproblémy s lineárnou zložitosťou. Majme trojdiagonálnu maticu T

$$T = \begin{pmatrix} a_1 & b_1 & & & \\ c_1 & a_2 & b_2 & & \\ & c_2 & \ddots & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ & & & c_{n-1} & a_n \end{pmatrix}$$

Obrázok 3: Trojdiagonálna matica

LU dekompozícia alebo LU rozklad nám ju umožňuje rozdeliť na dve matice, L – Lower a U – Upper, pričom platí A=LU

$$L = \begin{pmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & l_3 & 1 & & \\ & & \ddots & \ddots & \\ & & & l_n & 1 \end{pmatrix}, \; U = \begin{pmatrix} u_1 & c_1 & & & \\ & u_2 & c_2 & & \\ & & \ddots & \ddots & \\ & & & u_{n-1} & c_{n-1} \\ & & & & u_n \end{pmatrix}$$

Obrázok 4: LU dekompozícia

Kde hodnoty $c_1....c_{n-1}$ poznáme, a hodnoty $u_1....u_n$ resp. $l_2....l_n$ sa vypočítajú nasledovne:

$$u_i = \begin{cases} b_0; & i = 1 \\ b_i - \frac{a_i c_{i-1}}{u_{i-1}}; & 2 \leq i \leq n \end{cases} \qquad (7)$$
$$l_i = \frac{a_i}{u_{i-1}}$$

Vezmime trojdiagonálnu sústavu z obrázka 2. Z nej máme Ax=d, po LU dekompozícii LUx=d. No tento problém sa dá rozdeliť na dva podproblémy: Ux=y a Ly=d. Tieto rovnice pozostávajú už len z dvojdiagonálnych matíc a dajú sa riešiť v lineárnom čase.

2.2 Paralelizácia

Na účely paralelizácie využijeme algoritmus ABM - Austin Berndt Moulton popísaný v [1]. Tento algoritmus má za úlohu rozdeliť trojdiagonálnu maticu na niekoľko menších matíc. Nech má naša trojdiagonálna matica n riadkov a majme p processorov, kde p<n. Označme j_k ako najmenší index riadku patriaci processoru s indexom k. Nech $j_1=1$, $j_{p+1=}n+1$ a $j_k < j_{k+1}$, kde k = 1, … ,p. Processor s indexom k disponuje n_k riadkami matice, kde $n_k= j_{k+1}-j_k$. Riadky matice označíme v_i , kde i=1…n. Majme trojdiagonálnu sústavu ktorá má prvky s indexami a_i pod diagonálou, prvky s indexami b_i na diagonále a prvky s indexami c_i nad diagonálou v i-tom riadku(Obrázok 2). Samotný algoritmus ABM môžeme rozdeliť do troch fáz:

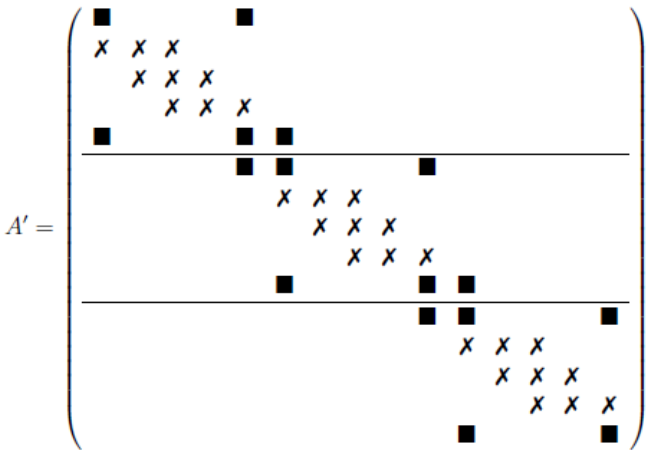
V prvej fáze každý processor pozmení prvý a posledný riadok, ktorý mu patrí. Hodnoty týchto riadkov si uložíme do premenných v_{lk} a v_{uk}, kde v_{lk} si uchováva hodnotu posledného riadku a v_{uk} si uchováva hodnotu

prvého riadku. Taktiež si uložíme pravú stranu našej sústavy do premenných r_{lk} (posledný riadok) a r_{uk} (prvý riadok). Hodnotu v_{lk} vypočítame nasledovne:

$$\begin{aligned} v_{l_k} &\leftarrow v_{j_k+1} \\ r_{l_k} &\leftarrow r_{j_k+1} \\ \text{Pre } i &= j_k+2, \ldots, j_k+N_k-1 \\ \alpha_i &\leftarrow \frac{a_i}{v_{l_k,i-1}} \\ v_{l_k} &\leftarrow v_i - \alpha_i v_{l_k} \\ r_{l_k} &\leftarrow r_i - \alpha_i r_{l_k} \end{aligned} \tag{8}$$

