MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Implementace difúzních filtrů na GPU

DIPLOMOVÁ PRÁCE

Lukáš Skovajsa

Brno, jaro 2011

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Pavel Matula, Ph.D., Mgr. Pavel Karas

Shrnutí

Tato diplomová práce se zabývá urychlením výpočtu difúzních filtrů pomocí GPU. V první kapitole je popsaná spojitá lineární, izotropní a anizotropní difúze. V druhé kapitole jsou popsaná diskretizační schémata, která vedou k AOS. V třetí kapitole je popsaná architektura GPU CUDA, pro kterou jsou algoritmy implementované. Hlavní výpočetní zátěž výpočtu difúze pomocí AOS je v řešení tridiagonálního systému lineárních rovnic. Ve čtvrté kapitole jsou popsané tři algoritmy, které tento systém řeší. V páté kapitole jsou popsané implementace těchto algoritmů pro řešení 2–D izotropní difúze. Po porovnání vlastností je vítězný algoritmus implementovaný i pro řešení lineární a anizotropní 2–D difúze. V šesté kapitole je popsané řešení 3–D lineární, izotropní a anizotropní difúze pomocí implementací z páté kapitoly.

Klíčová slova

zpracování obrazu, difúzní filtry, šum, AOS, CUDA, paralelní algoritmy, tridiagonální systém lineárních rovnic

Obsah

1 **Úvod** . . . 3
1.1 *Difúzní rovnice* . . . 4
1.2 *Konvoluce a difúze* . . . 4
1.3 *Spojitá izotropní difúze* . . . 5
1.4 *Spojitá anizotropní difúze* . . . 7
1.4.1 Anizotropní difúze pro vylepšení hran . . . 7
1.4.2 Anizotropní difúze pro zvýraznění koherence . . . 9
2 **Diskretizace** . . . 11
2.1 *Explicitní schéma* . . . 11
2.1.1 Vícerozměrný případ . . . 12
2.2 *Částečně implicitní schéma* . . . 13
2.2.1 Vícerozměrný případ . . . 13
2.3 *AOS* . . . 13
2.4 *Rozklad difúzního tenzoru* . . . 14
2.4.1 Trojrozměrný případ . . . 15
3 **CUDA** . . . 20
3.1 *Korespondence fyzických zařízení s pojmy v modelu* . . . 20
3.2 *Výkon* . . . 21
4 **Tridiagonální systém lineárních rovnic** . . . 22
4.1 *Thomasův algoritmus* . . . 22
4.2 *Dvousměrná Gaussova eliminace* . . . 23
4.3 *Paralelní cyklická redukce* . . . 25
4.4 *Dělící algoritmus* . . . 25
5 **CUDA implementace 2–D difúze** . . . 28
5.1 *Izotropní difúze* . . . 29
5.1.1 Thomasův algoritmus . . . 29
5.1.2 Dvousměrná Gaussova eliminace . . . 30
5.1.3 Dělící algoritmus . . . 34
5.1.4 Porovnání implementací . . . 35
5.2 *Lineární difúze* . . . 35
5.3 *Anizotropní difúze* . . . 36
5.3.1 Difúze na zlepšení hran . . . 37
5.3.2 Difúze na zlepšení koherence . . . 38
6 **CUDA implementace 3–D difúze** . . . 42
6.1 *Lineární 3–D difúze* . . . 42
6.2 *Izotropní 3–D difúze* . . . 42
6.3 *Anizotropní 3–D difúze* . . . 44
7 **Závěr** . . . 48
A **Vzhled GUI** . . . 49

B **Experimenty** 50
C **Dodatky** 54

Kapitola 1

Úvod

Obor zpracování obrazu je z určitého pohledu inverzní k oboru počítačové grafiky v tom smyslu, že v počítačové grafice jsou známé všechny vlastnosti objektů ve scéně a hledá se projekce těchto objektů do obrazové roviny. V oboru zpracování obrazu se naopak tyto vlastnosti zjišťují ze vstupních dat (např. obrázku). Reálná data obsahují ale kromě zjišťované vlastnosti i spoustu dalších nepotřebných vlastností.

Před vlastním zpracováním obrazu je často potřeba vstupní obraz zjednodušit, aby zanikly vlastnosti, které nejsou potřebné. Nejběžnější vlastnost, která není potřebná, ale je dokonce nežádoucí, je šum. Jedna z nejednodušších operací, která se dá použít na odstranění šumu, je konvoluce s Gaussovým filtrem. Hlavní nevýhoda této operace je, že výstupní obraz bude mít rozmazané hrany, které jsou důležitou vlastností obrazu a jejich možné posunutí není žádoucí.

V diplomové práci je popsaná metoda, kdy se na konvoluci s Gaussovým filtrem dívá jako na rovnici šíření tepla v materiálu, která je při nastavení koeficientu propustnosti tepla ve všech bodech jedné ekvivalentní s konvolucí s Gaussovým filtrem. Zobecnění této parciální diferenciální rovnice dokáže zohlednit hrany na obrázku, které v této fyzikální analogii s šířením tepla odpovídají izolujícímu materiálu, který má koeficient propustnosti menší než jedna. V oboru zpracování obrazu se na téhle rovnici neříká rovnice šíření tepla, ale difúzní rovnice. Tento název se mohl zažít proto, že při malování vodovými barvami dochází k difúzi barev. V tomhle příkladu koeficient propustnosti, difúzní koeficient, ovlivňuje vlhkost podkladu.

Protože žádné analytické řešeni pro zobecněnou difúzní rovnici není známé, je potřeba použít nějakou numerickou metodu. Nejednoduší metoda, která by se dala použít, je Eulerovo dopředné schéma. Weickert [1] ukázal, že toto schéma vyžaduje velice malý časový krok, takže k vyřešení je potřeba mnoho iterací. Weickert [1] také navrhl jiné schéma, které netrpí tímto problémem. Toto schéma se v anglickém originálu jmenuje *adaptive operator spliting* a v dalším textu se na něho bude odkazovat jako na AOS. Hlavní výpočetní zátěž při téhle metodě spočívá v řešení tridiagonálního systému lineárních rovnic.

Jádro diplomové práce je nalezení vhodného algoritmu pro řešení souboru tridiagonálních systémů lineárních rovnic, který dokáže využít potenciálu grafických karet, které podporují obecné výpočty. Konkrétní implementace je provedená pro grafické karty Nvidia, které podporují technologii CUDA. Programovací model, který odpovídá CUDA, je v diplomové práci také popsaný. Druhý problém, který tato diplomová práce řeší, je najít způsob, jak použít AOS k řešení anizotropní difúze na 3–D datech.

1.1 Difúzní rovnice

Difúzní rovnice se skládá ze dvou vztahů. První vyjadřuje, že rozdíl hustot dvou prostředí ∇u [1] vytváří tok, který se tento rozdíl snaží vyrovnat. Tomuto vztahu se říká Fickův zákon.

$$\phi = -D\nabla u \tag{1.1}$$

Weickert [2] podle typu D rozlišujeme tři druhy difúze. Pokud $D = 1$ difúze se nazývá *lineární* a je ekvivalentní s konvolucí s Gaussovým filtrem a výsledný tok ϕ je rovnoběžný s gradientem ∇u. Difúze se nazývá *nelineární izotropní*, když D je skalární funkce g. Nejčastější typy g jsou popsané dále v textu. V tomto případě bude výsledný tok ϕ také rovnoběžný s gradientem ∇u. Poslední případ nastane, když D je tenzor. Tento typ difúze se nazývá *nelineární anizotropní* a výsledný tok ϕ nemusí být obecně rovnoběžný s gradientem ∇u.

Požadavek na to, aby se při toku neměnila celková energie systému, vyjadřuje vztah[2][3]

$$\partial_t u \ = -div\phi. \tag{1.2}$$

Složením těchto dvou rovnic vznikne difúzní rovnice

$$\partial_t u \ = div(D\nabla u) \tag{1.3}$$

1.2 Konvoluce a difúze

Následující rovnice popisuje vztah pouze jednorozměrné lineární difúzní rovnice s jednorozměrnou konvolucí s Gaussovým filtrem, protože v následujících kapitolách bude potřeba pouze jednorozměrná konvoluce s Gaussovým filtrem. Gaussův filtr je zadaný následující rovnicí.

$$G_\sigma = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Lineární difúzní rovnice

$$u(x,0) \quad = \quad f(x) \tag{1.4}$$

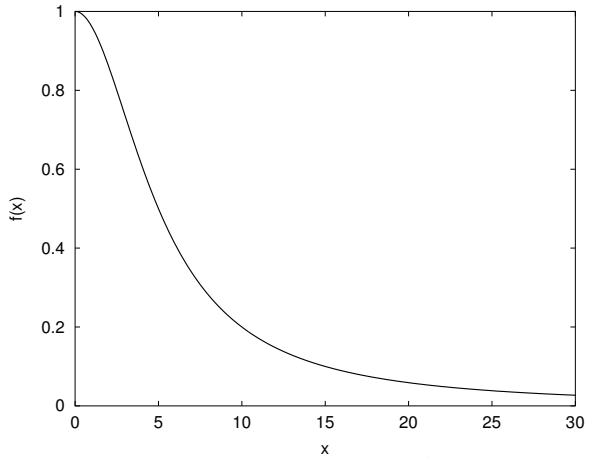
s počáteční podmínkou

$$\partial_t u \quad = \quad \partial_{xx} u \tag{1.5}$$

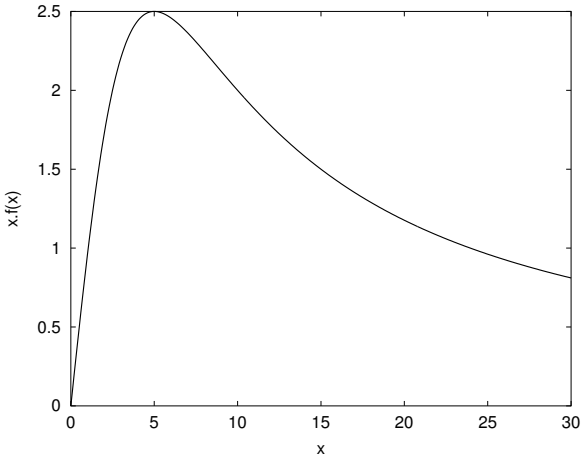
má řešení [4]

$$u(x,t) = \begin{cases} f(x) & t = 0 \\ (G_{\sqrt{2t}} * f)(x) & t > 0 \end{cases} \tag{1.6}$$

1. označení pro gradient, $\nabla u \ = (u_{x_1}, u_{x_2}, \ldots, u_{x_n})$; u_x je jedno z označení pro parciální derivaci
2. $div\phi$ je označení pro divergenci vektorového pole, $div\phi = \phi_{x_1} + \phi_{x_2} + \cdots + \phi_{x_n}$
3. ∂_t druhé možné označení pro parciální derivaci
4. $*$ je označení pro konvoluci



Obrázek 1.1: $g(x) = \frac{1}{1+x^2/\lambda^2}$



Obrázek 1.2: $xg(x) = \frac{x}{1+x^2/\lambda^2}$

1.3 Spojitá izotropní difúze

Myšlenka za izotropní difúzí je ta, že v bodech, které jsou součástí hrany, se omezí difúze. První otázka je, jak by měla vypadat funkce g a jaký by měla mít parametr. Mrázek [3] rozebírá historický vývoj, ve kterém byl první návrh od Perony a Malika. Podle nich by g měla být nenulová, monotónně klesající, parametrizovaná $|\nabla u|$ a s okrajovými podmínkami $g(0) = 1, g(\infty) = 0$. $|\nabla u|$ tady funguje jako detektor hrany. Hlavní problém takto navržené funkce je ten, že výsledný problém je špatně podmíněný. Špatně podmíněný problém znamená, že velice podobné vstupy můžou dávat velice různé výstupy. Pro odstranění tohoto problému stačí, aby funkce g byla parametrizovaná velikostí rozmazaného gradientu pomocí Gaussova filtru. Rovnice (1.3) pro izotropní difúzi má tedy tvar

$$\partial_t u \ = div[g(|\nabla u_\sigma|^2)\nabla u]. \tag{1.7}$$

Parametr σ se většinou volí tak, aby byl stejně velký jako diskretizační krok. Gaussův filtr by měl rozmazat útvary, které detektor hran po diskretizaci nedokáže rozeznat. Tyto útvary by neměly ovlivňovat výsledek difúze.

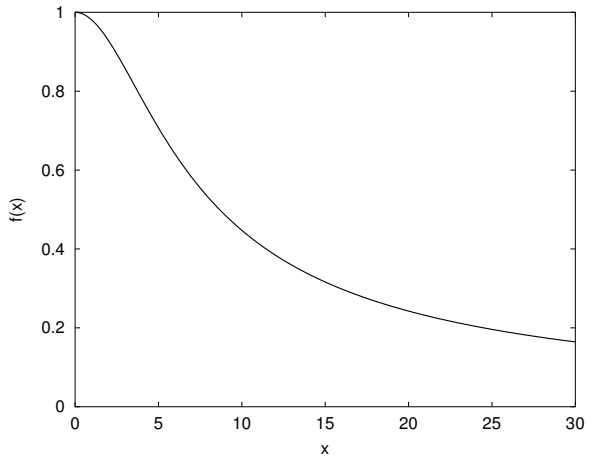
Je mnoho možností jak zvolit g. Na následujícím obrázku jsou zobrazené tři reprezentativní možnosti, jak může g vypadat. Jako první je zobrazená funkce, kterou navrhli Perona a Malik 1.1. Jako druhá je zobrazená funkce, kterou navrhl Charbonnier 1.3. Ta je významná mezi těmito třemi funkcemi proto, že při jejím dosazení do rovnice pro tok vznikne monotónní funkce. Význam monotónnosti je popsaný dále v textu. Jako třetí je zobrazená exponenciální funkce 1.5, kterou používá Weickert ve svém článku [1], podle kterého začala vznikat tato diplomová práce.

Funkce na obrázcích 1.1, 1.3 a 1.5 mají volný parametr λ. Tento parametr určuje potřebnou velikost gradientu, aby se bod považoval za hranu. Na obrázku 1.7 je příklad typické hrany pro 1–D spojitý případ. Na tomto 1–D příkladu jde vidět, jaký má na difúzi vliv tok, který je od určité hodnoty klesající. Po vynechání rozmazání Gaussovým filtrem a upravení rovnice (1.7) vznikne

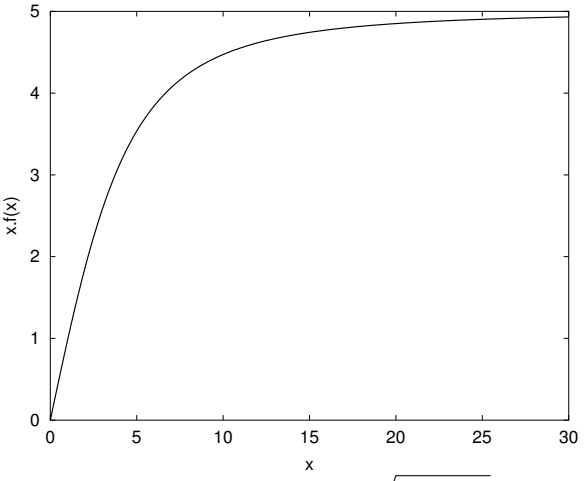
$$\phi(\partial_x u) = g(|\partial_x u|)\partial_x u \tag{1.8}$$

$$\partial_t u = \partial_x(\phi(\partial_x u)) \tag{1.9}$$

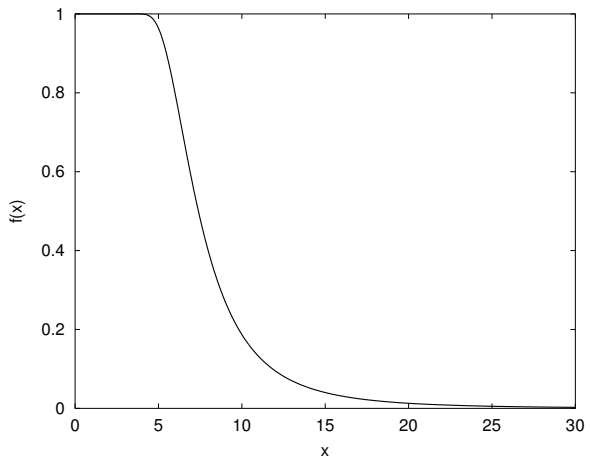
$$\partial_t u = \partial_x \phi(\partial_x u)\partial_{xx} u \tag{1.10}$$



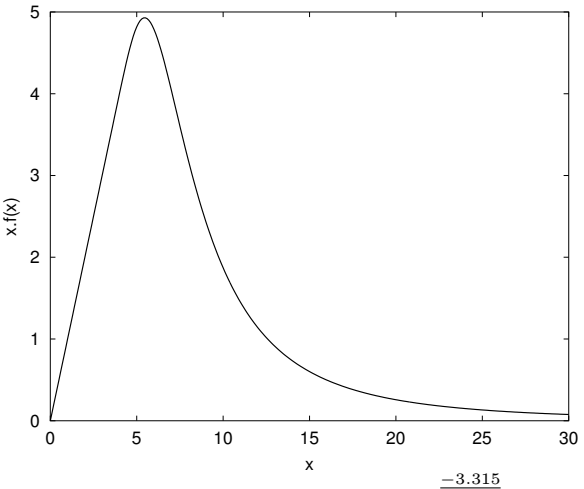
Obrázek 1.3: $g(x) = \frac{1}{\sqrt{1+x^2/\lambda^2}}$



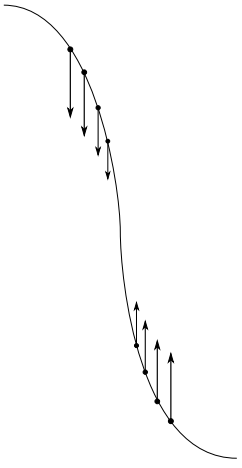
Obrázek 1.4: $xg(x) = \sqrt{\frac{x}{1+x^2/\lambda^2}}$



Obrázek 1.5: $g(x) = 1 - e^{\frac{-3.315}{(x/\lambda)^4}}$



Obrázek 1.6: $xg(x) = x - xe^{\frac{-3.315}{(x/\lambda)^4}}$



Obrázek 1.7: vzhled hrany, šipky znázorňují druhou derivaci

Difúze obrátí svůj směr, když hodnota $\partial_x \phi(\partial_x u) < 0$. Tím dokonce dojde k vylepšení hrany.

