SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

STAVEBNÁ FAKULTA

Evidenčné číslo: SvF-5342-67698

Vytváranie adaptívnych mriežok pre riešenie difúznych rovníc

Bakalárska práca

Študijný program: Matematicko-počítačové modelovanie

Číslo študijného odboru: 1114

Názov študijného odboru: Aplikovaná matematika

Školiace pracovisko (katedra/ústav): Katedra matematiky a deskriptívnej geometrie

Vedúci záverečnej práce/školiteľ: doc. RNDr. Zuzana Krivá, PhD.

Bratislava 2012 **Martin Trubač**

Čestné prehlásenie:

Čestne prehlasujem, že som bakalársku prácu vypracoval samostatne s použitím uvedenej literatúry a s odbornou pomocou vedúceho práce.

.....

V Bratislave 6.5.2012 Vlastnoručný podpis

Poďakovanie:

Rád by som sa poďakoval svojej konzultantke, doc. RNDr. Zuzane Krivej, PhD. za jej pomoc, odborné rady a cenné pripomienky, ktoré mi ochotne poskytovala pri tvorbe tejto práce.

Abstrakt

Práca popisuje postup pri vytváraní adaptívnej siete pre vstupné dáta (reprezentované ako obrázky) za pomoci stromovej štruktúry pracujúcej na princípe zlučovania susediacich častí obrázka, ktoré majú približne rovnakú hodnotu farby. Následne sa pre konkrétnu diferenciálnu rovnicu vytvorí sústava lineárnych rovníc a tá sa ďalej iteračne rieši pomocou SOR metódy. Program je vytvorený v jazyku C a pracuje s obrázkami VTK a PGM.

Abstract

This work describes a process of creating adaptive grid for input data (represented by images) using tree structure, which works in the way of merging adjacent parts of image, that have approximately the same value of color. Then, a linear system of equations is created for a specific differential equation, that is beeing solved by SOR iteration method. Program is created in C language and uses VTK and PGM images.

Obsah

Úvod

Numerické modelovanie matematických problémov vo väčšine prípadov vedie k riešeniu sústavy rovníc. Adaptívne mriežky slúžia okrem iného na diskretizáciu výpočtovej oblasti a ich veľkou výhodou oproti pravidelným mriežkam je najmä ten, že nám umožňujú zjednodušiť výpočet, pretože dokážu eliminovať častokrát veľké množstvo neznámych z tohto systému lineárnych rovníc. Na druhej strane narábanie a samotné pracovanie s takouto štruktúrou je o niečo komplikovanejšie než pri pravidelnej mriežke, a to najmä z hľadiska pristupovania k informáciam o susedných elementoch, ktoré môže v programe viesť k nepríjemnostiam v podobe rozsiahlej siete podmienok.

Práca ako taká sa dá rozdeliť do dvoch ťažiskových tém. Prvá časť bakalárskej práce popisuje princípy kvadrantového stromu, neskôr sa venuje vytváraniu samotnej adaptívnej mriežky, ktorá stojí práve na myšlienke tohto stromu. Vysvetlujú sa niektoré súčasti programu a modifikácie mriežky, ktoré sme vytvorili. V druhej časti sa popisuje diferenciálna rovnica vedenia tepla a rovnica pre krivostný filter, načrtnú sa algoritmy na vytvorenie lineárneho systému rovníc a napokon sa prezentujú a opisujú výstupy programu na niekoľkých vstupných obrázkoch.

1 Adaptívna mriežka

1.1 Kvadrantový strom

1.1.1 Hlavná myšlienka kvadrantového stromu

Predstavme si ľubovoľný čiernobiely obrázok. Zrejme najjednoduchší, ale aj pamäťovo najnáročnejší spôsob, akým tento obrázok možno v počítači uchovať je taký, že si za sebou odpamätáme hodnotu každého jedného pixela obrázku. Ak ale máme obrázok, ktorý je z 95% tvorený bielou farbou a zvyšných 5% je len nejaký malý čierny objekt v strede, odpamätávanie hodnoty každého jedného bodu obrázku je pomerne neefektívne, ako čoskoro uvidíme. Pri takomto kódovaní obrázkov nezáleží na tom, čo sa na samotnom obrázku nachádza. Veľkosť miesta, ktoré obrázok zaberá v počítači, závisí iba od rozmerov.