Tento výpočet prebieha paralelne pre každý processor. Hodnotu v_{uk} získame podobným spôsobom:

$$\begin{aligned} v_{u_k} &\leftarrow v_{j_k+n_k-2} \\ r_{u_k} &\leftarrow r_{j_k+n_k-2} \\ \text{Pre } i &= j_k+n_k-3, \ldots, 1 \\ \beta_i &\leftarrow \frac{c_i}{v_{u_k,i}} \\ v_{u_k} &\leftarrow v_i - \beta_i v_{u_k} \\ r_{u_k} &\leftarrow r_i - \beta_i r_{u_k} \end{aligned} \tag{9}$$



Obrázok 5: ABM

Na obrázku 5 môžeme vidieť ilustráciu úprav vykonaných v prvej fáze ABM algoritmu. Vodorovné čiary reprezentujú rozdelenie medzi processormi. Hodnoty reprezentované symbolmi ✗ ostanú po týchto úpravách nezmenené, čierne štvorce reprezentujú hodnoty prvých a posledných riadkov.

Z vlastností trojdiagonálnej matice je triviálne dokázateľné, že počas výpočtu vektory v_{lk} a v_{uk} nemajú nikdy viac ako 3 nenulové hodnoty. Tento fakt využijeme v druhej fáze algoritmu.

V druhej fáze získané vektory poskladáme do jednej sústavy rovníc:

$$\begin{pmatrix} v_{u_1} \\ v_{l_1} \\ \vdots \\ v_{u_{np}} \\ v_{l_{np}} \end{pmatrix} \begin{pmatrix} d_{j_1} \\ d_{j_2-1} \\ \vdots \\ d_{j_{np}} \\ d_{j_{np}+1-1} \end{pmatrix} = \begin{pmatrix} r_{u_1} \\ r_{l_1} \\ \vdots \\ r_{u_{np}} \\ r_{l_{np}} \end{pmatrix} \tag{10}$$

Takto sme vytvorili nové trojdiagonálne sústavy rovníc s menším počtom neznámych, ktoré vieme počítať pomocou sekvenčných algoritmov. Riešenia napokon odošleme spätne korešpondujúcim processorom.
V tretej fáze si každý processor dosadí hodnoty neznámych vypočítané v druhej fáze do svojej sústavy rovníc, čím získa vlástnú nezávislú trojdiagonálnu sústavu. Celý tento proces po vykonaní rozdelenia môže byť teda plne paralelizovaný bez potreby synchronizácie a kommunikácie medzi vláknami.

2.3 Carl de Boor algoritmus

Uvažujme uniformnú mriežku (1). Z (2)-(5) sú hodnoty z známe a hodnoty

$$\begin{aligned} &d^x_{i,j}, \quad i = 1, \ldots, I-2, \quad j = 0, \ldots, J-1, \\ &d^y_{i,j}, \quad i = 0, \ldots, I-1, \quad j = 1, \ldots, J-2, \\ &d^{x,y}_{i,j}, \quad i = 1, \ldots, I-2, \quad j = 0, \ldots, J-1, \\ &\text{a} \quad i = 0, \ldots, I-1, \quad j = 1, \ldots, J-2 \end{aligned} \tag{11}$$

sú jednoznačne určené nasledujúcimi 2I + J + 2 lineárnymi systémami pozostávajúcich v celku z 3IJ - 2I - 2J – 4 rovností.

$$d^x_{i+1,j} + 4d^x_{i,j} + d^x_{i-1,j} = \frac{3}{h_x}(z_{i+1,j} - z_{i-1,j}) \tag{12}$$

Pre všetky j=0,...,J-1 kde i=0,...,I-2

$$d^y_{i,j+1} + 4d^y_{i,j} + d^y_{i,j-1} = \frac{3}{h_y}(z_{i,j+1} - z_{i,j-1}) \tag{13}$$

Pre všetky i=0,...,I-1 kde j=1,...,J-2

$$d^{x,y}_{i+1,j} + 4d^{x,y}_{i,j} + d^{x,y}_{i-1,j} = \frac{3}{h_x}(d^y_{i+1,j} - d^y_{i-1,j}) \tag{14}$$