1.4 Spojitá anizotropní difúze

Anizotropní difúze je nejobecnější typ difúze, která vektor ∇u transformuje pomocí matice. Této matici se říká difúzní tenzor. Způsob konstrukce difúzního tenzoru rozlišuje dva druhy difúze, které jsou popsané ve dvou následujících podkapitolách.

1.4.1 Anizotropní difúze pro vylepšení hran

Důvod k zavedení dalšího typu difúze je ten, že izotropní difúze neodstraní šum z hran. Cílem je, aby difúze nerozmazávala napříč hranou, ale pouze podél hrany. Anizotropní difúze tento problém řeší tak, že nevyžaduje, aby tok byl rovnoběžný s ∇u_σ. Konstrukce difúzního tenzoru využívá následující vztah mezi maticí a jejími vlastními hodnotami a vektory.

Tvrzení: Nechť u, v jsou ortonormální vlastní vektory a λ_1, λ_2 jsou vlastní čísla matice A. Potom platí vztah [3]

$$A = \begin{pmatrix} u_x & v_x \\ u_y & v_y \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} u_x & u_y \\ v_x & v_y \end{pmatrix}. \tag{1.11}$$

Důkaz:

$$A = \begin{pmatrix} u_x & v_x \\ u_y & v_y \end{pmatrix} \begin{pmatrix} \lambda_1 u_x & \lambda_1 u_y \\ \lambda_2 v_x & \lambda_2 v_y \end{pmatrix} = \begin{pmatrix} \lambda_1 u_x^2 + \lambda_2 v_x^2 & \lambda_1 u_x u_y + \lambda_2 v_x v_y \\ \lambda_1 u_x u_y + \lambda_2 v_x v_y & \lambda_1 u_y^2 + \lambda_2 v_y^2 \end{pmatrix} \tag{1.12}$$

$$A \begin{pmatrix} u_x \\ u_y \end{pmatrix} = \begin{pmatrix} \lambda_1 u_x^3 + \lambda_2 u_x v_x^2 + \lambda_1 u_x u_y^2 + \lambda_2 u_y v_x v_y \\ \lambda_1 u_x^2 u_y + \lambda_2 u_x v_x v_y + \lambda_1 u_y^3 + \lambda_2 v_y^2 u_y \end{pmatrix} \tag{1.13}$$

$$A \begin{pmatrix} u_x \\ u_y \end{pmatrix} = \begin{pmatrix} \lambda_1 u_x (u_x^2 + u_y^2) + \lambda_2 v_x (v_x u_x + v_y u_y) \\ \lambda_1 u_y (u_x^2 + u_y^2) + \lambda_2 v_y (u_x v_x + u_y v_y) \end{pmatrix} = \begin{pmatrix} \lambda_1 u_x \\ \lambda_1 u_y \end{pmatrix} \tag{1.14}$$

Důkaz pro v je obdobný. Difúzní tenzor je zkonstruovaný pomocí předchozí věty, aby měl normalizované vlastní vektory $u || \nabla u_\sigma$, $v \perp \nabla u_\sigma$ a vlastní čísla $\lambda_1 = g(|\nabla u_\sigma|^2)$, $\lambda_2 = \varphi$. Vytvoření kolmého kolmého vektoru k danému vektoru je snadné. Stačí prohodit souřadnice x, y a u jedné z nich změnit znaménko. Mrázek [3] doporučuje, aby $\varphi = 0.2$, protože tato hodnota se více hodí pro AOS schéma, které difúzní tenzor dále zpracovává.

Konstrukce difúzního tenzoru pro tři dimenze je trochu složitější. Pro konstrukci se používá zobecnění předchozího tvrzení pro tři dimenze.

Tvrzení: Nechť u, v, w jsou ortonormální vlastní vektory a $\lambda_1, \lambda_2, \lambda_3$ jsou vlastní čísla matice A. Potom platí vztah

$$A = \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_x \\ u_z & v_z & w_z \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix}. \tag{1.15}$$

Důkaz:

$$A = \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix} \begin{pmatrix} \lambda_1 u_x & \lambda_1 u_y & \lambda_1 u_z \\ \lambda_2 v_x & \lambda_2 v_y & \lambda_2 v_z \\ \lambda_3 w_x & \lambda_3 w_y & \lambda_3 w_z \end{pmatrix} = \tag{1.16}$$

$$\begin{pmatrix} \lambda_1 u_x^2 + \lambda_2 v_x^2 + \lambda_3 w_x^2 & \lambda_1 u_x u_y + \lambda_2 v_x v_y + \lambda_3 w_x w_y & \lambda_1 u_x u_z + \lambda_2 v_x v_z + \lambda_3 w_x w_z \\ \lambda_1 u_x u_y + \lambda_2 v_x v_y + \lambda_3 w_x w_y & \lambda_1 u_y^2 + \lambda_2 v_y^2 + \lambda_3 w_y^2 & \lambda_1 u_y u_z + \lambda_2 v_y v_z + \lambda_3 w_y w_z \\ \lambda_1 u_x u_z + \lambda_2 v_x v_z + \lambda_3 w_x w_z & \lambda_1 u_y u_z + \lambda_2 v_y v_z + \lambda_3 w_y w_z & \lambda_1 u_z^2 + \lambda_2 v_z^2 + \lambda_3 w_z^2 \end{pmatrix}$$

$$A \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = \tag{1.17}$$

$$\begin{pmatrix} \lambda_1 u_x (u_x^2 + u_y^2 + u_z^2) + \lambda_2 v_x (u_x v_x + u_y v_y + u_z v_z) + \lambda_3 w_x (u_x w_x + u_y w_y + u_z w_z) \\ \lambda_1 u_y (u_x^2 + u_y^2 + u_z^2) + \lambda_2 v_y (u_x v_x + u_y v_y + u_z v_z) + \lambda_3 w_y (u_x w_x + u_y w_y + u_z w_z) \\ \lambda_1 u_z (u_x^2 + u_y^2 + u_z^2) + \lambda_2 v_z (u_x v_x + u_y v_y + u_z v_z) + \lambda_3 w_z (u_x w_x + u_y w_y + u_z w_z) \end{pmatrix} \tag{1.18}$$

$$A \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = \begin{pmatrix} \lambda_1 u_x \\ \lambda_2 u_y \\ \lambda_3 u_z \end{pmatrix} \tag{1.19}$$

Důkazy pro v, w jsou obdobné. Cílem této difúze ve třech dimenzích je, aby nerozmazávala napříč plochou, která je určená ∇u_σ, ale pouze uvnitř plochy. Pro konstrukci difúzního tenzoru je potřeba najít dva kolmé jednotkové vektory, které leží v této rovině. Rovnice roviny, která má normálový vektor ∇u_σ a prochází počátkem je

$$\partial_x u_\sigma x + \partial_y u_\sigma y + \partial_z u_\sigma z \ = 0. \tag{1.20}$$

Jestliže například $\partial_x u_\sigma \neq 0$, potom dva lineárně nezávislé vektory, které leží v této rovině, jsou například $v' = (-\partial_y u_\sigma / \partial_x u_\sigma \, 1, 0)^\top$, $w' = (-\partial_z u_\sigma / \partial_x u_\sigma, 0, 1)^\top$. Tyto vektory, které vznikly postupným dosazením hodnot $y = 1, z = 0$ a $y = 0, z = 1$, nemusí být ani ortogonální. K upravení vektorů na ortogonální se dá použít Gram–Schmidtův (GS) ortogonalizační proces [10]. GS v prvním kroku vybere libovolný vektor, podle kterého se druhý vektor upraví. Úprava spočívá v odečtení od druhého vektoru projekci druhého vektoru na první. Matematický zápis tohoto popisu je

$$v = v' \tag{1.21}$$

$$p = \frac{w' \cdot v}{v \cdot v} v \tag{1.22}$$

$$w = w' - p. \tag{1.23}$$

Výsledné vektory u, v, w stačí už jen normalizovat a přiřadit jim vlastní čísla $\lambda_1 = g(|\nabla u|^2)$, $\lambda_2 = \varphi$, $\lambda_3 = \varphi$. Difúzní tenzor se sestrojí pomocí těchto vypočítaných hodnot a předchozí věty. Tentokrát může být φ větší. AOS schéma zvládalo výpočet i pro $\varphi \geq 0.8$.

1.4.2 Anizotropní difúze pro zvýraznění koherence

Filtr pro zvýraznění koherence se dá použít pro spojení a vyhlazení čar, ale k tomu potřebuje mít znalost většího okolí. Tentokrát se nedá použít ∇u_σ, protože se dva vektory s opačným znaménkem navzájem vyruší. Aby Weickert [2] tento problém vyřešil, použil strukturu, která se jmenuje strukturní tenzor $J_\rho(\nabla u_\sigma)$. Tato struktura se vytvoří vynásobením vektoru ∇u s jeho transpozicí $(\nabla u)^\top$ a následným rozmazáním Gaussovým filtrem s rozptylem ρ. Následující příklad ukazuje chování strukturního tenzoru při zprůměrování vektorů $(1,0),(-1,0)$.

$$\frac{1}{2}[(1,0)^\top(1,0)+(-1,0)^\top(-1,0)] = \frac{1}{2}\left[\begin{pmatrix}1&0\\0&0\end{pmatrix}+\begin{pmatrix}1&0\\0&0\end{pmatrix}\right] = \begin{pmatrix}1&0\\0&0\end{pmatrix} \tag{1.24}$$

Matice $J_\rho(\nabla u_\sigma)$ je pozitivně semidefinitní a má ortogonální vlastní vektory μ_1, μ_2 [2]. Vlastní vektory popisují významné směry v okolí, které je určené parametrem ρ. Vlastní čísla určují kontrast v těchto směrech a dají se použít ke zjištění typu okolí. Konstantní plochy mají $\mu_1 = \mu_2$, přímé hrany mají $\mu_1 \gg \mu_2$ a rohy mají $\mu_1 > \mu_2 \gg 0$.

Cílem je vyhlazovat podél významných směrů. Velikost vyhlazování by měla růst s růstem $(\mu_1-\mu_2)^2$. Vlastní vektory tentokrát zůstávají stejné a mění se pouze vlastní čísla. Jestliže $\mu_1 > \mu_2$, potom $\lambda_1 = \alpha$ a [2]

$$\lambda_2 = \begin{cases} \alpha & \mu_1 = \mu_2 \\ \alpha + (1-\alpha)e^{\frac{-3.315}{(\mu_1-\mu_2)^2}} & \mu_1 \neq \mu_2. \end{cases} \tag{1.25}$$

Konstanta -3.315 je stejná jako u difúze na zvýraznění hran. Weickert v [2] používá konstantu -1, ale rozdíl výsledků není pozorovatelný. Vlastní hodnota λ_1 určuje vyhlazování napříč významného směru a λ_2 určuje vyhlazování podél významného směru. Parametr $\alpha > 0$ je potřeba k důkazu, že problém je dobře podmíněný. K získání vlastních vektorů je potřeba vyřešit systém rovnic

$$\begin{pmatrix}a&b\\b&c\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix} = \mu\begin{pmatrix}x\\y\end{pmatrix}. \tag{1.26}$$

Po úpravě vznikne homogenní systém lineárních rovnic, který má netriviální řešení právě tehdy, když matice, která ho reprezentuje, je singulární. Po rozvinutí determinantu

$$\begin{vmatrix} a-\mu & b \\ b & c-\mu \end{vmatrix} = 0 \tag{1.27}$$

vznikne charakteristický kvadratický polynom

$$(a-\mu)(c-\mu) - b^2 = \mu^2 - \mu(a+c) + ac - b^2 = 0. \tag{1.28}$$

V následujících rovnicích je shrnutý výpočet vlastních hodnot a jednotkových vlastních vektorů.

$$D = (a+c)^2 - 4ac + 4b^2 \tag{1.29}$$

$$\mu_{1,2} = \frac{a+c \pm \sqrt{D}}{2} \tag{1.30}$$

$$(a-\mu)x + by = 0 \tag{1.31}$$

$$bx + (c - \mu)y = 0 \tag{1.32}$$

$$x = \frac{-b}{\sqrt{b^2 + (a - \mu)^2}} \tag{1.33}$$

$$y = \frac{a - \mu}{\sqrt{b^2 + (a - \mu)^2}} \tag{1.34}$$

Nyní je už možné sestrojit difúzní tenzor pomocí vypočítaných hodnot a předchozího tvrzení. 3–D verze této difúze tady není popsaná, protože požaduje víc paměti, než je na GPU. Toto je zdůvodněné dále v textu.

Kapitola 2

Diskretizace

V této kapitole jsou popsané tři metody řešení difúzní rovnice (1.7) První popsaná metoda je explicitní Eulerovo schéma, které je nejjednodušší, ale vyžaduje malý krok. Druhá popsaná metoda je implicitní schéma, které se od explicitního liší pouze tím, že aproximuje jednu parciální derivaci jiným způsobem. Toto schéma odstraňuje problém s malým krokem, ale nelze s ním jednoduše vyřešit vícerozměrné případy. AOS modifikuje implicitní schéma tak, aby se daly jednoduše řešit vícerozměrné problémy. Jestliže není napsáno jinak, tak se používá systém souřadnic, který je na obrázku 2.1.

2.1 Explicitní schéma

Po upravení rovnice (1.7) pro 1–D případ vznikne rovnice

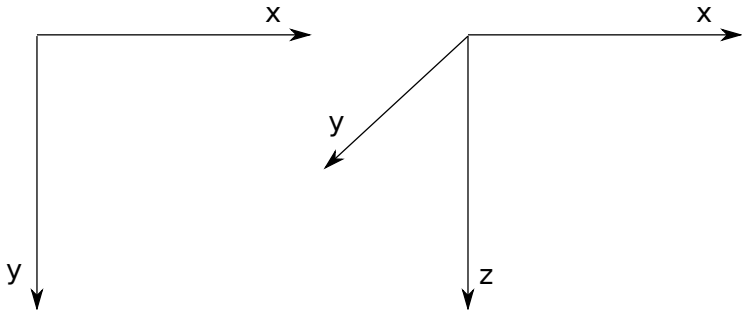
$$\partial_t u \ = \partial_x (g(|\partial_x u_\sigma|)\partial_x u). \tag{2.1}$$

Diskrétní data u_i^k vzniknou vzorkováním spojité funkce u. Hodnota u_i^k odpovídá funkční hodnotě v bodě $u(ih, \tau k)$, kde h je vzdálenost bodů na mřížce a τ je velikost časového kroku. V dalším textu se předpokládá, že $h = 1$. Nahrazením derivací pravou diferenční derivací a centrální diferenční derivací vzniknou rovnice

$$\begin{aligned} \frac{u_i^{k+1} - u_i^k}{\tau} &= (g\partial_x u)_{i+\frac{1}{2}} - (g\partial_x u)_{i-\frac{1}{2}}, & (2.2)\\ &= g_{i+\frac{1}{2}}^k (u_{i+1}^k - u_i^k) - g_{i-\frac{1}{2}}^k (u_i^k - u_{i-1}^k). & (2.3) \end{aligned}$$

Po nahrazení $g_{i\pm\frac{1}{2}}^k$ aritmetickým průměrem $\frac{g_i^k + g_{i\pm1}^k}{2}$ vznikne rovnice

$$\frac{u_i^{k+1} - u_i^k}{\tau} = \frac{g_i^k + g_{i+1}^k}{2}(u_{i+1}^k - u_i^k) - \frac{g_{i-1}^k + g_i^k}{2}(u_i^k - u_{i-1}^k). \tag{2.4}$$



Obrázek 2.1: systém souřadnic

V krajních bodech musí platit, že neprobíhá žádný tok mezi vnějším okolím a daným bodem. Po vyjádření rovnice pro levý krajní bod vznikne

$$\frac{u_0^{k+1} - u_0^k}{\tau} = \frac{g_0^k + g_1^k}{2}(u_1^k - u_0^k) - \frac{g_{-1}^k + g_0^k}{2}(u_0^k - u_{-1}^k). \tag{2.5}$$

Člen $\frac{g_{-1}^k+g_0^k}{2}(u_0^k - u_{-1}^k)$, který je nulový v případě, když $u_0^k = u_{-1}^k$, odpovídá toku mezi vnějším okolím a levým krajním bodem. Pro pravý krajní bod vznikne po úpravách obdobná rovnice. Z toho plyne, že zrcadlení dat na krajích je dostatečné opatření, aby neprobíhal tok mezi krajními body a vnějším okolím. Při předpokladu zrcadlení dat dojde k zjednodušení rovnice (2.4) v krajních bodech o člen, který popisuje tok mezi krajním bodem a okolím. Rovnici (2.4) lze vyjádřit pomocí násobení matice vektorem jako

$$\frac{u^{k+1} - u^k}{\tau} = Au^k. \tag{2.6}$$

$$A = \frac{1}{2}\begin{pmatrix} -(g_0^k + g_1^k) & g_0^k + g_1^k & 0 & 0 \\ g_0^k + g_1^k & -(g_0^k + 2g_1^k + g_2^k) & g_1^k + g_2^k & 0 \\ \ddots & \ddots & \ddots & 0 \\ 0 & g_{i-1} + g_i & -(g_{i-1} + 2g_i + g_{i+1}) & g_i + g_{i+1} \\ 0 & \ddots & \ddots & \ddots \\ 0 & 0 & g_{N-2} + g_{N-1} & -(g_{N-2} + g_{N-1}) \end{pmatrix} \tag{2.7}$$

Po vyjádření u^{k+1} vznikne rovnice

$$u^{k+1} = [I + \tau A]u^k. \tag{2.8}$$

Weickert [1] uvádí a zároveň ověřuje podmínky, které musí splňovat diskrétní difúzní proces tvaru

$$u^0 = f \tag{2.9}$$
$$u^{k+1} = Q(u^k)u^k \qquad \forall k \in N_0, \tag{2.10}$$

aby byl dobře podmíněný a měl další užitečné vlastnosti. Mezi užitečné vlastnosti například patří zachování průměru šedé a konvergence ke konstantní hodnotě. Matice (2.7) je na první pohled parametrizovaná špatným parametrem u_σ^k, ale ne u^k. To ale nevadí, protože rozmazání Gaussovým filtrem a získání parametru u_σ^k lze provést pomocí násobení vhodnou maticí. Takže z parametru u^k lze jednoduše získat parametr u_σ^k.