V dnešnej dobe sa ale snažíme zefektívniť čo sa dá, a preto nastupuje na rad kompresia. Jeden zo spôsobov, ako zakódovať obrázok je použiť takzvaný kvadrantový strom (z anglického "quadtree"). Tento druh kompresie ťaží z toho, že niektoré súvislé časti obrázka majú rovnakú, resp. približne rovnakú farbu. Princíp fungovania je veľmi jednoduchý - v prvom kroku sa skontroluje, či sú farby všetkých pixelov obrázka rovnaké (resp. približne rovnaké) a v prípade že nie sú, obrázok sa rozdelí na 4 rovnaké kvadranty a v každom z nich sa skontroluje intenzita farieb všetkých pixelov v ich vnútri. Pokiaľ sa zistí, že v niektorom z kvadrantov nie je zachovaná homogenita farieb, použije sa na tento kvadrant rovnaký postup ako v predošlom na celý obrázok, t.j. rozdelí sa na 4 rovnaké časti a pre každú z nich sa skontroluje intenzita farieb vnútorných pixelov. V opačnom prípade, ak sú farby v nejakom kvadrante rovnaké (resp. približne rovnaké), oblasť sa ďalej nedelí a berie sa ako jeden súvislý element, pre ktorý si stačí odpamätať jedinú hodnotu farby - tá bude reprezentovať farbu všetkych pixelov, ktoré sa v tomto elemente nachádzajú. Rekurzívnym algoritmom sa teda z obrázka vytvorí akási stromová štruktúra, ktorá nie je z hľadiska pamäte náročná na odpamätanie.

Ak použijeme takéto kódovanie na obrázok, ktorý je kompletne celý pokrytý bielou farbou, vidíme, že rekurzívny algoritmus delenia zastane už pri prvom kroku. Na zakódovanie takéhoto obrázku nám teda stačí odpamätať si 2 veci, a to, že obrázok sa ďalej nedelí a jednu hodnotu farby, ktorá je reprezentatívna pre celý obrázok. Tu vidieť, že zatiaľ čo pri prvom spôsobe kódovania takéhoto obrázku by sa veľkosť pamäťových

nárokov zvyšovala, ak by sme rozmery obrázku zväčšovali, pri kódovaní pomocou kvadrantových stromov môže byť obrázok akokoľvek veľký, pokiaľ je však celý biely, bude zaberať stále rovnaké množstvo pamäte.

1.1.2 Teoretický popis kvadrantového stromu

Na kvadrantový strom sa možno z matematického hľadiska pozerať ako na graf. Počiatočný kvadrant veľkosti $2^N \times 2^N$ pokrývajúci celé vstupné dáta sa nazýva koreň *(root)*. Štyri podkvadranty, ktoré vzniknú delením nejakého kvadrantu P, sa nazývajú jeho deti *(children)* a P je ich rodič *(parent)*. Pre nás bude základné kritérium pre ukončenie delenia nejakého kvadrantu také, že rozdiel intenzít všetkých pixelov z tohto kvadrantu bude v rámci nejakej vopred predpísanej hodnoty. Zastavenie delenia pre kvadrant veľkosti $2^{N-i} \times 2^{N-i}$ znamená, že kritérium bolo splnené a že namiesto pracovania s $2^{N-i} \times 2^{N-i}$ pixelmi podobnej intenzity vytvoríme jednu veľkú bunku takejto veľkosti, pričom jej reprezentatívna farba sa vypočíta ako priemerná hodnota intenzít farieb pixelov, ktoré sa v nej nachádzajú. Naopak, delenie kvadrantu o veľkosti $2^{N-i} \times 2^{N-i}$ znamená, že kritérium nebolo splnené, vďaka čomu tento kvadrant rozdelíme na 4 podkvadranty, kde každý z nich má veľkosť $2^{N-i-1} \times 2^{N-i-1}$. Ďalej budeme vychádzať z toho, že koreň je na úrovni 0. Ak je potom nejaký rodič na úrovni i, jeho 4 deti budú na úrovni $i + 1$. Napokon môžeme v grafe ešte rozlíšiť 2 druhy uzlov. Kvadrant, ktorý ďalej nemá žiadnych potomkov, sa nazýva list stromu *(leaf)*, naopak kvadrant s deťmi sa nazýva vnútorný uzol *(inner node)* grafu. Samotný obrázok je potom reprezentovaný listami stromu.

Takto popísaný kvadrantový strom je možno jednoducho popísať dvoma jednorozmernými zoznamami L_1 a L_2 takým spôsobom, aby nám umožňovali prehľadávať graf do hĺbky. Pole L_1 uchováva informáciu o štruktúre stromu, za pomoci postupnosti čísel 1 (delenie kvadrantu) a 0 (nedelenie kvadrantu). Druhé pole L_2 v sebe nesie údaje o intenzitách farieb pre jednotlivé vytvorené bunky, čo sú z hľadiska grafu listy stromu. V tejto práci však z určitých dôvodov, ktoré neskôr vyplynú, budeme namiesto jednorozmerných zoznamov používať dvojrozmerné polia L_1 a L_2 [1].