Pre j=0 a J-1 kde i=0,...,I-1

$$d^{x,y}_{i,j+1} + 4d^{x,y}_{i,j} + d^{x,y}_{i,j-1} = \frac{3}{h_y}(d^x_{i,j+1} - d^x_{i,j-1}) \tag{15}$$

Pre všetky i=0,...,I-1 a j=1,...,J-2

2.4 Redukovaný algoritmus

Z výskumu doc. Töröka sa však ukázalo, že ak sa jedná o uniformné splajny alebo povrchy, tak týmto spôsobom stačí vypočítať iba každú párnu deriváciu a hodnotu zvyšných premenných zistíme triviálne. Hodnoty (10) sa teda vypočítajú použitím nasledujúcich $\frac{5IJ-I-J-23}{4}$ rovníc v $\frac{3IJ+2I+5}{2}$ lineárnych systémoch a $\frac{7IJ-7I-7J+7}{4}$ formulách, celkovo 3IJ - 2I - 2J – 4 rovností.

Krok 1a (parciálne derivácie podľa x pri párnom indexe i):

$$d^x_{i+2,j} - 14d^x_{i,j} + d^x_{i-2,j} = \\ = \frac{3}{h_x}(z_{i+2,j} - z_{i-2,j}) - \frac{12}{h_x}(z_{i+1,j} - z_{i-1,j}) \tag{16}$$

Pre j=0,...,J-1 kde i=2,4..I-3

Krok 1b (parciálne derivácie podľa x pri nepárnom indexe i):

$$d^x_{i,j} = \frac{3}{4h_x}(z_{i+1,j} - z_{i-1,j}) - \frac{1}{4}(d^x_{i+1,j} + d^x_{i-1,j}) \tag{17}$$

Pre j=0,...,J-2, i=1,3...,I-2

Krok 2a (parciálne derivácie podľa y pri párnom indexe j):

$$d^y_{i,j+2} - 14d^y_{i,j} + d^y_{i,j-2} = \\ = \frac{3}{h_y}(z_{i,j+2} - z_{i,j-2} - \frac{12}{h_y}(z_{i,j+1} - z_{i,j-1}) \tag{18}$$

Pre i=0,1,...,I-1 kde j=2,4..J-3

Krok 2b (parciálne derivácie podľa y pri nepárnom indexe j):

$$d^y_{i,j} = \frac{3}{4h_y}(z_{i,j+1} - z_{i,j-1}) - \frac{1}{4}(d^y_{i,j+1} + d^y_{i,j-1}) \tag{19}$$

Pre j=1,3...,J-2 a i=1,3,...,I-2

Krok 3a (parciálne derivácie podľa x,y v krajných bodoch osi y):

$$d^{x,y}_{i+1,j} + 4d^{x,y}_{i,j} + d^{x,y}_{i-1,j} = \frac{3}{h_x}(d^y_{i+1,j} - d^y_{i-1,j}) \tag{20}$$

Pre j=0,J-1 kde i=1,...,J-2

Krok 3b (parciálne derivácie podľa x,y v krajných bodoch osi x):

$$d^{x,y}_{0,j+1} + 4d^{x,y}_{0,j} + d^{x,y}_{0,j-1} = \frac{3}{h_y}(d^x_{0,j+1} - d^x_{0,j-1}) \tag{21}$$

Pre i=0,I-1 a j=0,...,J-2

Krok 3c (parciálne derivácie podľa x,y vo vnútorných bodoch pri párnych hodnotách i,j):