2.1.1 Vícerozměrný případ

Vícerozměrná izotropní difúze má rovnici

$$\partial_t u = \sum_{l=0}^{m-1} \partial_{x_l}[g(|\nabla u_\sigma|^2)\partial_{x_l} u], \tag{2.11}$$

kde m je počet dimenzí. Výsledek diskretizace vícerozměrné funkce se dá reprezentovat vektorem, který má velikost $N = N_0 \cdot N_1 \ldots N_{m-1}$, kde N_i je počet vzorků na ose x_i. Pro

každou osu l jde vytvořit matici A_l podobně jako pro rovnici (2.1). Jeden iterační krok jde vyjádřit jako

$$u^{k+1} = [I \ + \tau \sum_{l=0}^{m} A_l] u^k. \tag{2.12}$$

Tento zápis explicitního schématu je vhodný jenom pro ověření podmínek, které diskrétní difúzní proces musí splňovat. Weickert [1] ve svém článku odkazuje na tyto důkazy. Největší možný časový krok, který lze použít, se s počtem dimenzí zmenšuje na

$$\tau < \frac{1}{2m}. \tag{2.13}$$

2.2 Částečně implicitní schéma

Jeden krok explicitního schématu není výpočetně náročný, ale je jich zapotřebí mnoho, protože výpočet je stabilní jenom pro malé hodnoty τ. Weickert [1] ukázal, že pro 1–D difúzi musí být $\tau < 1/2$. Nevýhoda explicitního schématu není v přesnosti, ale požadavcích, které zaručují jeho stabilitu [1]. Proto Weickert [1] ověřil podmínky i pro částečně implicitní schéma

$$\frac{u^{k+1} - u^k}{\tau} = A u^{k+1}. \tag{2.14}$$

Stabilita tohoto schématu není závislá na velikosti τ, ale velikost τ stále ovlivňuje přesnost výpočtu. Po upravení této rovnice vznikne

$$u^{k+1} = (I \ - \tau A)^{-1} u^k. \tag{2.15}$$

Tuhle rovnici lze řešit jako systém lineárních rovnic, který je tridiagonální a diagonálně dominantní, takže se dá efektivně řešit. Efektivně v tomto případě znamená, že optimální algoritmus má lineární časovou i prostorovou složitost. Možné způsoby řešení tohoto problému jsou popsané v čtvrté kapitole.

2.2.1 Vícerozměrný případ

Implicitní obdoba k (2.12) je

$$u^{k+1} = [I - \tau \sum_{l=0}^{m} A_l]^{-1} u^k. \tag{2.16}$$

Matici A, která vznikne součtem matice A_l, nejde uspořádat tak, aby nenulové prvky byly jen v malém $\pm m$ okolí hlavní diagonály [1]. Stabilita tohoto schématu pro více dimenzí není také závislá na velikosti τ, ale matici A nejde efektivně uložit ani vyřešit odpovídající systém lineárních rovnic.

2.3 AOS

AOS řeší předchozí problém tím, že modifikuje (2.16) na

$$u^{k+1} = \frac{1}{m} \sum_{l=0}^{m-1} [I - m\tau A_l]^{-1} u^k. \tag{2.17}$$

Každý člen v sumě

$$B_l = I \ - m\tau A_l \tag{2.18}$$

popisuje 1–D izotropní difúzi podél osy x_l. Pro každou osu lze přeskládat vektor u^k tak, aby matice B_l byla tridiagonální. Tuto matici lze rozdělit na menší. Například z matice B_x, která určuje difúzi ve směru řádků, vznikne matice pro každý řádek vstupních dat (obrázku). Toto je možné, protože pravý krajní bod řádku i nezávisí na levém krajním bodu řádku $i+1$ a naopak. Část matice B_x, ve které dochází k přechodu z řádku i na řádek $i+1$, vypadá jako

$$B_x = \begin{pmatrix} \cdots & -a & 1+a & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 1+b & -b & \cdots \end{pmatrix}. \tag{2.19}$$

I u AOS Weickert [1] ověřil podmínky, které diskrétní difúzní proces musí splňovat.

2.4 Rozklad difúzního tenzoru

Cílem této kapitoly je nahradit difúzní tenzor u anizotropního filtrování systémem izotropních difúzí. V rámci konzistence s 3–D případem, ve kterém je jednoduší popsat směr pomocí vektoru, budou směry ve 2–D případu popsané také pomocí vektorů. Pro 2–D případ se difúzní tenzor

$$D = \begin{pmatrix} a & b \\ b & c \end{pmatrix} \tag{2.20}$$

rozloží do čtyř izotropních difúzí, které budou probíhat ve směrech

$$e_{-1} = (\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}), e_0 = (1,0), e_1 = (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}), e_2 = (0,1). \tag{2.21}$$

Mrázek [3] říká aproximaci konzistentní, jestliže

$$\sum_{l=-1}^{2} \partial_{e_l}(\alpha_l \partial_{e_l} u) = div(D\nabla u). \tag{2.22}$$

Po úpravě pravé strany vznikne

$$div \begin{pmatrix} a\partial_x u + b\partial_y u \\ b\partial_x u + c\partial_y u \end{pmatrix} = a\partial_{xx}u + 2b\partial_{xy}u + c\partial_{yy}u. \tag{2.23}$$

Po úpravě členů na levé straně s využitím vztahu $\partial_e = \nabla u \cdot e$ vznikne

$$\sum_{l=-1}^{2} \partial_{l_e}\alpha_l({}^l e_x \partial_x u \ + {}^l e_y \partial_y u) = \sum_{l=-1}^{2} \alpha_l({}^l e_x^2 \partial_{xx} u \ + {}^l e_x {}^l e_y \partial_{xy} u \ + {}^l e_x {}^l e_y \partial_{xy} u \ + {}^l e_y^2 \partial_{yy} u). \tag{2.24}$$

Po sečtení koeficientů u jednotlivých parciálních derivací vzniknou následující rovnice.

$$\partial_{xx}u: \ \sum_{l=-1}^{2} {}^l e_x^2 \alpha_l = a \ \Big| \ \partial_{xy}u: \ \sum_{l=-1}^{2} {}^l e_x {}^l e_y 2\alpha_l = 2b \ \Big| \ \partial_{yy}u: \ \sum_{l=-1}^{2} {}^l e_y^2 \alpha_l = c \tag{2.25}$$

Po číselném dosazení vznikne systém lineárních rovnic

$$\begin{pmatrix} \frac{1}{2} & 1 & \frac{1}{2} & 0 \\ -1 & 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} \alpha_{-1} \\ \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} a \\ 2b \\ c \end{pmatrix}. \tag{2.26}$$

Po úpravách této matice vznikne

$$\begin{pmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \alpha_{-1} \\ \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} 2a \\ a+b \\ b+c \end{pmatrix}. \tag{2.27}$$

Weickert [1] v důkazech předpokládá, že $\alpha_i \geq 0$. Porušení této podmínky má za následek, že B^{-1} může mít záporné prvky. Jedna z podmínek, kterou každý difúzní proces musí splňovat, ale je, že tato inverzní matice musí mít všechny prvky kladné. Podmínka na nezápornost α_i vytvoří z předchozího sytém lineárních rovnic následující systém lineárních nerovnic.

$$\alpha_2 = x \geq 0 \tag{2.28}$$
$$\alpha_1 = b + c - x \geq 0 \tag{2.29}$$
$$\alpha_0 = a - c + x \geq 0 \tag{2.30}$$
$$\alpha_{-1} = c - b - x \geq 0 \tag{2.31}$$

Dvě předchozí nerovnice omezují velikost x shora a dvě zdola. Interval hodnot, které může x nabývat, lze vyjádřit jako

$$[\max(0, c-a), \min(b+c, c-b)]. \tag{2.32}$$

Weickert [2] ukázal, že tento interval je prázdný, jestliže spektrální číslo podmíněnosti $\kappa > 3 + 2\sqrt{(2)} \approx 5.83$. Definice různých čísel podmíněnosti a různé způsoby jejich výpočtu jsou uvedené například v [9]. Mrázek [3] zkoušel, který bod z tohoto intervalu je nejvhodnější. Došel k závěru, že nejvhodnější je použít střed intervalu. Když použil krajní hodnoty k filtrování dvourozměrného Gausiánu pomocí difúze na vylepšení hran, tak získal přibližně tvar čtverce a kosočtverce, ale při použití středu intervalu získal přibližně tvar osmiúhelníku. Mrázek [3] sice doporučuje, aby $\varphi = 0.2$, ale lepší možnost je použít vztah $\kappa = |\lambda_{\max}/\lambda_{min}|$ [9] a přizpůsobovat difúzi po směru hrany, jestliže je to nutné. V případě difúze pro zvýšení koherence je možné zmenšit difúzi ve významném směru a zachovat malou difúzi napříč významným směrem.

2.4.1 Trojrozměrný případ

V trojrozměrném případě se difúzní tenzor

$$\begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix} \tag{2.33}$$

rozloží do devíti izotropních difúzí, které budou probíhat ve směrech

$$e_0 = (1,0,0), e_1 = (0,1,0), e_2 = (0,0,1), \tag{2.34}$$

$$e_3 = (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0), e_4 = (\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}, 0), e_5 = (\frac{\sqrt{2}}{2}, 0, \frac{\sqrt{2}}{2}), \tag{2.35}$$

$$e_6 = (\frac{\sqrt{2}}{2}, 0, -\frac{\sqrt{2}}{2}), e_7 = (0, \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}), e_8 = (0, \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}). \tag{2.36}$$

V úvahu by ještě připadalo rozložit difúzní tenzor do třinácti směrů, ale tahle možnost by se špatně implementovala. V následujícím odvození se bude používat ${}^k e_x$ pro zápis souřadnice x vektoru e_k. Stejně jako v 2–D případu musí být aproximace konzistentní a tedy splňovat rovnici

$$\sum_{l=0}^{8} \partial_{e_k}(\alpha_k \partial_{e_k} u) = div(D\nabla u). \tag{2.37}$$

Po úpravě pravé strany vznikne

$$div \begin{pmatrix} a\partial_x u + b\partial_y u + c\partial_z u \\ b\partial_x u + d\partial_y u + e\partial_z u \\ c\partial_x u + e\partial_y u + f\partial_z u \end{pmatrix} = a\partial_{xx} u + 2b\partial_{xy} u + 2c\partial_{xz} u + d\partial_{yy} u + 2e\partial_{yz} u + f\partial_{zz} u \tag{2.38}$$

Po úpravě členů na levé straně vznikne

$$\textstyle\sum_{l=0}^{8} \partial_{{}^l e} \alpha_l ({}^l e_x \partial_x u + {}^l e_y \partial_y u + {}^l e_z \partial_z u) = \tag{2.39}$$

$$\textstyle\sum_{l=0}^{8} \alpha_l ({}^l e_x^2 \partial_{xx} u + 2\,{}^l e_x\, {}^l e_y \partial_{xy} u + 2\,{}^l e_x\, {}^l e_z \partial_{xz} u + {}^l e_y^2 \partial_{yy} u + 2\,{}^l e_y\, {}^l e_z \partial_{yz} u + {}^l e_z^2 \partial_{zz} u) \tag{2.40}$$

Po sečtení koeficientů u jednotlivých parciálních derivací vzniknou následující rovnice.

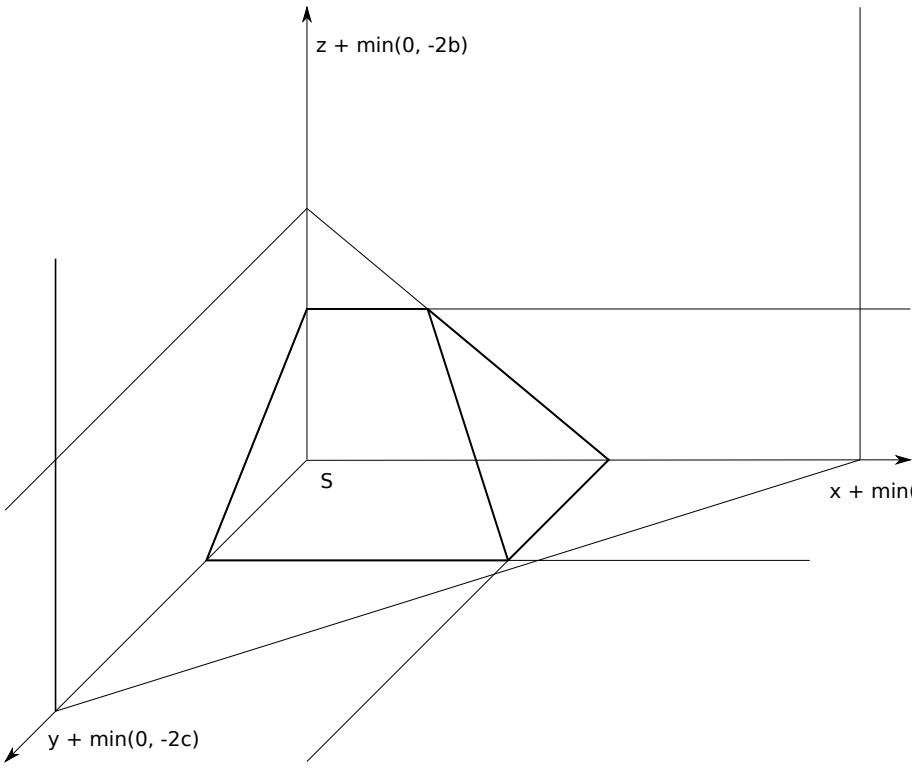
$$\begin{array}{l|l|l} \partial_{xx} u: \ \sum_{l=0}^{8} {}^l e_x^2 \alpha_l = a & \partial_{xy} u: \ \sum_{l=0}^{8} {}^l e_x\, {}^l e_y 2\alpha_l = 2b & \partial_{xz} u: \ \sum_{l=0}^{8} {}^l e_x\, {}^l e_z 2\alpha_l = 2c \\ \partial_{yy} u: \ \sum_{l=0}^{8} {}^l e_y^2 \alpha_l = d & \partial_{yz} u: \ \sum_{l=0}^{8} {}^l e_y\, {}^l e_z 2\alpha_l = 2e & \partial_{zz} u: \ \sum_{l=0}^{8} {}^l e_z^2 \alpha_l = f \end{array} \tag{2.41}$$

Po číselném dosazení vznikne systém lineárních rovnic

$$\begin{pmatrix} 1 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \\ \alpha_8 \end{pmatrix} = \begin{pmatrix} a \\ 2b \\ 2c \\ d \\ 2e \\ f \end{pmatrix}. \tag{2.42}$$

Po úpravách této matice vznikne

$$\begin{pmatrix} 1 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \\ \alpha_8 \end{pmatrix} = \begin{pmatrix} a \\ d \\ f \\ 2b \\ 2c \\ 2e \end{pmatrix}. \tag{2.43}$$



Obrázek 2.2: geometrický význam nerovnic

Podmínka na nezápornost také zde vytvoří systém lineárních nerovnic s volnými proměnnými x, y, z.

$$\alpha_8 = x \geq 0 \tag{2.44}$$
$$\alpha_7 = 2e + x \geq 0 \tag{2.45}$$
$$\alpha_6 = y \geq 0 \tag{2.46}$$
$$\alpha_5 = 2c + y \geq 0 \tag{2.47}$$
$$\alpha_4 = z \geq 0 \tag{2.48}$$
$$\alpha_3 = 2b + z \geq 0 \tag{2.49}$$
$$\alpha_2 = f - x - y - e - c \geq 0 \tag{2.50}$$
$$\alpha_1 = d - x - z - e - b \geq 0 \tag{2.51}$$
$$\alpha_0 = a - y - z - c - b \geq 0 \tag{2.52}$$

Tento systém je možné vyřešit pomocí analytické geometrie. Tři volné proměnné určují bod v prostoru X, který musí splňovat předchozí nerovnice. Prvních šest nerovnic určuje poloprostory, ohraničené rovinami, které jsou vždy rovnoběžné s jednou z průměten. Poslední tři nerovnice určují poloprostory ohraničené rovinami, které jsou vždy kolmé na jednu z průměten. Obrázek 2.2 ukazuje příklad množiny bodů, která může vzniknout z předchozích nerovnic. Z obrázku plyne, že množina přípustných řešení musí být konvexní.

Hledání přípustného bodu, který by neměl přímo ležet na žádné stěně, probíhá nezávisle v každém voxelu a příslušná procedura musí být tedy rychlá. Jedno z nejjednodušších řešení je najít průsečík roviny a přímky, která prochází bodem S a má směr $v = (\sqrt{3}/3, \sqrt{3}/3, \sqrt{3}/3))$.