1.2 Využitie mriežky v práci

Čiernobiely obrázok môžme z matematického hľadiska chápať ako nejakú dvojrozmernú oblasť bodov, kde každý bod v sebe nesie akúsi informáciu (intenzitu farby), ktorá z praktického hľadiska môže predstavovať napríklad teplotu, hustotu, nadmorskú výšku a pod. Uvažujme teda čiernobiely obrázok rozmerov $n \times n$, ktorý nám reprezentuje akýsi počiatočný stav výpočtovej oblasti. Našou úlohou je sledovať, resp. počítať nejaké správanie sa tejto oblasti v čase, ak vieme, že toto správanie je popísané nejakou matematickou diferenciálnou rovnicou. Pre numerické počítanie je zrejme najjednoduchšia diskretizácia obrázka taká, že za uzlové body diskretizačnej siete budeme považovať všetky pixely obrázka. Ide o takzvanú neadaptívnu metódu. Tu však rovnako možno pouvažovať nad využitím kvadrantového stromu a vytvoriť pomocou neho adaptívnu diskretizačnú sieť. Ak totiž máme nejakú súvislú časť obrázka, v ktorej majú všetky pixely rovnakú alebo približne rovnakú farbu, môžeme ju prehlásiť za jeden celok, pretože i z hľadiska matematického výpočtu, medzi pixelmi s rovnakou farbou nedôjde k vzájomnej "výmene intenzít". Akákoľvek zmena farby v tomto elemente môže byť vyvolaná len jeho okolím, nie však pixelami z vnútra tejto oblasti. Z tohto pohľadu teda viaceré výpočty pre niekoľko pixelov môžem nahradiť jedným výpočtom pre jeden element. Pre neadaptívnu metódu platí, že každému uzlovému bodu zodpovedá neznáma v systéme lineárnych rovníc. Pri adaptívnej metóde môžeme $n \times n$ neznámych, ktoré nám vzniknú na pravidelnej hustej sieti zredukovať dokonca aj niekoľkonásobne, v závislosti od samotného obrázka. Okrem toho, ak pre jeden výpočtový element treba uchovávať veľa informácií, môžeme pomocou adaptívnej mriežky výrazne znížiť aj pamäťové nároky.

1.3 Vytvorenie základnej adaptívnej mriežky

1.3.1 Slovný popis algoritmu

K dispozícii máme teda oblasť pixelov v podobe obrázka a našou úlohou je nejakým spôsobom zlúčiť pixely, ktoré majú približne rovnakú intenzitu farby. K dosiahnutiu tohto cieľa využijeme práve kvadrantový strom. Na vytvorenie adaptívnej mriežky používame nerekurzívny algoritmus zlučovania, kdežto akékoľvek neskoršie počítanie na mriežke sa robí rekurzívne. Pre jednoduchosť predpokladáme, že náš

vstupný čiernobiely obrázok je z hľadiska rozmerov štvorcového charakteru o veľkosti $2^N \times 2^N$ pixelov, prípadne ak zavedieme označenie $n = 2^N$, môžeme zjednodušene písať, že vstupný obrázok je rozmerov n riadkov $\times n$ stĺpcov. Ďalej zadefinujme polia L_1 a L_2. Ako bolo vyššie uvedené, tieto polia volíme dvojrozmerné, pretože neskôr budeme potrebovať pristupovať k susedným elementom skúmaného štvorčeka, čo použitím tohto prístupu nie je také náročné, ako pri použití jednorozmerných polí opísaných v 1.1.2. Funkcia poľa L_2 spočíva v odpamätávaní si farieb elementov mriežky. Keďže vopred nevieme, ako mriežka bude vyzerať, uvažujúc najhorší možný prípad, ktorý môže nastať pri tvorbe mriežky, a to že nám vznikne pravidelná sieť veľkosti pôvodného obrázka, veľkosť tohto poľa zvolíme takú istú, t.j. $n \times n$. Pole L_1 nám bude vytvárať štruktúru mriežky. Potrebujeme pomocou neho do stredov štvorčekov zakódovať čísla 1 alebo 0. Možno ho chápať ako pole, ktorého prvky ležia na hranách jednotlivých pixelov. Jeho rozmery teda budú zodpovedať počtu vodorovných a zvislých čiar tvoriacich mriežku, čiže $(n + 1) \times (n + 1)$. Na začiatku pred vytváraním mriežky bude pole L_1 naplnené nulami a prvky poľa L_2 budú predstavovať farby všetkých pixelov. Vždy, keď dôjde k zlúčeniu 4 menších štvorčekov do jedného väčšieho elementu, tak sa do L_2 na pozíciu ľavého dolného rohu tohto elementu zapíše priemerná hodnota pixelov, v opačnom prípade sa do stredu elementu v poli L_1 zapíše 1, čo značí nezlúčenie, teda štvorček ostane rozdelený. Po vysvetlení základných potrebných prvkov možno algoritmus popísať v hrubých črtách nasledovne:

1. Začni zlučovanie na najvyššej úrovni $i = N$.

2. Postupne prechádzaj oblasťami veľkosti $2^{N-i+1} \times 2^{N-i+1}$ v obrázku a zisťuj, či môžeš zlúčiť 4 menšie štvorčeky o veľkosti $2^{N-i} \times 2^{N-i}$ z tejto skúmanej oblasti do jedného celku.