$$
\begin{aligned}
& d^{x,y}_{i,j+2} - 14d^{x,y}_{i,j} + d^{x,y}_{i,j-2} = \\
& = \frac{1}{7}(d^{x,y}_{i-2,j+2} + d^{x,y}_{i-2,j-2}) - 2d^{x,y}_{i-2,j} + \\
& + \frac{3}{7h_x}(d^{y}_{i-2,j+2} + d^{y}_{i-2,j-2}) + \frac{3}{7h_y}(-d^{x}_{i-2,j+2} + d^{x}_{i-2,j-2}) + \\
& + \frac{9}{7h_x}(d^{y}_{i,j+2} + d^{y}_{i,j-2}) + \frac{9}{7h_xh_y}(-z_{i-2,j+2} + z_{i-2,j-2}) + \\
& + \frac{12}{7h_x}(-d^{y}_{i-1,j+2} - d^{y}_{i-1,j-2}) + \frac{12}{7h_y}(d^{x}_{i-2,j+1} - d^{x}_{i-2,j-1}) + \\
& + \frac{3}{h_y}(d^{x}_{i,j+2} - d^{x}_{i,j-2}) + \frac{27}{7h_xh_y}(-z_{i,j+2} + z_{i,j-2}) + \\
& + \frac{36}{7h_xh_y}(z_{i-1,j+2} - z_{i-1,j-2} + z_{i-2,j+1} - z_{i-2,j-1}) - \\
& - \frac{6}{h_x}d^{y}_{i-2,j} + \frac{12}{h_y}(d^{x}_{i,j+1} + d^{x}_{i,j-1}) + \frac{108}{7h_xh_y}(z_{i,j+1} - z_{i,j-1}) - \\
& - \frac{18}{h_x}d^{y}_{i,j} + \frac{144}{7h_xh_y}(-z_{i-1,j+1} + z_{i-1,j-1}) + \frac{24}{h_x}d^{y}_{i-1,j},
\end{aligned}
\tag{22}
$$

kde i=2,4,6,....I-3 a j=4,6,…,J-5

Krok 3d (parciálne derivácie podľa x,y vo vnútorných bodoch pri nepárnych hodnotách i,j):

$$
\begin{aligned}
d^{x,y}_{i,j} = & \frac{1}{16}(d^{x,y}_{i+1,j+1} + d^{x,y}_{i+1,j-1} + d^{x,y}_{i-1,j+1} + d^{x,y}_{i-1,j-1}) - \\
& - \frac{3}{16h_y}(d^{x}_{i+1,j+1} - d^{x}_{i+1,j-1} + d^{x}_{i-1,j+1} - d^{x}_{i-1,j-1}) - \\
& - \frac{3}{16h_x}(d^{y}_{i+1,j+1} + d^{y}_{i+1,j-1} - d^{y}_{i-1,j+1} - d^{y}_{i-1,j-1}) + \\
& + \frac{9}{16h_xh_y}(z_{i+1,j+1} - z_{i+1,j-1} - z_{i-1,j+1} + z_{i-1,j-1}),
\end{aligned}
\tag{23}
$$

pre i=1,3,..I-2 a j=1,3...J-2

Krok 3e (parciálne derivácie podľa x,y vo vnútorných bodoch pri nepárnych hodnotách i a párnych hodnotách j):

$$
d^{x,y}_{i,j} = \frac{3}{4h_y}(d^{x}_{i,j+1} - d^{x}_{i,j-1}) - \frac{1}{4}(d^{x,y}_{i,j+1} + d^{x,y}_{i,j-1}),
\tag{24}
$$

pre i=1,3,..I-2 a j=2,4...J-3

Krok 3f (parciálne derivácie podľa x,y vo vnútorných bodoch pri párnych hodnotách i a nepárnych hodnotách j):

$$d^{x,y}_{i,j} = \frac{3}{4h_y}(d^x_{i,j+1} - d^x_{i,j-1}) - \frac{1}{4}(d^{x,y}_{i,j+1} + d^{x,y}_{i,j-1}), \tag{25}$$

pre i=2,4,..I-3 a j=1,3,...,J-2

2.5 Redukovaný algoritmus s využitím Carl de Boor algoritmu:

Po úprave redukovaného algoritmu a jeho spojením s algoritmom Carl de Boor dostaneme nasledujúcu sústavu rovníc:

Krok 1a (parciálne derivácie podľa x pri párnom indexe i):

$$\begin{aligned} &d^x_{i+2,j} - 14d^x_{i,j} + d^x_{i-2,j} = \\ &= \frac{3}{h_x}(z_{i+2,j} - z_{i-2,j}) - \frac{12}{h_x}(z_{i+1,j} - z_{i-1,j}) \end{aligned} \tag{26}$$

pre j=0,1...,J-1 kde i=2,4..I-3

Krok 1b (parciálne derivácie podľa x pri nepárnom indexe i):

$$d^x_{i,j} = \frac{3}{4h_x}(z_{i+1,j} - z_{i-1,j}) - \frac{1}{4}(d^x_{i+1,j} + d^x_{i-1,j}) \tag{27}$$

pre j=1,3,...,J-2, i=1,3...,I-2

Krok 2a (parciálne derivácie podľa y pri párnom indexe j):