Směr v má výhodu, že neupřednostňuje žádnou osu. Procedura tedy zjistí maximální vzdálenost, o kterou se bod může posunout tak, aby stále ležel na správné straně všech tří rovin. Po dosazení rovnice přímky $P = S + tv$ za X do všech šesti nerovnic a jejich upravení vzniknou nerovnice

$$t \geq -\frac{S_x}{v_x} \tag{2.53}$$

$$t \geq -\frac{2e + S_x}{v_x} \tag{2.54}$$

$$t \geq -\frac{S_y}{v_y} \tag{2.55}$$

$$t \geq -\frac{2c + S_y}{v_y} \tag{2.56}$$

$$t \geq -\frac{S_z}{v_z} \tag{2.57}$$

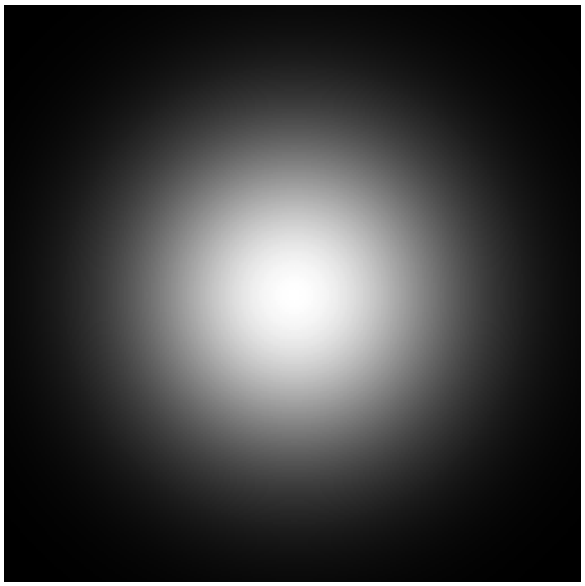
$$t \geq -\frac{2b + S_z}{v_z} \tag{2.58}$$

$$t \leq \frac{f - S_x - e - c - S_y}{v_x + v_y} \tag{2.59}$$

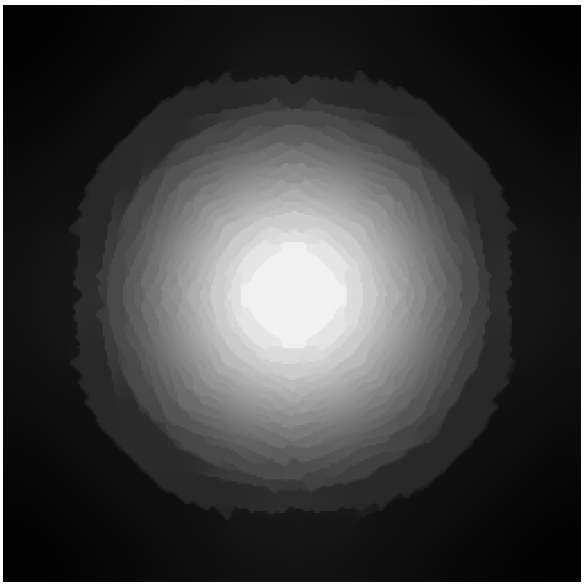
$$t \leq \frac{d - S_x - e - b - S_z}{v_x + v_z} \tag{2.60}$$

$$t \leq \frac{a - S_y - c - b - S_z}{v_y + v_z}. \tag{2.61}$$

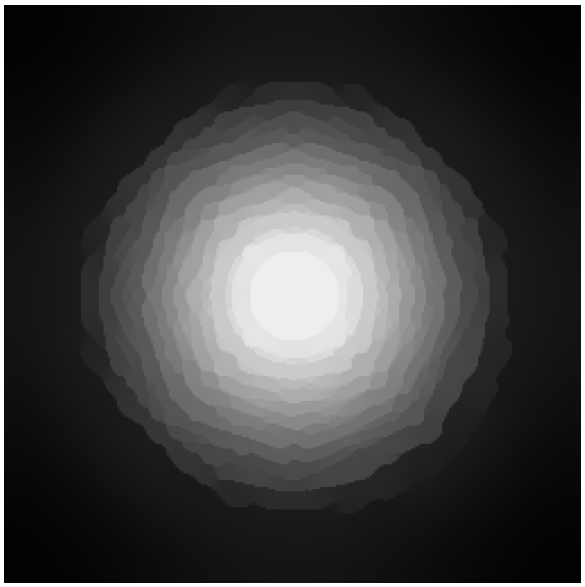
Tím vznikne obdobný interval jako u 2–D případu. Otázka je stejná jako u 2–D případu, kterou hodnotu z vypočítaného intervalu použít. Testovací obrázek 2.3 je poskládaný z několika gausiánů s $\sigma_x = \sigma_y = 50$ nad sebou. Na obrázcích 2.4, 2.5 a 2.6 jsou výsledky difúze pro vylepšení hran při použití nejmenší hodnoty, prostřední hodnoty a největší hodnoty intervalu. Difúze měla parametry $\tau_\sigma = 0.5, n_\sigma = 1, \tau = 5, n = 50, \lambda = 1$. Výsledkem by mělo být po částech konstantní mezikruží. Tohoto nedosáhla ani jedna z možností. Jestliže by se porovnával pouze vzhled vnitřního kruhu, potom by nejlepší výsledek byl na obrázku 2.5, na kterém je vnitřní část neporušená a vypadá jako osmiúhelník. V 3–D případu se osvědčil stejný postup jako v 2–D případu, tj. nepoužívat krajní hodnoty, ale střed intervalu.



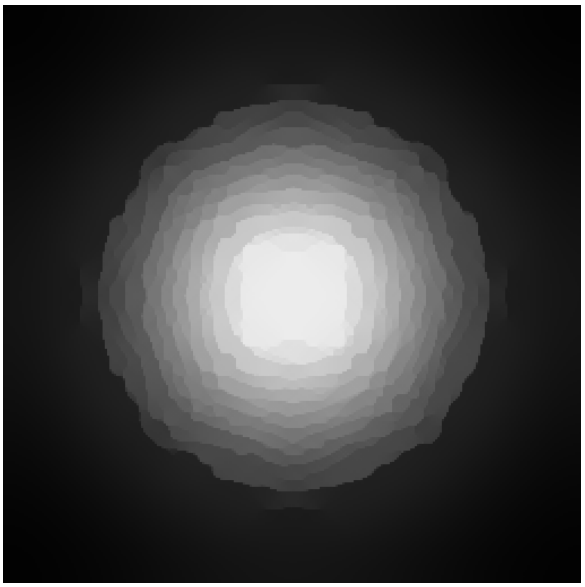
Obrázek 2.3: vstupní obrázek



Obrázek 2.4: výstup bez posunu bodu



Obrázek 2.5: výstup s posunem do poloviny intervalu



Obrázek 2.6: výstup s posunem na konec intervalu

Kapitola 3

CUDA

CUDA [4] je architektura pro programování obecných výpočtů na GPU. GPU se v programu chová jako koprocesor, na kterém může procesor spouštět výpočty. Hlavní jazyk pro programování CUDA je C, který je rozšířený o konstrukce, které jsou navíc v programovacím modelu pro CUDA.

Architektura CUDA je produktem firmy NVIDIA a prošla od svého uvedení v roce 2006 několika revizemi *(compute capability)*. V revizích nedocházelo k změnám programovacího modelu, ale docházelo k přidání funkcionality nebo zjednodušení přístupu k paměti. Dále v textu se předpokládá použití revize 1.2, protože klade menší požadavky na přístup do hlavní paměti GPU – globální paměti. Před výběrem vhodného paralelního algoritmu je nutné se seznámit s programovacím modelem od CUDA, protože přibližně desetkrát vyšší teoretický výpočetní výkon GPU oproti CPU je vykoupený určitými omezeními, kterými se musí programátor řídit.

3.1 Korespondence fyzických zařízení s pojmy v modelu

CUDA GPU obsahuje několik multiprocesorů, které nezávisle zpracovávají bloky vláken. Všechna vlákna provádějí stejnou funkci, které se říká kernel. Vlákno se dozví o své pozici pomocí indexu bloku *(blockIdx.[x,y])* a indexu, který udává pozici vlákna uvnitř bloku *(threadIdx.[x,y,z])*. Jestliže je blok vláken přiřazený nějakému multiprocesoru, tak na něm zůstane až do ukončení svého výpočtu. Na jednom multiprocesoru může být ale víc bloků aktivních, takže plánovač může přepínat mezi bloky, jestliže nějaký blok není připravený k běhu. Výpočetní výkon GPU tedy nezávisí na revizi, ale hlavně na počtu multiprocesorů – schopnosti paralelně zpracovávat bloky. Multiprocesor obsahuje 8 procesorů, které v každém taktu vykonávají stejnou instrukci. Počet procesorů je v některých revizích jiný.

Multiprocesor zpracovává vlákna v bloku po warpech. *Warp* je dávka vždy 32 vláken. Pro využití dostupného výkonu je nutné, aby jednotlivé warpy vykonávaly stejnou instrukci. V případě, že vlákna ve warpu vykonávají rozdílnou instrukci, tak se tyto běhy serializují. Výsledek bude tedy správný, ale dojde ke ztrátě výkonu. V případě, že warp není připravený k běhu, tak plánovač může začít zpracovávat jiný warp.

Vlákna v jednom bloku se mohou synchronizovat a sdílet data ve sdílené paměti, která je umístěná na každém multiprocesoru a má velikost 16kB. Velikost sdílené paměti je v některých revizích jiná. Pro rychlejší vystavení dat je sdílená paměť rozdělená do 16 paměťových bank, do nichž můžou vlákna přistupovat paralelně. Přístup do sdílené paměti je skoro stejně rychlý jako k registrům, za podmínky, že vlákna přistupují do různých bank, nebo že všechna vlákna přistupují ke stejné paměťové buňce (v takovém případě dochází k broadcastu).

Pro všechny bloky je k dispozici společná paměť konstant, pro kterou má každý multiprocesor vyrovnávací paměť. Paměť konstant má velikost 16kB a vyrovnávací paměť pro každý

multiprocesor má velikost 8kB. Tahle paměť obsluhuje vždy jeden požadavek na paměťové místo za takt a umí broadcasty.

Část globální paměti může být prohlášena za texturu. Tato paměť je potom pouze pro čtení a je jí přiřazená vyrovnávací paměť. Do vyrovnávací paměti umísťuje okolí čteného prvku. Tohle může být i nevýhoda, protože se může načítat spoustu zbytečných prvků.

K využití celé propustnosti globální paměti je nutné, aby vlákna ve warpu přistupovala k paměti po souvislých částech. Podrobný popis přístupu do globální paměti i všech ostatních vlastností je v [4].

3.2 Výkon

CUDA dosahuje velkého výpočetního výkonu oproti CPU tím, že nemá komplikované řízení běhu pro jedno vlákno ani velkou vyrovnávací paměť. Ušetřené tranzistory místo toho používá na multiprocesory. Řízení běhu v CPU je proto, aby se zamaskovala latence hlavní paměti. CUDA na rozdíl od CPU maskuje latenci přepínáním warpů a bloků, takže je potřeba mít řádově víc aktivních vláken než procesorů.

Díky rozložení výpočtu na výpočetně nezávislé bloky dochází k lineárnímu škálování výkonu s počtem multiprocesorů, jestliže je vstupní instance řešeného problému dostatečně velká. Nevýhoda tohoto přístupu je, že během výpočtu nejde synchronizovat mezi bloky. Tuto nevýhodu zjemňují atomické instrukce. Bloky můžou například společně zvyšovat nějaký čítač. Obvyklý způsob synchronizace mezi bloky je používat ukončení výpočtu kernelu jako synchronizační bariéru. Tedy rozdělit výpočet před a po synchronizaci do dvou kernelů.

Kapitola 4

Tridiagonální systém lineárních rovnic

V této kapitole jsou popsané tři algoritmy pro řešení tridiagonálního systému lineárních rovnic. Thomasův algoritmus (*TA*), dvousměrná Gaussova eliminace (*DGE*) a dělící algoritmus, který používá paralelní cyklickou redukci (*DA*). Originální anglické názvy jsou *Thomas algorithm*, *two–way factorization algorithm*, *partition algorithm* a *paralel cyclic reduction*. Všechny tři algoritmy hledají přesné řešení, takže odpadá problém s volbou ukončovací podmínky u iteračních metod.

Řešením tridiagonálního systému lineárních rovnic v modelu CUDA se zabývali autoři v [5]. Používali cyklickou redukci, paralelní cyklickou redukci, rekurzivní dvojení (*recursive doubling*) a některé jejich kombinace. Jejich řešení ale předpokládá, že počet rovnice je menší roven 512, protože jejich postup je omezen velikostí sdílené paměti. Zmiňují se, že jejich řešení podporuje i větší počet rovnic, když použijí globální místo sdílené paměti. Tato změna jim ale způsobila trojnásobné zpomalení. Matice

$$B = \begin{pmatrix} \alpha_0 & \beta_0 & 0 & & \\ \gamma_0 & \alpha_1 & \beta_1 & & \\ & \ddots & \ddots & \ddots & \\ & & \gamma_{N-3} & \alpha_{N-2} & \beta_{N-2} \\ & & & \gamma_{N-2} & \alpha_{N-1} \end{pmatrix} \tag{4.1}$$

dále v textu označuje systém lineárních rovnic, který má N rovnic. Neznámé jsou označené vektorem u a hodnoty na pravé straně jsou označené vektorem d. Toto označení je konzistentní s [1]. Matice B, která vznikne z částečně implicitního schématu, je speciální tím, že jde zkonstruovat pouze z jednoho pole g, které má délku N. Tato vlastnost snižuje nároky na paměťovou propustnost. Předchozí vlastnost a potřeba řešit větší systémy než 512 rovnic byly motivací pro hledání jiných algoritmů.

4.1 Thomasův algoritmus

Optimální způsob k vyřešení tridiagonálního systému lineárních rovnic na sériovém stroji je Thomasův algoritmus (*TA*) [1]. Thomasův algoritmus je pouze speciální případ Gaussovy eliminace. V následujících pseudokódech označuje l_i koeficient, který se použije k vynásobení řádku i před přičtením k řádku $i + 1$. Prvek m_i označuje novou hodnota na diagonále, y_i označuje novou hodnotu na pravé straně a u_i označuje výsledek. Thomasův algoritmus jde rozložit na dvě fáze. V první fázi, kterou popisuje pseudokód 1, postupuje algoritmus shora dolů a nuluje diagonálu pod hlavní diagonálou. V druhé fázi, kterou popisuje pseudokód 2, postupuje algoritmus zdola nahoru. Při tomto kroku nuluje diagonálu nad hlavní diagonálou a počítá konečné řešení. Délky pomocných vektorů jsou následující. Velikost vektorů y a β je N. Velikost vektoru β je $N - 1$. Počet potřebných kroků je $2N$.

Algorithm 1 Dopředná fáze

$m_0 \leftarrow \alpha_0$
$l_0 \leftarrow \gamma_0 / m_0$
$m_1 \leftarrow \alpha_1 - l_0 \beta_0$
$y_0 \leftarrow d_0$
for $i = 1$ **to** $N - 3$ **do**
  $y_i \leftarrow d_i - l_{i-1} y_{i-1}$
  $l_i \leftarrow \gamma_i / m_i$
  $m_{i+1} \leftarrow \alpha_{i+1} - l_i \beta_i$
end for
$y_{N-2} \leftarrow d_{N-2} - l_{N-3} y_{N-3}$
$l_{N-2} \leftarrow \gamma_{N-2} / m_{N-2}$
$m_{N-1} \leftarrow \alpha_{N-1} - l_{N-2} \beta_{N-2}$
$y_{N-1} \leftarrow d_{N-1} - l_{N-2} y_{N-2}$

Algorithm 2 Zpětná fáze

$u_{N-1} \leftarrow y_{N-1} / m_{N-1}$
for $i = N - 2$ **to** 0 **do**
  $u_i \leftarrow (y_i - \beta_i u_{i+1}) / m_i$
end for

4.2 Dvousměrná Gaussova eliminace

Dvousměrná Gaussova eliminace je úprava obyčejné Gaussovy eliminace pro dva procesory [6] a také jde rozložit do dvou fází. V první fázi první procesor postupuje maticí směrem shora dolů, zatímco druhý procesor postupuje zdola nahoru. Uprostřed matice na sebe počkají a vymění si data. První fáze je popsaná pro oba procesory zvlášť v pseudokódech 3 a 4. Druhá fáze se skládá ze dvou kroků. V prvním kroku druhé fáze použijí oba procesory získaná data k vyřešení dvou lineárních rovnic o dvou neznámých. V druhém kroku první procesor postupuje maticí nahoru k začátku matice, zatímco druhý procesor postupuje dolů ke konci matice. Druhá fáze je popsaná pro oba procesory zvlášť v pseudokódech 5 a 6. Střed matice B má při výměně dat tvar

$$B = \begin{pmatrix} 0 & m_{q-2} & \beta_{q-2} & & & \\ & 0 & m_{q-1} & \beta_{q-1} & & \\ & & \gamma_{q-1} & m_q & 0 & \\ & & & \gamma_q & m_{q+1} & 0 \end{pmatrix}. \tag{4.2}$$

Hodnota q se získá jako $\lfloor N/2 \rfloor$.
Délky pomocných vektorů jsou následující. Velikost vektorů y a m je N. Velikost vektoru β je $N - 1$. Tento algoritmus dokáže výpočet zrychlit maximálně pouze dvakrát oproti TA. Cena za toto zrychlení je ale minimální. Pouze se navíc řeší dvě lineární rovnice o dvou neznámých. Počet potřebných kroků je N.