$$\begin{aligned} &d^y_{i,j+2} - 14d^y_{i,j} + d^y_{i,j-2} = \\ &= \frac{3}{h_y}(z_{i,j+2} - z_{i,j-2} - \frac{12}{h_y}(z_{i,j+1} - z_{i,j-1}) \end{aligned} \tag{28}$$

pre i=0,1,...,I-1 kde j=2,4..J-3

Krok 2b (parciálne derivácie podľa y pri nepárnom indexe j):

$$d^y_{i,j} = \frac{3}{4h_y}(z_{i,j+1} - z_{i,j-1}) - \frac{1}{4}(d^y_{i,j+1} + d^y_{i,j-1}) \tag{29}$$

pre j=1,3...,J-2 a i=1,3,...,I-2

Krok 3a (parciálne derivácie podľa x,y pri párnom indexe i):

$$\begin{aligned} &d^{x,y}_{i+2,j} - 14d^{x,y}_{i,j} + d^{x,y}_{i-2,j} = \\ &= \frac{3}{h_x}(d^y_{i+2,j} - d^y_{i-2,j}) - \frac{12}{h_x}(d^y_{i+1,j} - d^y_{i-1,j}) \end{aligned} \tag{30}$$

pre j=0,J-1 kde i=2,4..I-3

Krok 3b (parciálne derivácie podľa x,y pri nepárnom indexe i):

$$d_{i,j}^{x,y} = \frac{3}{4h_x}(d_{i+1,j}^{x} - d_{i-1,j}^{x}) - \frac{1}{4}(d_{i+1,j}^{x,y} + d_{i-1,j}^{x,y}), \tag{31}$$

pre j=0,J a i=0,...,I-2

Krok 4a (parciálne derivácie podľa x,y pri párnom indexe j):

$$\begin{aligned} & d_{i,j+2}^{x,y} - 14d_{i,j}^{x,y} + d_{i,j-2}^{x,y} = \\ & = \frac{3}{h_y}(d_{i,j+2}^{y} - d_{i,j-2}^{y}) - \frac{12}{h_y}(d_{i,j+1}^{y} - d_{i,j-1}^{y}) \end{aligned} \tag{32}$$

pre i=0,..,I-1 kde j=2,4..J-3

Krok 4a (parciálne derivácie podľa x,y pri párnom indexe j):

$$d_{i,j}^{x,y} = \frac{3}{4h_y}(d_{i,j+1}^{y} - d_{i,j-1}^{y}) - \frac{1}{4}(d_{i,j+1}^{x,y} + d_{i,j-1}^{x,y}) \tag{33}$$

pre i=1,3..,I-2 kde j=1,3,...,J-2

2.6 Spojenie algoritmov

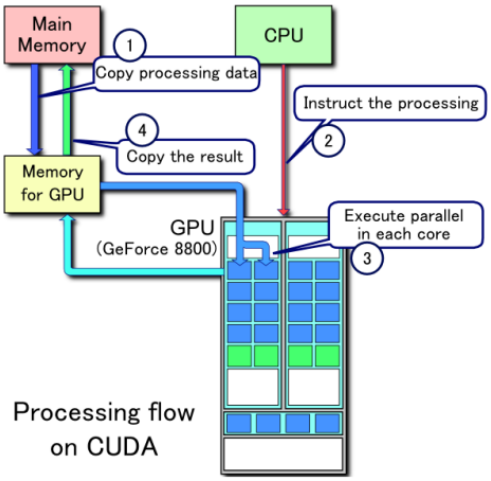
Pri použití finálnej formy redukovaného algoritmu pre bikubický uniformný splajnový povrch dokážeme zredukovať našu sústavu rovníc o takmer polovičný počet neznámych, ktoré sa potom počítajú pomocou algoritmu ABM, kde sa jednotlivé menšie sústavy paralelne počítajú cez sériový algoritmus LU rozklad. Pri párnych hodnotách sa jedná a triválny výpočet rovníc, ktoré sú absolútne nezávislé od seba a teda môžu byť plne paralelizovateľné. Pre implementáciu daného algoritmu sme zvolili programovací jazyk C++ a vývojové prostredie CUDA.