Algorithm 3 Dopředná fáze, procesor 1

$m_0 \leftarrow \alpha_0$
$l_0 \leftarrow \gamma_0 / m_0$
$m_1 \leftarrow \alpha_1 - l_0 \beta_0$
$y_0 \leftarrow d_0$
for $i = 1$ **to** $q - 2$ **do**
  $y_i \leftarrow d_i - l_{i-1} y_{i-1}$
  $l_i \leftarrow \gamma_i / m_i$
  $m_{i+1} \leftarrow \alpha_{i+1} - l_i \beta_i$
end for
$y_{q-1} \leftarrow d_{q-1} - l_{q-2} y_{q-2}$

Algorithm 4 Dopředná fáze, procesor 2

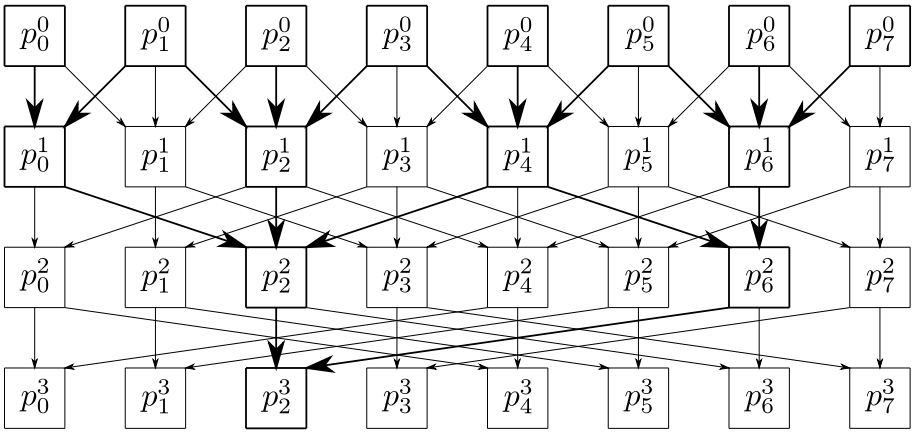
$m_{N-1} \leftarrow \alpha_{N-1}$
$l_{N-1} \leftarrow \beta_{N-2} / m_{N-1}$
$m_{N-2} \leftarrow \alpha_{N-2} - l_{N-1} \gamma_{N-2}$
$y_{N-1} \leftarrow d_{N-1}$
for $i = N - 2$ **to** $q + 1$ **do**
  $y_i \leftarrow d_i - l_{i+1} y_{i+1}$
  $l_i \leftarrow \beta_{i-1} / m_i$
  $m_{i-1} \leftarrow \alpha_{i-1} - l_i \gamma_{i-1}$
end for
$y_q \leftarrow d_q - l_{q+1} y_{q+1}$

Algorithm 5 Zpětná fáze, procesor 1

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} m_{q-1} & \beta_{q-1} \\ \beta_{q-1} & m_q \end{pmatrix}^{-1} \begin{pmatrix} y_{q-1} \\ y_q \end{pmatrix}$$
$u_{q-1} \leftarrow x$
for $i = q - 2$ **to** 0 **do**
  $u_i \leftarrow (y_i - \beta_i u_{i+1}) / m_i$
end for

Algorithm 6 Zpětná fáze, procesor 2

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} m_{q-1} & \beta_{q-1} \\ \beta_{q-1} & m_q \end{pmatrix}^{-1} \begin{pmatrix} y_{q-1} \\ y_q \end{pmatrix}$$
$u_q \leftarrow y$
for $i = q + 1$ **to** $N - 1$ **do**
  $u_i \leftarrow (y_i - \gamma_{i-1} u_{i-1}) / m_i$
end for



Obrázek 4.1: příklad paralelní cyklické redukce

4.3 Paralelní cyklická redukce

Cyklická redukce [5]v každém kroku sníží počet rovnic i neznámých na polovinu, dokud nezůstane pouze jedna rovnice o jedné neznámé. CR postupuje tak, že k rovnici i přičte rovnici $i+1$ vynásobenou $-\beta_i/\alpha_{i+1}$ a rovnici $i-1$ vynásobenou $-\gamma_{i-1}/\alpha_{i-1}$. Tím získá rovnici, která nezávisí na u_{i-1} a u_{i+1}. Rovnice i se sousedy vypadá před touto úpravou jako

$$\begin{pmatrix} \gamma_{i-2} & \alpha_{i-1} & \beta_{i-1} & & \\ & \gamma_{i-1} & \alpha_i & \beta_i & \\ & & \gamma_i & \alpha_{i+1} & \beta_{i+1} \end{pmatrix} = \begin{pmatrix} d_{i-1} \\ d_i \\ d_{i+1} \end{pmatrix} \cdot \begin{matrix} -\gamma_{i-1}/\alpha_{i-1} & & \\ & & \\ & & -\beta_i/\alpha_{i+1} \end{matrix} . \tag{4.3}$$

CR může například odstraňovat systematicky všechny rovnice, které jsou sudé v pořadí. Zpětnou substitucí následně zjistí všechny ostatní hodnoty neznámých. Cyklická redukce potřebuje k výpočtu $2\log_2(N)$ kroků.

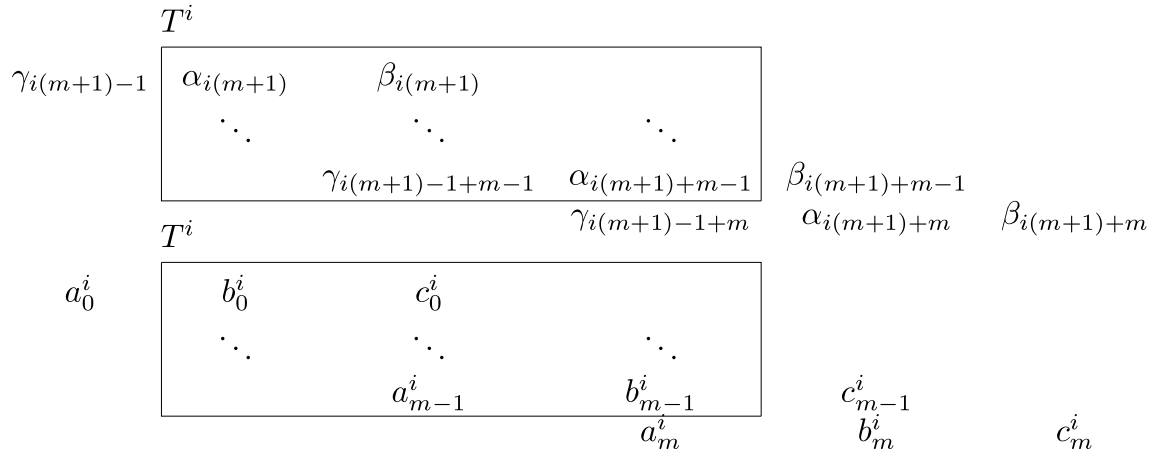
Paralelní cyklická redukce dělá stejné operace jako cyklická redukce, ale v každém kroku generuje dva poloviční systémy lineárních rovnic. Jeden systém obsahuje pouze sudé a druhý liché proměnné z upravovaného systému. Tímto způsobem získá PCR hodnoty všech proměnných už po $log_2(N)$ krocích. Na obrázku 4.1 je ukázka postupu algoritmu PCR. Je na něm také zvýrazněná cesta, která vede k řešení jedné proměnné.

PCR nemá implementační kapitolu, protože v [7] je napsaná kompletní implementace pro CUDA. Tahle implementace je jenom upravená, aby byla schopná najednou vyřešit tři různé systémy lineárních rovnic, které se liší pouze vektorem d.

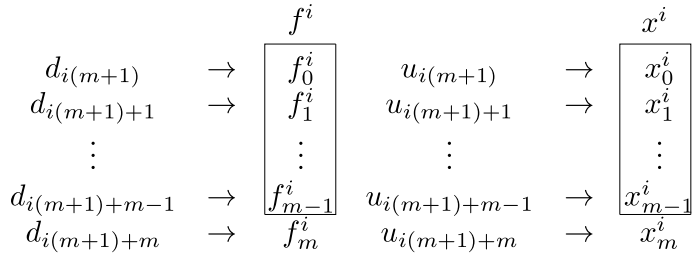
4.4 Dělící algoritmus

Dělící algoritmus (DA), který byl publikovaný v [6], řeší systém lineárních rovnic pomocí P procesorů. DA se dělí do tří fází. V první a poslední fázi počítají procesory nezávisle nad různými částmi matice B. V druhé fázi probíhá výměna dat a ve třetí fází se vypočítá konečné řešení. DA rozděluje matici B na P podmatic T^i, které jsou vždy oddělené jedním řádkem. Pro jednoduchost se v této kapitole předpokládá, že existuje m, které splňuje $N = mP+P-1$. Na obrázku 4.2 a 4.3 je zavedené značení, které se používá v této kapitole. Nyní lze každému procesoru přiřadit k řešení

$$a_0^i e_0 x_m^{i-1} + T^i x^i + c_{m-1}^i e_{m-1} x_m^i = f^i. \tag{4.4}$$



Obrázek 4.2: označení prvků matice B



Obrázek 4.3: označení vektorů

Vektor e_i je jednotkový vektor, ve kterém je jednička na pozici i. Rovnice, které leží mezi T_i, mají v novém značení tvar

$$a_m^i x_{m-1}^i + b_m^i x_m^i + c_m^i x_0^{i+1} = f_m^i. \tag{4.5}$$

Po úpravě rovnice (4.4) a vyjádření x^i vznikne

$$\begin{aligned} x^i &= w^i - r^i x_m^{i-1} - s^i x_m^i && (4.6)\\ w^i &= (T^i)^{-1} f^i && (4.7)\\ r^i &= a_0^i (T^i)^{-1} e_0 && (4.8)\\ s^i &= c_{m-1}^i (T^i)^{-1} e_{m-1}. && (4.9) \end{aligned}$$

K získání w, r, s je potřeba, aby každý procesor invertoval svou matici T^i. Tohle je výpočetně nejnáročnější operace v tomto algoritmu, ale každý procesor ji provádí pouze na svých lokálních datech a nezávisle na ostatních procesorech. Výpočet w, r, s tvoří první fázi. Po dosazení za proměnné x_{m-1}^i a x_0^{i+1} do rovnice (4.5) vznikne

$$a_m^i (w_{m-1}^i - r_{m-1}^i x_m^{i-1} - s_{m-1}^i x_m^i) + b_m^i x_m^i + c_m^i (w_0^{i+1} - r_0^{i+1} x_m^i - s_0^{i+1} x_m^{i+1}) = f^i. \tag{4.10}$$

Po úpravě této rovnice vznikne

$$\begin{aligned} (-a_m^i r_{m-1}^i) x_m^{i-1} + (b_m^i - a_m^i s_{m-1}^i - c_m^i r_0^{i+1}) x_m^i + (-c_m^i s_0^{i+1}) x_m^{i+1} = \\ f_m^i - a_m^i w_{m-1}^i - c_m^i w_0^{i+1}. \end{aligned} \tag{4.11}$$

Po dosazení za i vznikne tridiagonální systém lineárních rovnic R o $P-1$ rovnicích. Po naplnění matice R jeden procesor vypočítá hodnoty x_m^i pro všechna $i \in \{0, 1, \ldots P-2\}$. Tento výpočet tvoří druhou fázi. Po výpočtu neznámých x_m^i pro mezilehlé rovnice můžou všechny procesory paralelně vypočítat konečné hodnoty neznámých podle rovnice (4.6). Celkový počet kroků závisí na tom, který algoritmus se použije pro invertování T^i.

Kapitola 5

CUDA implementace 2–D difúze

V této kapitole je popsaná CUDA implementace algoritmů z předchozí kapitoly. Jejich výkon v CUDA je porovnaný při výpočtu 2–D izotropní difúze. Vítězná implementace je použitá v podkapitolách lineární a anizotropní difúze. U každé implementace je popsané seskupení vláken do bloků a přiřazení vláken k výpočetním problémům. Všechny naměřené a vypočítané informace jsou zapsané do jedné tabulky. Ke každé implementaci je přiřazený plán, který znázorňuje obsazení paměti GPU daty a procedury, které tato data transformují.

Z plánu jsou vypočítaná dvě teoretická časová minima, protože se předpokládá, že výkon je omezený propustností datové sběrnice. Jedno určuje minimální čas, který potřebují samotné algoritmy při řešení difúze na 2–D datech. Druhé určuje minimální čas, který určuje celkovou rychlost výpočtu. Každý minimální čas se skládá ze dvou časů. První je vypočítaný z teoretické propustnosti sběrnice, druhý je vypočítaný z naměřené propustnosti a je uzávorkovaný. V celkové rychlosti je zahrnutý výpočet difúzních koeficientů, kopírování, transpozice, vážené součty.

Implementace algoritmu pro transpozici dat je převzatý a upravený z [8]. Implementace algoritmu pro výpočet difúzních koeficientů pouze využívá sdílené paměti na načtení rozmazaných dat. Každé vlákno načte příslušnou hodnotu a vlákna, která nejsou okrajová, vypočítají difúzní koeficient a uloží ho zpět do globální paměti. Oddělením výpočtu difúzních koeficientů od řešení tridiagonálního systému dojde k zjednodušení a zpřehlednění kódu pouze za cenu dvou nových přístupů do globální paměti. Po sečtení všech přenosů dat do a z do globální paměti (S) je možné minimální časy určit z rovnice

$$t = S/b. \tag{5.1}$$

Propustnost datové sběrnice určuje rovnice

$$b = f \cdot w. \tag{5.2}$$

Hodnota f je frekvence a w je šířka sběrnice. Také je možné propustnost datové sběrnice změřit například pomocí funkce cuMemcpyDtoD().

Následně je naměřený čas, který potřebuje samotný algoritmus a celkový čas výpočtu. Měření času samotného algoritmu je implementované tak, že samotný algoritmus je spuštěný při splnění podmínky $\tau > 0$. Čas samotného algoritmu vznikne po odečtení času získaného s $\tau = 0$ od celkového času s $\tau > 0$. Poměr minimálního času a naměřeného času určuje užitečné využití propustnosti datové sběrnice. Počet iterací u každého vstupu je zvolený tak, aby objem přenesených dat byl stejný pro všechny vstupy. Tato volba zjednodušuje výpočet využití propustnosti, protože vypočtená minima jsou stejná pro všechny vstupní velikosti dat. Také to má tu výhodu, že z konstantního času potřebného na výpočet při různě velkých datech plyne lineární závislost potřebného času na výpočet v závislosti na velikosti vstupu. Celkový čas je porovnaný s časem, který potřebuje implementace pro CPU, která používá

	GT 220	GTX 470
počet multiprocesorů	6	14
počet výpočetních jednotek na MP	8	32
šířka sběrnice [bit]	128	320
frekvence pamětí [Mhz]	810	3348
teoretická prop. [GB/s]	12.07	124.72
změřená prop. [GB/s]	10.72	93.02

Tabulka 5.1: paramentry grafických karet

pouze jedno jádro. Poměr mezi počtem bloků a počtem multiprocesorů by měl být větší než jedna, protože jinak některý multiprocesor nebude během výpočtu vůbec nic dělat. Kvůli maskování latence globální paměti by měl být poměr vyšší. Maximální počet bloků, mezi kterými se může přepínat, je 8. Vyšší poměr než osm má smysl v tom, že ve větší části výpočtu je možné přepínat mezi osmi bloky. Pro vlastní testování jsou použité dva počítače, na které se dále bude odkazovat jako na COMP1, COMP2. COMP1 obsahuje procesor Intel Pentium Dual–Core E5300 @ 2.6Ghz. Frekvence jeho sběrnice je 800Mhz. COMP2 obsahuje procesor Intel Core 2 Q6600 @ 2.4Ghz. Frekvence jeho sběrnice je 1066Mhz. COMP1 obsahuje grafickou kartu Nvidia GT 220 a COMP2 obsahuje grafickou kartu Nvidia GTX 470. Potřebné parametry karet jsou v tabulce 5.1. U izotropní difúze jsou také porovnané výsledky výpočtu všech implementací pro GPU s implementací Thomasova algoritmu pro CPU. Pro porovnání se vypočítá maximální relativní chyba mezi oběma výsledky. Relativní chyba se vypočítá jako

$$err_r = \frac{|x - y|}{|x|}. \tag{5.3}$$

Tato chyba se mění v závislosti na počtu iterací, velikosti strany dat, velikosti τ a velikosti λ. Pro zjednodušení se vždy bere velikost strany 1024 a $\lambda = 5$, aby bylo možné výsledky zapsat do tabulky. Pro anizotropní difúze je obtížné najít nějaké smysluplné porovnání pomocí jednoho čísla, protože dochází k větvením kódu v podmínkách, které mají tvar $x > \varepsilon$. Díky tomu mohou například vznikat výsledky, které se liší hodně, ale pouze v pár pixelech.

5.1 Izotropní difúze

5.1.1 Thomasův algoritmus

Každé vlákno dostane na starost jeden sloupec v datech. Pro zarovnaný přístup do globální paměti stačí, aby implementace TA pro izotropní difúzi (ITA) pouze přepočítávala paměťové adresy tak, aby vlákno i přistupovalo na sloupec i. Pomocná data y, β, m implementace ukládá do pole, které má stejné rozměry jako vstupní data. Pro zpracování řádků stačí vstupní data transponovat. Vlákna mezi sebou nekomunikují, takže je možné zvolit velikost bloku téměř libovolně. Nevýhoda ITA je ale ta, že pro čtvercový vstupní obrázek s N^2 pixely je potřeba pouze N vláken. Kvůli tomu je velikost bloku v testech nastavená pouze na 64. Při použití této velikosti bloku by měla mít vstupní data na COMP1 velikost hrany alespoň 3072 $(64 \cdot 6 \cdot 8)$, aby se plně využilo přepínání bloků. Na COMP2 to je už alespoň 7168 $(64 \cdot 14 \cdot 8)$. V tabulce 5.2 je zachycený plán použití paměti a v tabulce 5.3 jeho popisky. V tabulkách 5.4 a 5.5 jsou zachycené výsledky měření. ITA k řešení na COMP1 potřebuje minimálně 1877

0	1	2	3	4	5	6	7	8	9
im	im					im_1			im
	$im^\top$							im_2	
	im_σ	im_σ			β		β		
	im_σ	im_σ			m		m		
			g						
				$g^\top$					
					y		y		

Tabulka 5.2: plán při použití ITA

přechod	popis	přechod	popis
$0 \to 1$	kopírování, transpozice (4rw)	$5 \to 6$	zpětná fáze (4rw)
$1 \to 2$	lin. difúze (-rw)	$6 \to 7$	dopředná fáze (5rw)
$2 \to 3$	difúzní koeficienty (2rw)	$7 \to 8$	zpětná fáze (4rw)
$3 \to 4$	transpozice (2rw)	$8 \to 9$	průměr (3rw)
$4 \to 5$	dopředná fáze (5rw)		

Tabulka 5.3: popisky plánu

(2113)ms. Z toho 1165 (1311)ms zabírají vlastní výpočty. ITA k řešení na COMP2 potřebuje minimálně 182 (244)ms. Z toho 113 (151)ms zabírají vlastní výpočty. Stejná minima mají i IDGE a IDGE2. V tabulce 5.6 jsou zachycené maximální relativní chyby. Po řádcích se mění počet iterací a po sloupcích se mění velikost τ.

5.1.2 Dvousměrná Gaussova eliminace

Implementace Gaussovy eliminace pro izotropní difúzi (IDGE) má stejné přiřazení vláken datům, způsob přístupu k paměti, plán přístupu k paměti, pomocná data a velikost bloku jako ITA. Liší se ale tím, že na zpracování vstupních dat o velikosti N^2 použije $2N$ vláken. Teoreticky by mělo být možné dosáhnout až dvojnásobného zrychlení oproti ITA, jestliže by byla GPU při použití IDGE maximálně vytížená a při použití ITA by byla vytížená jenom

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
CPU čas[ms]	57346	76830	71448	79155	79342	79139	78499
GPU čas[ms]	6506	4384	4071	3963	4103	3936	3946
zrychlení	8.81	17.53	17.55	19.97	19.34	20.11	19.89
využitá teor. prop.	0.29	0.43	0.46	0.47	0.46	0.48	0.48
využitá prop. memcpy	0.32	0.48	0.52	0.53	0.51	0.54	0.54
GPU čas TA[ms]	4634	2933	2698	2634	2777	2657	2667
využitá teor. prop. alg.	0.25	0.40	0.43	0.44	0.42	0.44	0.44
využitá prop. memcpy alg.	0.28	0.45	0.49	0.50	0.47	0.49	0.49
obsazení MP	0.67	1.33	2	2.67	4	5.23	8

Tabulka 5.4: ITA, hodnoty naměřené na COMP1

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
CPU čas[ms]	36720	37570	38650	54690	54330	55590	55310
GPU čas[ms]	2370	1330	970	780	600	520	440
zrychlení	15.49	28.25	39.38	70.12	90.55	106.90	125.70
využitá teor. prop.	0.08	0.14	0.19	0.23	0.30	0.35	0.41
využitá prop. memcpy	0.10	0.18	0.25	0.31	0.41	0.47	0.55
GPU čas TA[ms]	2100	1100	760	580	400	330	260
využitá teor. prop. alg.	0.05	0.10	0.15	0.19	0.28	0.34	0.43
využitá prop. memcpy alg.	0.07	0.14	0.20	0.26	0.38	0.46	0.58
obsazení MP	0.29	0.57	0.86	1.14	1.71	2.29	3.43

Tabulka 5.5: ITA, hodnoty naměřené na COMP2

$\times 10^{-3}$	1.0	3.0	9.0	$\times 10^{-3}$	1.0	3.0	9.0
10	0.002	0.003	0.017	10	0.001	0.002	0.003
30	0.006	0.017	0.364	30	0.003	0.004	0.017
50	0.009	0.072	1.201	50	0.004	0.009	0.059

Tabulka 5.6: ITA, vlevo max. realativní chyby naměřené na COMP1, vpravo na COMP2

z poloviny. Tohoto okrajového případu se v testech nepodařilo dosáhnout. Výsledky testů jsou v tabulkách 5.7 a 5.8. V tabulce 5.9 jsou zachycené maximální relativní chyby. Po řádcích se mění počet iterací a po sloupcích se mění velikost τ.

Za cenu zvýšení paměťových nároků je možné najednou zpracovávat řádky i sloupce (IDGE2). IDGE2 už na zpracování vstupních dat o velikosti N^2 použije $4N$ vláken. V tomto případě se ale liší plán přístupu k paměti od plánu ITA. Plán pro IDGE2 je v tabulce 5.10 a výsledky měření v tabulkách 5.12 a 5.13. Z měření plyne, že k největšímu navýšení výkonu oproti IDGE dochází u malých dat. To je očekávaný výsledek, protože malá data zpracovává malý počet vláken, která nedokáží plně využít GPU. Popisky k plánu 5.10 jsou v tabulce 5.11. V tabulce 5.14 jsou zachycené maximální relativní chyby. Po řádcích se mění počet iterací a po sloupcích se mění velikost τ.

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
GPU čas[ms]	3869	2839	2699	2621	2668	2605	2605
zrychlení	14.82	27.06	26.47	30.20	29.74	30.38	30.13
využitá teor. prop.	0.49	0.66	0.70	0.72	0.70	0.72	0.72
využitá prop. memcpy	0.55	0.74	0.78	0.81	0.79	0.81	0.81
GPU čas IDGE[ms]	2012	1388	1326	1310	1342	1310	1310
využitá teor. prop. alg.	0.58	0.84	0.88	0.89	0.87	0.89	0.89
využitá prop. memcpy alg.	0.65	0.94	0.99	1.00	0.98	1.00	1.00
obsazení MP	1.33	2.67	4	5.33	8	10.67	16

Tabulka 5.7: IDGE, hodnoty naměřené na COMP1

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
GPU čas[ms]	1670	970	720	600	480	410	360
zrychlení	21.99	38.73	53.68	91.15	113.19	135.59	153.64
využitá teor. prop.	0.11	0.19	0.25	0.30	0.38	0.44	0.51
využitá prop. memcpy	0.15	0.25	0.34	0.41	0.51	0.60	0.68
GPU čas IDGE[ms]	1400	740	510	400	290	220	170
využitá teor. prop. alg.	0.07	0.12	0.16	0.19	0.24	0.28	0.31
využitá prop. memcpy alg.	0.09	0.16	0.21	0.25	0.31	0.37	0.42
obsazení MP	0.57	1.14	1.71	2.29	3.43	4.57	6.86

Tabulka 5.8: IDGE, hodnoty naměřené na COMP2

$\times 10^{-3}$	1.0	3.0	9.0	$\times 10^{-3}$	1.0	3.0	9.0
10	0.002	0.004	0.007	10	0.001	0.002	0.004
30	0.006	0.012	0.054	30	0.003	0.003	0.013
50	0.009	0.052	0.202	50	0.004	0.011	0.039

Tabulka 5.9: IDGE, vlevo max. realativní chyby naměřené na COMP1, vpravo na COMP2

0	1	2	3	4	5	6	7
im	im					im	im
	$im^\top$					$im^\top$	
	im_σ	im_σ			β		
	$im_\sigma^\top$	$im_\sigma^\top$			m		
			g				
				$g^\top$			
					y		
					$y^\top$		
					$\beta^\top$		
					$m^\top$		

Tabulka 5.10: plán při použití IDGE2

přechod	popis	přechod	popis
$0 \to 1$	kopírování, transpozice (4rw)	$5 \to 6$	zpětná fáze (8rw)
$1 \to 2$	lin. difúze (-rw)	$6 \to 7$	průměr (2rw)
$2 \to 3$	difúzní koeficienty (2rw)		
$3 \to 4$	transpozice (2rw)		
$4 \to 5$	dopředná fáze (10rw)		

Tabulka 5.11: popisky plánu pro IDGE2

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
GPU čas[ms]	3385	2808	2839	2652	2699	2605	2621
zrychlení	16.94	27.36	25.17	29.85	29.40	30.38	29.95
využitá teor. prop.	0.55	0.67	0.66	0.71	0.70	0.72	0.72
využitá prop. memcpy	0.62	0.75	0.74	0.80	0.78	0.81	0.81
GPU čas IDGE2[ms]	1451	1326	1466	1326	1388	1326	1326
využitá teor. prop. alg.	0.80	0.88	0.79	0.88	0.84	0.88	0.88
využitá prop. memcpy alg.	0.90	0.99	0.89	0.99	0.94	0.99	0.99
obsazení MP	2.7	5.3	8	10.7	24	21.3	32

Tabulka 5.12: IDGE2, hodnoty naměřené na COMP1

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
GPU čas[ms]	1100	620	490	440	370	410	370
zrychlení	33.38	60.60	78.80	124.30	146.84	135.59	149.49
využitá teor. prop.	0.18	0.29	0.37	0.41	0.49	0.44	0.49
využitá prop. memcpy	0.24	0.39	0.50	0.55	0.66	0.60	0.66
GPU čas IDGE2[ms]	740	400	290	240	170	220	180
využitá teor. prop. alg.	0.15	0.28	0.38	0.47	0.66	0.51	0.63
využitá prop. memcpy alg.	0.20	0.38	0.52	0.63	0.89	0.69	0.84
obsazení MP	1.14	2.29	3.43	4.57	6.86	9.14	13.71

Tabulka 5.13: IDGE2, hodnoty naměřené na COMP2

$\times 10^{-3}$	1.0	3.0	9.0	$\times 10^{-3}$	1.0	3.0	9.0
10	0.002	0.004	0.007	10	0.001	0.002	0.004
30	0.006	0.012	0.054	30	0.003	0.003	0.013
50	0.009	0.052	0.202	50	0.004	0.011	0.039

Tabulka 5.14: IDGE2, vlevo max. realativní chyby naměřené na COMP1, vpravo na COMP2

0	1	2	3	4	5	6	7
im	im				im_1		im
	$im^\top$					im_2	
	im_σ	im_σ		$g^\top$			
	$im_\sigma^\top$	$im_\sigma^\top$	g				

Tabulka 5.15: plán při použití IDA

přechod	popis	přechod	popis
$0 \to 1$	kopírování, transpozice (4rw)	$5 \to 6$	dělící algoritmus (5rw)
$1 \to 2$	lin. difúze (-rw)	$6 \to 7$	průměr (2rw)
$2 \to 3$	difúzní koeficienty (2rw)		
$3 \to 4$	transpozice (2rw)		
$4 \to 5$	dělící algoritmus (5rw)		

Tabulka 5.16: popisky plánu pro IDA

5.1.3 Dělící algoritmus

Implementace předchozích dvou algoritmů má dva problémy. První je, že potřebují hodně paměťových operací. Druhý je, že k řešení je potřeba málo bloků vláken. Implementace DA pro izotropní difúzi (IDA) se snaží oba problémy řešit za cenu zvýšené aritmetické náročnosti. IDA používá k řešení $(T^i)^{-1}$ PCR, aby využila SIMD povahu multiprocesoru a minimalizoval se čas na řešení části, které se budou řešit dvakrát. IDA totiž v první fázi neukládá vektory w, r, s, ale v třetí fázi je počítá znovu. Neukládá je kvůli omezené velikosti sdílené paměti. Každý blok vláken řeší jeden řádek dat. Pro zarovnaný přístup do globální paměti stačí, aby IDA pouze přepočítávala paměťové adresy tak, aby vlákna z bloku i přistupovala na řádek i. IDA používá také bloky o velikosti 64 vláken. Díky vlastnostem PCR je velikost T^i také 64. To znamená, že IDA postupně hledá jednotlivé w^i, r^i, s^i a pomocí nich vyplní vždy jeden řádek v R. Malá velikost znamená rychlejší vyřešení T^i, ale klade větší paměťové i výpočetní nároky na řešení R. IDA má pevně zvolenou velikost R na 64. Z toho plyne omezení na maximální velikost vstupních dat na $65 \cdot 64 + 64 = 4224$.

V tabulce 5.15 je zachycený plán použití paměti a tabulce 5.16 jeho popisky. V tabulkách 5.17 a 5.18 jsou výsledky měření. U téhle implementace se nepodařilo dosáhnout úspěchu a vyřešit problémy předchozích implementací. Při prvním pohledu na výsledky je zřejmé, že by bylo zbytečné počítat minima a využití sběrnice. V tabulce 5.19 jsou zachycené maximální relativní chyby. Po řádcích se mění počet iterací a po sloupcích se mění velikost τ.

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
GPU čas[ms]	14680	12964	12511	12412	12184	12028	11841
zrychlení	3.91	5.93	5.71	6.38	6.51	6.58	6.63
GPU čas IDA[ms]	12824	11513	11138	11070	10874	10748	10547
obsazení MP	42.67	85.33	128	170.67	256	341.33	512

Tabulka 5.17: IDA, hodnoty naměřené na COMP1

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
GPU čas[ms]	2160	1800	1730	1700	1670	1650	1610
zrychlení	17.0	20.87	22.34	32.17	32.53	33.69	34.35
GPU čas IDA[ms]	1890	1580	1520	1500	1470	1460	1420
obsazení MP	18.29	36.57	54.86	73.14	109.71	146.29	219.43

Tabulka 5.18: IDA, hodnoty naměřené na COMP2

$\times 10^{-3}$	1.0	3.0	9.0	$\times 10^{-3}$	1.0	3.0	9.0
10	0.001	0.004	0.016	10	0.001	0.003	0.005
30	0.003	0.017	0.293	30	0.004	0.010	0.015
50	0.006	0.070	0.913	50	0.008	0.036	0.037

Tabulka 5.19: IDA, vlevo max. realativní chyby naměřené na COMP1, vpravo na COMP2

5.1.4 Porovnání implementací

V tabulce 5.20 jsou pro porovnání vypsaná zrychlení pro data, která mají velikost 1024×1024. Z porovnání plyne, že nejlepších výsledků dosahuje IDGE2. Druhé nejlepší výsledky dosahuje IDGE. IDGE2 má také výhodu v tom, že při zpracování dat, která mají rozdílný počet řádků a sloupců, je počet pracujících vláken roven součtu počtu řádků a sloupců. IDGE v tomto případě má v prvním kroku počet pracujících vláken roven počtu sloupců a v druhém kroku má počet pracujících vláken roven počtu řádků. Při velkém nepoměru počtu řádků a sloupců nebude IDGE schopné využít plně zdroje GPU.

5.2 Lineární difúze

U lineární difúze se dá s výhodou použít vlastnost, že vektory l, m jsou pro všechny řádky a sloupce stejné. Při použití DGE jsou tyto vektory navíc symetrické. Implementace DGE pro lineární difúzi (LDGE2) díky těmto vlastnostem může vektory m, l předpočítat a uložit do paměti konstant vždy jen polovinu vektoru. Pro výpočet vektoru α, β, γ není potřeba přenášet g, protože g má všechny prvky rovny jedné. Tyto vlastnosti zmenšují nároky na propustnost globální paměti. LDGE2 má stejné přiřazení vláken datům a způsob přístupu do paměti jako IDGE2. LDGE2 tedy na zpracování vstupních dat o velikosti $O(N^2)$ použije 4N vláken. V tabulce 5.21 je plán použití paměti a v tabulce 5.22 jeho popisky. V tabulkách 5.23 a 5.24 jsou zachycené výsledky měření. LDGE2 k řešení na COMP1 potřebuje minimálně 777 (875)ms. Z toho 518 (583)ms zabírají vlastní výpočty. LDGE2 k řešení na COMP2 potřebuje

algoritmus	zrychlení, COMP1	zrychlení COMP2
ITA	19.97	39.38
IDGE	30.20	91.15
IDGE2	29.85	124.30
IDA	6.38	32.17

Tabulka 5.20: porovnání zrychlení pro velikost 1024

0	1	2	3
im		im_1	im
$im^\top$		$im_2^\top$	$im^\top$
	y		
	$y^\top$		

Tabulka 5.21: plán při použití LDGE2

přechod	popis	přechod	popis
$0 \to 1$	dopředná fáze (4rw)	$2 \to 3$	průměr, transpozice (4rw)
$1 \to 2$	zpětná fáze (4rw)		

Tabulka 5.22: popisky plánu pro LDGE2

minimálně 75 (101)ms. Z toho 50 (67)ms zabírají vlastní výpočty.

5.3 Anizotropní difúze

Implementace DGE pro anizotropní difúzi (ADGE) má stejné přiřazení vláken datům a přístup do paměti, jestliže se zpracovávají směry, které jsou rovnoběžné s osami. ADGE přiřadí každé vlákno jedné diagonále, jestliže zpracovává jeden z diagonálních směrů. ADGE neprovádí žádné výpočty na jednoprvkových diagonálách (rozích). Tyto rohy pouze zkopíruje do výsledku. Celkový počet vláken při zpracování diagonálního směru je tedy $h - 2 + w - h + 1 + h - 2$. Význam proměnných je znázorněný na obrázku 5.1.

Hlavní problém ADGE je fakt, že je potřeba postupovat po diagonálách vstupních dat. Pro zjednodušení ADGE předpokládá, že šířka vstupních dat je větší než výška. ADGE rozděluje přístup po diagonálách na šest případů. Dále je popsaný pouze jeden případ, protože ostatní se odvodí obdobně. Na obrázku 5.1 je znázorněný případ, ve kterém ADGE zpracovává levý spodní roh. V tomto případě tedy ADGE zpracovává difúzi ve směru severozápad–jihovýchod.

Algoritmus DGE se musí ještě upravit, protože podporuje pouze řešení systému čtyř a více rovnic a ADGE musí umět vyřešit i dvě rovnice o dvou neznámých. Úprava spočívá

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
CPU čas[ms]	20592	22745	23462	34725	33837	34164	33946
GPU čas[ms]	1264	982	1045	983	998	951	967
zrychlení	16.29	23.16	22.45	35.33	33.90	35.92	35.10
využitá teor. prop.	0.61	0.79	0.74	0.79	0.78	0.82	0.80
využitá prop. memcpy	0.69	0.89	0.84	0.89	0.88	0.92	0.90
GPU čas LDGE2[ms]	827	592	673	624	624	574	593
využitá teor. prop. alg.	0.63	0.88	0.77	0.83	0.83	0.90	0.87
využitá prop. memcpy alg.	0.70	0.98	0.87	0.93	0.93	1.02	0.98
obsazení MP	2.67	5.33	8	10.67	16	21.3	32

Tabulka 5.23: LDGE2, hodnoty naměřené na COMP1

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
CPU čas[ms]	16690	16760	17390	22620	26470	28130	28170
GPU čas[ms]	450	280	210	160	130	170	130
zrychlení	37.09	59.86	82.81	141.38	203.62	165.47	216.69
využitá teor. prop.	0.17	0.27	0.36	0.47	0.58	0.44	0.58
využitá prop. memcpy	0.22	0.36	0.48	0.63	0.78	0.59	0.78
GPU čas LDGE2[ms]	390	230	160	120	90	130	90
využitá teor. prop. alg.	0.13	0.22	0.31	0.42	0.56	0.38	0.56
využitá prop. memcpy alg.	0.17	0.29	0.42	0.56	0.74	0.52	0.74
obsazení MP	1.14	2.29	3.43	4.57	6.86	9.14	13.71

Tabulka 5.24: LDGE2, hodnoty naměřené na COMP2

v přidání dvou výplňových rovnic, které neovlivní řešení. Jedna rovnice se přidá na začátek a jedna na konec původního systému. Přidané rovnice můžou být například tvaru $x = 1$, kde x je nová proměnná, která se nevyskytuje v původním systému.