Implementácia

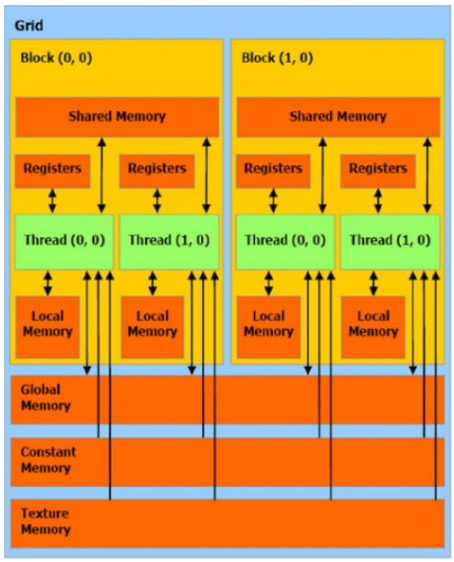
3.1 Nvidia CUDA

CUDA (Compute Unified Device Architecture) je API od firmy Nvidia, ktorá umožňuje priamu komunikáciu s GPU. Táto platforma je určená na paralelizáciu výpočtov, ako aj manažment pamäte a podporuje programovacie jazyky ako C, C++ a Fortran. Oproti predošlým API ako sú Direct3D a OpenGL je toto vývojárske prostredie pohodlnejšie na použitie a nevyžaduje si hlboké znalosti grafického programovania. V grafickom vývoji sa GPU používa hlavne na účely renderovania a výpočtov fyziky(detekcia a simulácia kolízií, deformácia mäkkých telies, simulácia pohybu viacero prepojených objektov) a iných simulácií(napr. effektov ako oheň, dym, simulácie kvapalín, distorcií pri použití displacement máp). CUDA podporuje softvéry,

ktoré sa zaoberajú takýmito simuláciami, ako napríklad Nvidia PhysX alebo Bullet, no taktiež sa využíva na zrýchlenie výpočtov, ktoré nie sú grafického charakteru, ako napríklad v oblasti cryptografie alebo výpočtovej biológie.



Obrázok 6: Priebeh procesu v CUDA



Obrázok 7: Štruktúra CUDA

Proces spustení v CUDA, znázornení na obrázku 6 prebieha nasledovne: Najprv sa prekopírujú údaje potrebné pre proces z hlavnej pamäte počítača do pamäte grafickej karty, kernel z CPU inicializuje proces na GPU, ten sa napokon paralelne vykoná a výsledky sa skopírujú spätne z pamäte grafickej karty do hlavnej pamäte počítača. Pre paralelné vykonávanie kernelu je možné si nakonfigurovať štruktúru vlákien a pamäte(Obrázok 7). V každom GPU je isté množstvo multiprocessorov, ktoré disponujú naďalej 32-192 jadier na ktorých môžu bežať vlákna. Tieto vlákna môžeme zoskupiť do tzv. blokov, ktoré si po zadaní ich počtu automaticky rozdelia systémové prostriedky grafickej karty medzi sebou. Tieto bloky majú vlastné registre a rôzne typy pamäte. Globálna pamäť je dostupná každému vláknu avšak disponuje vysokou latenciou. Hoci zabudované kešovanie tento problém do istej miery rieši, každý blok má taktiež zdielanú pamäť, ktorá je prístupná iba vláknam v rovnakom bloku, avšak má omnoho nižšiu latenciu ako globálna pamäť. Pre prípady kedy vlákna vôbec nepotrebujú medzi sebou komunikovať je vyhradená miestna pamäť. K pamäti sa dá taktiež priamo pristupovať cez ukazovateľe.

Záver

Je jednoznačné, že tento algoritmus prináša významné zlepšenie čo sa týka výpočtového času. Finálne výsledky budú čoskoro publikované.

Poďakovanie

Chcel by som sa poďakovať môjmu vedúcemu práce, doc. RNDr. Csabovi Törökovi za návrh tejto práce, extenzívnemu výskumu v tejto oblasti, bez ktorej by táto práca nemohla vzniknúť a za neoceniteľnú pomoc pri jej zhotovení a konzultantovi Mgr. Viliamovi Kačalovi za nesmiernu pomoc a podporu vo forme pomocných zdrojových kódov. Taktiež veľké poďakovanie patrí Mgr. Martinovi Jadlošovi, koho diplomovú prácu táto práca rozširuje.

Literatúra

[1] Austin, T.M, Berndt, M. Moulton, J.D.: A Memory E_cient Parallel Tridiagonal Solver. PreprintLA-VR-03-4149, 2004

[2] de Boor C.: Bicubic Spline Interpolation, Journal of Mathematics and Physics, 41(3), 1962, 212-218

[3] Cs. TÖRÖK: Speedup of interpolating spline construction, to appear.