ADGE zavádí nový pomocný cyklus přes proměnnou j a převodní funkci. Převodní funkce má parametry index vlákna t a číslo iterace j. Hodnota převodní funkce určuje číslo iterace i cyklu v DGE, která se má provést. Hodnota převodní funkce, která je o jedna menší než možné hodnoty cyklu, určuje, že se má provést kód před cyklem. Hodnota převodní funkce, která je o jedna větší než možné hodnoty cyklu, určuje, že se má provést kód za cyklem. Otázka, která se klade při vytváření převodní funkce, je následující. Jaký má být vztah j a i, aby všechna aktivní vlákna ADGE přistupovala ke stejnému řádku v paměti? Takto vytvořená převodní funkce umožňuje přistupovat vláknům zarovnaně ke globální paměti. Převodní funkce má v tomto případě tvar

$$i = j - t. \tag{5.4}$$

Po získání tohoto vztahu je ještě potřeba určit počet iterací pomocného cyklu pro všechny čtyři algoritmy 3, 5, 4 a 6. Začátek pomocného cyklu určuje první aktivní vlákno a konec pomocného cyklu určuje vlákno, které jako poslední ukončí svůj výpočet. Tato vlákna se získají porovnáním obrázku 5.1 a rovnice (5.4). Pro zpřehlednění výpočtu je vhodné zavést pomocnou funkci M, která určuje délku diagonály plus dva, která přísluší vláknu t. Přičtení dvojky je tam kvůli dvěma výplňovým rovnicím. Funkce M má v tomto případě tvar

$$M = h - 1 + 2 - t. \tag{5.5}$$

Tabulka 5.25 obsahuje vypočítané okrajové hodnoty j pomocného cyklu. Z indexu vlákna a posunutí k v diagonále už je jednoduché vypočítat absolutní umístění hodnoty v globální paměti a to jako

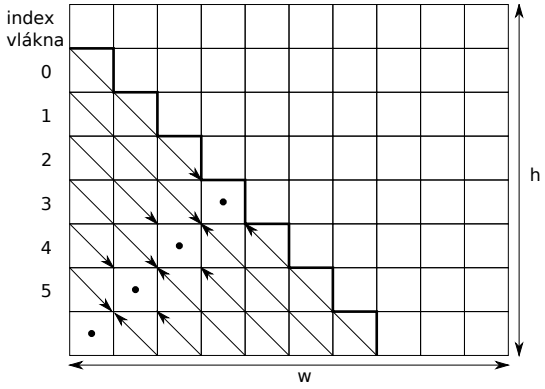
$$adr = w + w \cdot k \ + k \ + t \cdot w. \tag{5.6}$$

5.3.1 Difúze na zlepšení hran

V tabulce 5.26 je zachycený plán použití paměti a v tabulce 5.27 jeho popisky. V tabulkách 5.28 a 5.29 jsou výsledky měření. ADGE pro zvýraznění hran k řešení na COMP1 potřebuje

dopředná fáze, procesor1	$i = 0, t = 0$	$j_{min} = i + t = 0$
dopředná fáze, procesor1	$i = q - 1, t = h - 3$	$j_{max} = \lfloor \frac{M}{2} \rfloor - 1 + t = h - 2$
zpětná fáze, procesor1	$i = q - 1, t = h - 3$	$j_{max} = \lfloor \frac{M}{2} \rfloor - 1 + t = h - 2$
zpětná fáze, procesor1	$i = 1, t = 0$	$j_{min} = i + t = 1$
dopředná fáze, procesor2	$i = M - 1, t = 0$	$j_{max} = M - 1 + t = h$
dopředná fáze, procesor2	$i = q, t = 0$	$j_{min} = \lfloor \frac{M}{2} \rfloor + t = \lfloor \frac{h+1}{2} \rfloor$
zpětná fáze, procesor2	$i = q, t = 0$	$j_{min} = \lfloor \frac{M}{2} \rfloor + t = \lfloor \frac{h+1}{2} \rfloor$
zpětná fáze, procesor2	$i = M - 2, t = 0$	$j_{max} = M - 2 + t = h - 1$

Tabulka 5.25: okrajové hodnoty j pomocného cyklu



Obrázek 5.1: difúze v levém spodním rohu

minimálně 841 (947)ms. Z toho 583 (656)ms zabírají vlastní výpočty. ADGE pro zvýraznění hran k řešení na COMP2 potřebuje minimálně 81 (109)ms. Z toho 56 (76)ms zabírají vlastní výpočty.

5.3.2 Difúze na zlepšení koherence

Difúze na zlepšení koherence má oproti difúzi na zlepšení hran větší paměťové nároky, protože musí rozmazávat difúzní tenzor. Kvůli použitému algoritmu není možné dělat toto rozmazání lokálně a při difúzi ve 3–D by bylo potřeba mnoho dat navíc. 3–D difúze na zlepšení koherence není implementovaná pro GPU kvůli těmto paměťovým nárokům. V tabulce 5.30 je zachycený plán použití paměti a v tabulce 5.31 jeho popisky. V tabulkách 5.32 a 5.33 jsou výsledky měření. ADGE pro zlepšení koherence k řešení na COMP1 potřebuje minimálně 1036 (1166)ms. Z toho 583 (656)ms zabírají vlastní výpočty. ADGE pro zlepšení koherence k řešení na COMP2 potřebuje minimálně 100 (134)ms. Z toho 56 (76)ms zabírají vlastní výpočty.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
im	im												im
	$im^\top$												
	im	im_σ			y		y		y		y		
	$im^\top$	$im_\sigma^\top$			β		β		β		β		
			g_{-1}			im_{-1}							
			g_0		m		m		m		m		
			g_1					im_1					
			g_2							im_2			
				$g_0^\top$								$im_0^\top$	

Tabulka 5.26: plán při použití ADGE pro zvýraznění hran

přechod	popis	přechod	popis
$0 \to 1$	kopírování, transpozice (4rw)	$7 \to 8$	zpětná fáze (4rw)
$1 \to 2$	lin. difúze (-rw)	$8 \to 9$	dopředná fáze (5rw)
$2 \to 3$	difúzní koeficienty (5rw)	$9 \to 10$	zpětná fáze (4rw)
$3 \to 4$	transpozice (2rw)	$10 \to 11$	dopředná fáze (5rw)
$4 \to 5$	dopředná fáze (5rw)	$11 \to 12$	zpětná fáze (4rw)
$5 \to 6$	zpětná fáze (4rw)	$12 \to 13$	průměr (5rw)
$6 \to 7$	dopředná fáze (5rw)		

Tabulka 5.27: popisky plánu pro ADGE pro zvýraznění hran

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
CPU čas[ms]	46036	51559	51106	52104	51854	54211	55614
GPU čas[ms]	3072	1904	1700	1591	1482	1467	1467
zrychlení	14.99	27.08	30.06	32.75	34.99	36.95	37.91
využitá teor. prop.	0.27	0.44	0.49	0.53	0.57	0.57	0.57
využitá prop. memcpy	0.31	0.50	0.56	0.60	0.64	0.65	0.65
GPU čas ADGE[ms]	2495	1467	1279	1201	1077	1046	1045
využitá teor. prop. alg.	0.23	0.40	0.46	0.49	0.54	0.56	0.56
využitá prop. memcpy alg.	0.26	0.45	0.51	0.55	0.61	0.63	0.63
obsazení MP	1.33	2.67	4	5.33	8	10.67	16

Tabulka 5.28: ADGE pro zlepšení hran, hodnoty naměřené na COMP1

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
CPU čas[ms]	24840	26340	26550	29330	29700	32090	33040
GPU čas[ms]	1450	770	550	430	310	270	230
zrychlení	17.13	34.21	48.27	68.21	95.81	118.85	139.30
využitá teor. prop.	0.06	0.11	0.15	0.19	0.26	0.30	0.35
využitá prop. memcpy	0.08	0.14	0.20	0.25	0.35	0.40	0.47
GPU čas ADGE[ms]	1370	700	490	370	250	220	170
využitá teor. prop. alg.	0.04	0.08	0.11	0.15	0.22	0.25	0.33
využitá prop. memcpy alg.	0.06	0.11	0.16	0.21	0.30	0.35	0.45
obsazení MP	0.57	1.14	1.71	2.29	3.43	4.57	6.86

Tabulka 5.29: ADGE pro zlepšení hran, hodnoty naměřené na COMP2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
im	im																			im
	$im^\top$																			
	im	im_σ								g_{-1}			im_{-1}							
	$im^\top$	$im_\sigma^\top$								g_0		y		y		y		y		
			a		a_ρ						$g_0^\top$				$im_0^\top$					
			b				b_ρ					β		β		β		β		
			c						c_ρ			m		m		m		m		
				$a^\top$	$a_\rho^\top$	$b^\top$	$b_\rho^\top$	$c^\top$	$c_\rho^\top$	g_1							im_1			
										g_2									im_2	

Tabulka 5.30: plán při použití ADGE pro zvýraznění koherence

přechod	popis	přechod	popis
$0 \to 1$	kopírování, transpozice (4rw)	$10 \to 11$	transpozice (2rw)
$1 \to 2$	lin. difúze (-rw)	$11 \to 12$	dopředná fáze (5rw)
$2 \to 3$	strukturní tenzor (4rw)	$12 \to 13$	zpětná fáze (4rw)
$3 \to 4$	transpozice (2rw)	$13 \to 14$	dopředná fáze (5rw)
$4 \to 5$	lin. difúze (-rw)	$14 \to 15$	zpětná fáze (4rw)
$5 \to 6$	transpozice (2rw)	$15 \to 16$	dopředná fáze (5rw)
$6 \to 7$	lin. difúze (-rw)	$16 \to 17$	zpětná fáze (4rw)
$7 \to 8$	transpozice (2rw)	$17 \to 18$	dopředná fáze (5rw)
$8 \to 9$	lin. difúze (-rw)	$18 \to 19$	zpětná fáze (4rw)
$9 \to 10$	difúzní koeficienty (7rw)	$19 \to 20$	průměr (5rw)

Tabulka 5.31: popisky plánu pro ADGE pro zvýraznění koherence

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
CPU čas[ms]	66159	72057	71994	72712	72041	75847	78219
GPU čas[ms]	3526	2216	1997	1841	1732	1716	1747
zrychlení	18.76	32.52	36.05	39.50	41.59	44.20	44.77
využitá teor. prop.	0.29	0.47	0.52	0.56	0.60	0.60	0.59
využitá prop. memcpy	0.33	0.53	0.58	0.63	0.67	0.68	0.67
GPU čas ADGE[ms]	2450	1467	1295	1186	1092	1061	1061
využitá teor. prop. alg.	0.24	0.40	0.45	0.49	0.53	0.55	0.55
využitá prop. memcpy alg.	0.27	0.45	0.51	0.55	0.60	0.62	0.62
obsazení MP	1.33	2.67	4	5.33	8	10.67	16

Tabulka 5.32: ADGE pro zlepšení koherence, hodnoty naměřené na COMP1

počet iterací	3200	800	356	200	89	50	22
velikost strany[pix]	256	512	768	1024	1536	2048	3072
CPU čas[ms]	34360	35400	35960	39280	38670	41720	41960
GPU čas[ms]	1510	820	580	460	350	310	250
zrychlení	22.75	43.17	62.00	85.39	110.49	134.58	167.84
využitá teor. prop.	0.07	0.12	0.17	0.22	0.29	0.32	0.40
využitá prop. memcpy	0.09	0.16	0.23	0.29	0.38	0.43	0.54
GPU čas ADGE[ms]	1380	720	490	370	270	230	160
využitá teor. prop. alg.	0.04	0.08	0.11	0.15	0.21	0.24	0.35
využitá prop. memcpy alg.	0.06	0.11	0.16	0.21	0.28	0.33	0.48
obsazení MP	0.57	1.14	1.71	2.29	3.43	4.57	6.86

Tabulka 5.33: ADGE pro zlepšení koherence, hodnoty naměřené na COMP2

Kapitola 6

CUDA implementace 3–D difúze

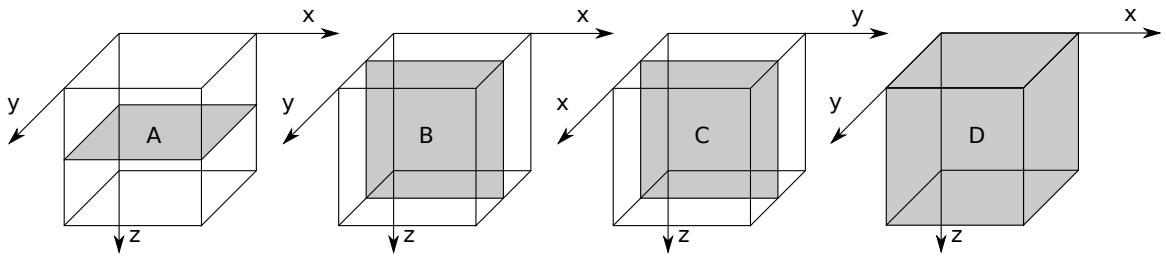
Implementace z předchozí kapitoly jsou použité jako stavební kameny pro 3–D lineární (L3D), izotropní (I3D) a anizotropní difúzi se zvýrazněním hran (A3D). Všechny tři implementace zpracovávají 3–D data po řezech, protože při 3–D difúzi dochází rychle k vyčerpání paměti. Do 1GB se například pouze vejde jedna krychle, která má délku hrany 645. Dále v textu se označuje blokem paměť o velikosti, kterou zabírají vstupní data. Nejdůležitější je tedy, aby implementace potřebovaly nejmenší možný počet bloků dat. Na obrázku 6.1 jsou zobrazené možné řezy 3–D daty a je zavedené označení řezů, které se bude dále v textu používat. Počet čárek u označení určuje, ve kterém bloku se daný řez nachází. Dále v textu se používá označení rw_A, rw_B a rw_C, aby se rozlišily velikosti různých druhů řezů při počítání paměťových operací.

6.1 Lineární 3–D difúze

L3D k výpočtu používá dva bloky a pomocnou paměť pro výpočet difúze na řezech. Jeden blok odpovídá vstupu iterace i a druhý blok výstupu iterace i. V $i + 1$ iteraci si bloky role prohodí. Ve výstupním bloku se postupně ukládají výsledky difúzí jednotlivých řezů. V tabulce 6.1 je zachycený plán použití paměti a v tabulce 6.2 jeho popisky. V tabulce 6.3 jsou výsledky měření. L3D k řešení na COMP1 potřebuje minimálně 4557 (5131)ms. L3D k řešení na COMP2 potřebuje minimálně 441 (591)ms.

6.2 Izotropní 3–D difúze

I3D k výpočtu používá tři bloky a pomocnou paměť pro výpočet difúze na řezech. První blok odpovídá vstupu iterace i, druhý blok rozmazaným datům a difúzním koeficientům a třetí blok výstupu iterace i. V $i + 1$ iteraci si první a třetí blok role prohodí. Ve výstupním bloku se postupně ukládají výsledky difúze jednotlivých řezů. V tabulce 6.4 je zachycený plán použití paměti a v tabulce 6.5 jeho popisky. V tabulce 6.6 jsou výsledky měření. I3D



Obrázek 6.1: možné řezy v 3–D datech

pro každý řez A					pro každý řez B				
0	1	2	3	4	5	6	7	8	9
A					B				
				A'					B'
	$r^\top$		$r^\top$			r		r	
		y					y		
		$y^\top$							
			r						

Tabulka 6.1: plán pro L3D

přechod	popis	přechod	popis
$0 \to 1$	transpozice ($2rw_A$)	$5 \to 6$	řez ($2rw_B$)
$1 \to 2$	dopředná fáze ($4rw_A$)	$6 \to 7$	dopředná fáze ($2rw_B$)
$2 \to 3$	zpětná fáze ($4rw_A$)	$7 \to 8$	zpětná fáze ($2rw_B$)
$3 \to 4$	průměr ($3rw_A$)	$8 \to 9$	vážené přičtení řezu ($3rw_B$)

Tabulka 6.2: popisky plánu pro L3D

počet iterací	10	10	10	10	10	10
velikost	$512^2 \times 256$	$1024^2 \times 64$	$1536^2 \times 28$	$512^2 \times 256$	$1024^2 \times 64$	$1536^2 \times 28$
GPU čas	6310	6146	6380	1810	1060	940
využitá teor. prop.	0.72	0.74	0.71	0.24	0.42	0.47
prop. memcpy	0.81	0.83	0.80	0.33	0.56	0.63

Tabulka 6.3: L3D, vlevo hodnoty naměřené na COMP1, vpravo na COMP2

				pro každý řez A						pro každý řez B					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
D				A						B					
	D'	D'	D'	A'						B'					
									A''						B''
					$r^\top$			$r^\top$			$r^\top$			$r^\top$	
						$gr^\top$		r				$gr^\top$			
							y						y		
							β						β		
							m						m		
							$y^\top$								
							$\beta^\top$								
							$m^\top$								

Tabulka 6.4: plán pro I3D

přechod	popis	přechod	popis
$0 \to 1$	kopírování (2 velikosti bloku)	$8 \to 9$	průměr ($3rw_A$)
$1 \to 2$	lin. difúze (-)	$10 \to 11$	řez ($2rw_B$)
$2 \to 3$	difúzní koef.(2 velikosti bloku)	$11 \to 12$	řez ($2rw_B$)
$4 \to 5$	transpozice ($2rw_A$)	$12 \to 13$	dopředná fáze ($5rw_B$)
$5 \to 6$	transpozice ($2rw_A$)	$13 \to 14$	zpětná fáze ($4rw_B$)
$6 \to 7$	dopředná fáze ($10rw_A$)	$14 \to 15$	vážené přičtení řezu ($3rw_B$)
$7 \to 8$	zpětná fáze ($8rw_A$)		

Tabulka 6.5: popisky plánu pro I3D

k řešení na COMP1 potřebuje minimálně 8906 (10028)ms. I3D k řešení na COMP2 potřebuje minimálně 862 (1156)ms.

6.3 Anizotropní 3–D difúze

A3D k výpočtu používá tři bloky a pomocnou paměť pro výpočet difúze na řezech. První blok odpovídá vstupu iterace i, druhý blok rozmazaným datům a třetí blok výstupu iterace i. V $i+1$ iteraci si tentokrát první a třetí blok role neprohazují. Ve výstupním bloku se postupně ukládají výsledky difúzí jednotlivých řezů. Tentokrát se musí počítat difúzní koeficienty vícekrát, protože není dostatek paměti, aby se uložily všechny směry. V každém řezu se tedy opakovaně počítají difúzní koeficienty pro všechny směry, uloží se ale pouze směry,

počet iterací	10	10	10	10	10	10
velikost	$512^2 \times 256$	$1024^2 \times 64$	$1536^2 \times 28$	$512^2 \times 256$	$1024^2 \times 64$	$1536^2 \times 28$
GPU čas	13713	13916	13993	3510	2320	2010
využitá teor. prop.	0.65	0.64	0.64	0.25	0.37	0.43
prop. memcpy	0.73	0.72	0.72	0.33	0.50	0.58

Tabulka 6.6: I3D, vlevo hodnoty naměřené na COMP1, vpravo na COMP2

			pro každý řez A												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
D			A												
	D'	D'	A'												
															A''
				$g_0 r$			y		y		y		y		
				$g_1 r$						im_1					
				$g_3 r$								im_3			
				$g_4 r$										im_4	
					$g_0 r^\top$			$im_0^\top$							
							β		β		β		β		
							m		m		m		m		
						$r^\top$									

Tabulka 6.7: plán pro A3D, část a

přechod	popis	přechod	popis
$0 \to 1$	kopírování (2 velikosti bloku)	$8 \to 9$	dopředná fáze ($5rw_A$)
$1 \to 2$	lin. difúze (-)	$9 \to 10$	zpětná fáze ($4rw_A$)
$3 \to 4$	difúzní koef. ($5rw_A$)	$10 \to 11$	dopředná fáze ($5rw_A$)
$4 \to 5$	transpozice ($2rw_A$)	$11 \to 12$	zpětná fáze ($4rw_A$)
$5 \to 6$	transpozice ($2rw_A$)	$12 \to 13$	dopředná fáze ($5rw_A$)
$6 \to 7$	dopředná fáze ($5rw_A$)	$13 \to 14$	zpětná fáze ($4rw_A$)
$7 \to 8$	zpětná fáze ($4rw_A$)	$14 \to 15$	průměr ($5rw_A$)

Tabulka 6.8: popisky plánu pro A3D, část a

které jsou právě zapotřebí. K výpočtu řezu C se napřed provede transpozice řezu A. Po této operaci se z řezu C stanou řezy B a je možné k nim efektivně přistupovat. Tabulky 6.7, 6.9 a 6.11 zachycují plán použití paměti a tabulky 6.8, 6.10 a 6.12 jeho popisky. Předchozí tabulky zachycují pouze případ, ve kterém je šířka větší než výška, šířka je větší než hloubka a výška je větší než hloubka. Tento případ je nejjednodušší, protože nepotřebuje žádné pomocné přesuny v paměti. Ostatní případy ale nejsou o moc komplikovanější a dají se znázornit obdobně. V tabulce 6.13 jsou výsledky měření. A3D k řešení na COMP1 potřebuje minimálně 26305 (29618)ms. A3D k řešení na COMP2 potřebuje minimálně 2546 (3413)ms.

pro každý řez B										
16	17	18	19	20	21	22	23	24	25	26
B										
B'										
										B''
	r								r	
		$g_2 r$		im_2						
		$g_5 r$				im_5				
		$g_6 r$						im_6		
			y		y		y			
			β		β		β			
			m		m		m			

Tabulka 6.9: plán pro A3D, část b

přechod	popis	přechod	popis
$16 \to 17$	řez ($2rw_B$)	$21 \to 22$	zpětná fáze ($4rw_B$)
$17 \to 18$	difúzní koef. ($4rw_B$)	$22 \to 23$	dopředná fáze ($5rw_B$)
$18 \to 19$	dopředná fáze ($5rw_B$)	$23 \to 24$	zpětná fáze ($4rw_B$)
$19 \to 20$	zpětná fáze ($4rw_B$)	$24 \to 25$	průměr ($4rw_B$)
$20 \to 21$	dopředná fáze ($5rw_B$)	$25 \to 26$	vážené přičtení řezu ($2rw_B$)

Tabulka 6.10: popisky plánu pro A3D, část b

pro každý řez A				pro každý řez C										
27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
A				C									A''	
	A			C'										A
A'												C''		
		A'			r						r			
A''						$g_7 r$		im_7						
			A''			$g_8 r$				im_8				
							y		y					
							β		β					
							m		m					

Tabulka 6.11: plán pro A3D, část c

přechod	popis	přechod	popis
$27 \to 28$	transpozice ($2rw_A$)	$34 \to 35$	zpětná fáze ($4rw_C$)
$28 \to 29$	transpozice ($2rw_A$)	$35 \to 36$	dopředná fáze ($5rw_C$)
$29 \to 30$	transpozice ($2rw_A$)	$36 \to 37$	zpětná fáze ($4rw_B$)
$31 \to 32$	řez ($2rw_C$)	$37 \to 38$	průměr ($3rw_C$)
$32 \to 33$	difúzní koef. ($3rw_C$)	$38 \to 39$	vážené přičtení řezu ($2rw_C$)
$33 \to 34$	dopředná fáze ($5rw_C$)	$40 \to 41$	transpozice ($2rw_A$)

Tabulka 6.12: popisky plánu pro A3D, část c

počet iterací	10	10	10	10	10	10
velikost	$512^2 \times 256$	$1024^2 \times 64$	$1536^2 \times 28$	$512^2 \times 256$	$1024^2 \times 64$	$1536^2 \times 28$
GPU čas	74110	68625	67314	27320	17610	15210
využitá teor. prop.	0.35	0.38	0.39	0.09	0.14	0.17
prop. memcpy	0.40	0.43	0.44	0.12	0.19	0.22

Tabulka 6.13: A3D, vlevo hodnoty naměřené na COMP1, vpravo na COMP2

Kapitola 7

Závěr

V této diplomové práci se podařilo najít vhodné algoritmy pro GPU a napsat podstatně rychlejší implementace difuzních filtrů pro GPU než pro CPU. Implementace pro CPU určitě jde vylepšit, protože používá pouze jedno vlákno. To se musí vzít v úvahu, když se porovnávají rychlosti mezi CPU a GPU implementacemi. Implementace pro CPU ale není úplně neefektivní, protože stejně jako implementace pro GPU lineárně škáluje s velikostí vstupních dat. Hlavní indicie toho, že GPU implementace jsou efektivní, je část teoretické propustnosti globální paměti, která je využitá. U ADGE by mělo být možné vylepšit využití propustnosti globální paměti, kdyby se zpracovávaly všechny směry najednou.

Zpracovává se pouze jeden směr, protože u 3–D difúze se zpracovává různý počet směrů v závislosti na typu řezu. Tímto krokem se trochu zjednodušila implementace 3–D difúze a ušetřilo trochu paměti, které je u 3–D difúze nedostatek. Hlavní nevýhoda pro počítání 3–D difúze na GPU je totiž omezená velikost paměti GPU. Pro 3–D anizotropní difúzi se také podařilo najít rozklad strukturního tenzoru. Tento rozklad se mnoho neliší od rozkladu 2–D difúzního tenzoru, ale je to jediný teoretický výsledek této diplomové práce.

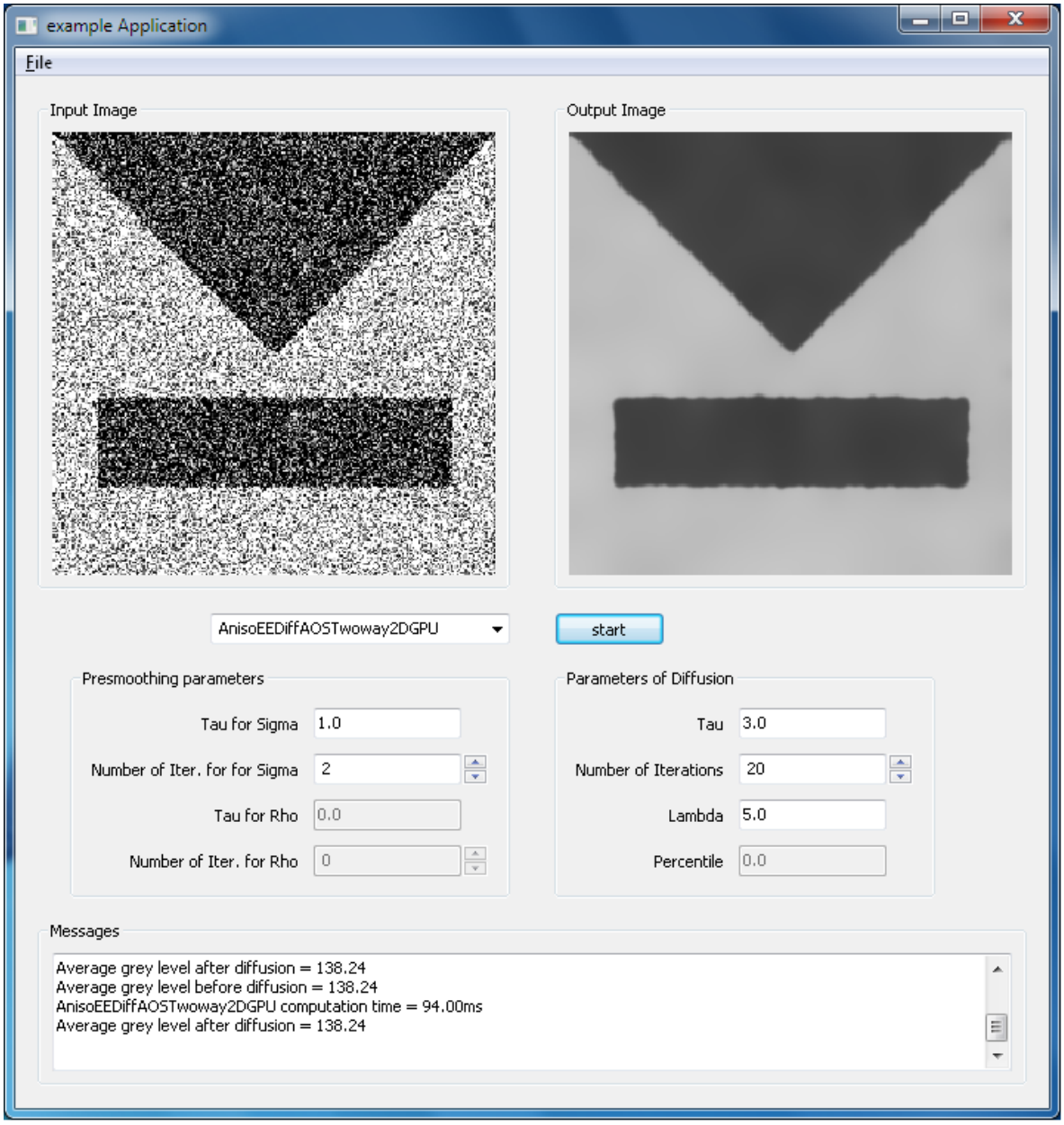
Jako součást této diplomové práce byly napsané dva programy. První program je GUI aplikace, ve které je možné vyzkoušet všechny popsané 2–D difúze. Program zobrazí zmenšený výsledek v pravém panelu a nezmenšený výsledek uloží do souboru *vystup.png*. Je v něm také možné porovnat rozdíl výsledků vypočítaných na CPU a GPU. Tento program používá knihovnu *wxWidgets*, takže pro přeložení je nutné mít tuto knihovnu nainstalovanou v systému. V příloze A je ukázaný vzhled tohoto programu. V příloze B jsou příklady, jakých výsledků se dá při použití difúzních filtrů dosáhnout. [1]

Druhý program počítá 3–D difúzi a není interaktivní. Program nepodporuje anizotropní difúzi pro zvýraznění koherence z důvodů, které je popsané v páté kapitole. Tento program používá pro načítaní a ukládání souborů knihovnu *i3dcore*[2]. Tento program vypíše návod pro použití, jestliže je spuštěný bez parametrů.

1. dostupné na http://imageprocessingplace.com/root_files_V3_image_databases.htm
2. dostupná na http://cbia.fi.muni.cz

Příloha A

Vzhled GUI



Obrázek A.1: vzhled GUI

Příloha B

Experimenty

Obrázek B.1: kameraman

Obrázek B.2: kameraman s Gaussovým šumem, $\sigma = 30$

Obrázek B.3: odstranění šumu lineární difúzí $\tau = 3.0$, $n = 10$

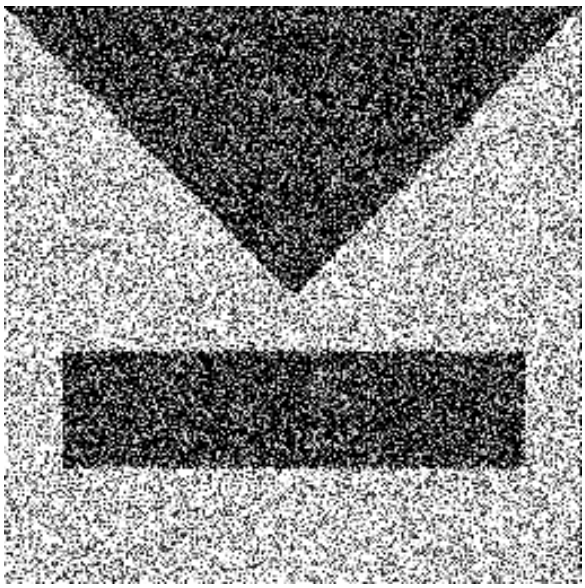
Obrázek B.4: odstranění šumu izotropní difúzí $\tau_\sigma = 1.0$, $n_\sigma = 1$, $\tau = 3.0$, $n = 10$, $\lambda = 4.0$

Obrázek B.5: odstranění šumu anizotropní difúzí $\tau_\sigma = 1.0$, $n_\sigma = 1$, $\tau = 3.0$, $n = 10$, $\lambda = 4.0$

Obrázek B.6: otisk prstu

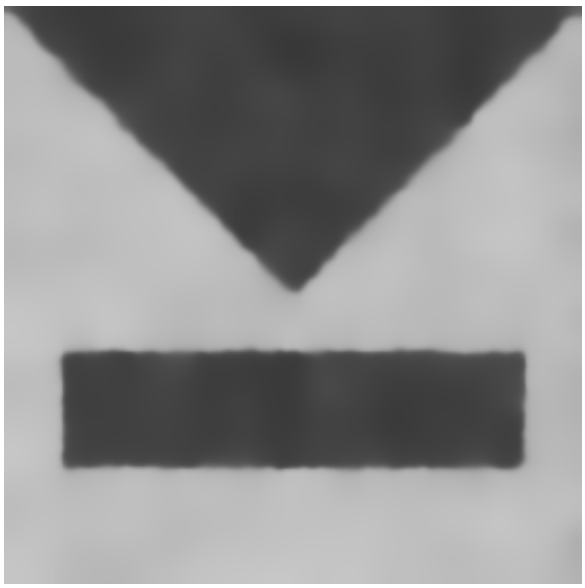
Obrázek B.7: zvýraznění koherence otisku prstu $\tau_\sigma = 1.0$, $n_\sigma = 1$, $\tau_\rho = 1.0$, $n_\rho = 15$, $\tau = 3.0$, $n = 10$

Obrázek B.8: trojúhelník s obdélníkem



Obrázek B.9: trojúhelník s obdélníkem s rovnoměrným šumem

Obrázek B.10: odstranění šumu lineární difúzí $\tau = 3.0$, $n = 20$



Obrázek B.11: odstranění šumu izotropní difúzí $\tau_\sigma = 1.0$, $n_\sigma = 1$, $\tau = 3.0$, $n = 20$, $\lambda = 5.0$

Obrázek B.12: odstranění šumu anizotropní difúzí $\tau_\sigma = 1.0$, $n_\sigma = 1$, $\tau = 3.0$, $n = 20$, $\lambda = 5.0$

Příloha C

Dodatky

Na přiloženém disku jsou všechny zdrojové kódy, které jsou zapotřebí k vytvoření programu, LaTeXsoubory, ze kterých byla vysázená tato diplomová práce a všechny obrázky.
Struktura disku:

- /source - zdrojové kódy a obrázky
- /master_thesis - zdrojové LaTeXsoubory a obrázky

Literatura

[1] Weickert, J.; Romeny, B.M.T.; Viergever, M.A. *Efficient and reliable schemes for nonlinear diffusion filtering*. In IEEE Transactions on Image Processing. 1998.

[2] Weickert, J. *Anisotropic diffusion in image processing*. 1998. Dostupné na: http://citeseerx .ist.psu.edu/viewdoc/download?doi=10.1.1.11.7513 &rep=rep1&.

[3] Mrázek, Pavel. *Nonlinear Diffusion for Image Filtering and Monotonicity Enhancement*. 2001. PhD Thesis. Dostupné na: http://cmp.felk.cvut.cz/ftp/articles/mrazek/Mrazek-phd01.pdf.

[4] NVIDIA. *CUDA Programming Guide 3.1*, 2010. Dostupné na http://developer. download.nvidia.com/compute/cuda/3_1/ toolkit/docs/ NVIDIA_CUDA_C_ ProgrammingGuide_3.1.pdf.

[5] Zhang, Yao; Cohen, Jonathan; Owens, John D.. *Fast Tridiagonal Solvers on the GPU*. 2010. Dostupné na: http://jcohen.name/papers/Zhang_Fast_2009.pdf.

[6] Walshaw, C.; Farr, J.. *A Two–Way Parallel Partition Method for Solving Tridiagonal Systems*. 1993. Dostupné na: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.3646.

[7] Egloff, Daniel. *High Performance Finite Difference PDE Solvers on GPUs*. 2010. Dostupné na: http://download.quantalea.net/fdm_gpu.pdf.

[8] Ruetsch, Greg; Micikevicius, Paulius. *Optimizing Matrix Transpose in CUDA*. 2009. Dostupné na: http://developer.download.nvidia.com/assets/cuda/files/MatrixTranspose.pdf.

[9] Horová, Ivana; Zelinka, Jiří. *Numerické metody*. 2008. ISBN 978-80-210-3317-7.

[10] Hefferon, Jim. *Linear Algerbra*. 2008. Dostupné na http://joshua.smcvt.edu/ linearalgebra.