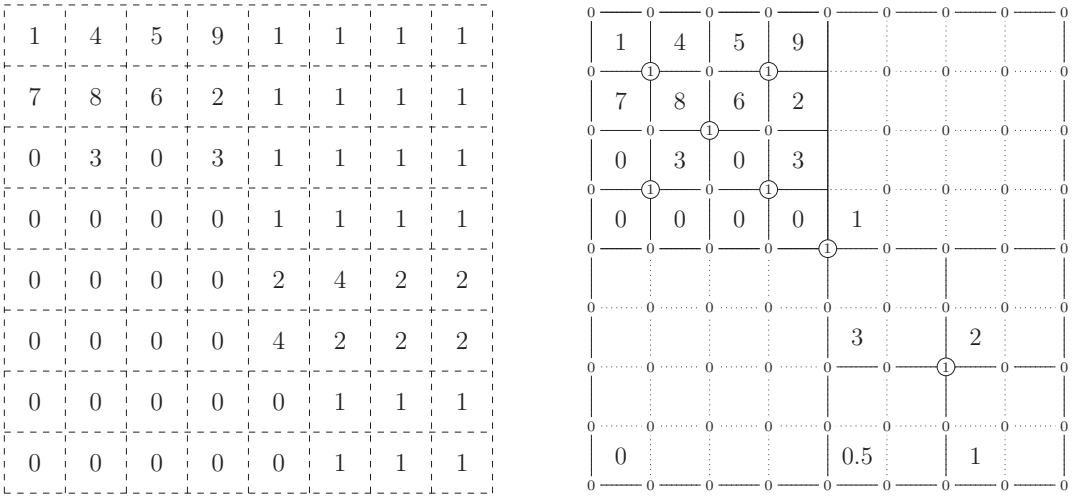
3. Ak je kritérium na zlučovanie splnené, zlúč štvorčeky, inak nezlučuj. V oboch prípadoch, ak je to potrebné, aktualizuj polia L_1 a L_2 a zapíš do nich potrebné informácie.

4. V cykle zníž i o jednotku.

5. Postupne opakuj kroky 2 - 5, pokiaľ je $i > 0$, v opačnom prípade skonči.

Inými slovami, program sa najprv snaží zlúčiť všetky možné štvorice pixelov (štvorčekov o dĺžke strany 1) do štvorčekov 2×2. Na ďalšej úrovni sa pokúša zlúčiť štvorice týchto 2×2 štvorčekov do štvorčeka 4×4. Takto sa postupuje, až kým sa nedostane do stavu, že sa pokúša zlúčiť 4 oblasti do elementu veľkosti celého obrázka, čo by sa samozrejme uskutočnilo len v prípade, že by všetky pixely obrázka mali takmer tú istú farbu.

1.3.2 Vysvetlenie niektorých súčastí programu na príklade



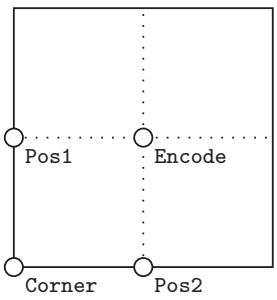
(a) Pôvodný obrázok **(b)** Po vytvorení mriežky

Obr. 1.1: Príklad tvorby mriežky

Obrázok 1.1 ukazuje vytvorenie mriežky pre vstupný obrázok o veľkosti 8×8 pixelov. Proces zlúčenia pixelov do nejakej oblasti sa v tomto prípade uskutoční len vtedy, ak rozdiel intenzít všetkých pixelov ktoré chcem zlúčiť je menší ako hodnota 3. Túto prahovú hodnotu (z ang. threshold) budeme označovať ε_1. Je to vlastne akési číslo, ktoré hovorí o tom, ako veľmi pri zlučovaní tolerujeme rôznorodosť farieb. V prípade, že zvolím $\varepsilon_1 = 0$, kvadranty sa zlúčia len za predpokladu, že všetky pixely v ich vnútri majú presne rovnakú intenzitu farby. Pole L_1 v obrázku môžme vidieť vo vrcholoch jednotlivých pixelov. Je naplnené nulami a jednotkami. Pre štvorčeky s veľkosťou hrany 2 a viac v skutočnosti prvky poľa L_1 môžu predstavovať ich stredy, čo využívame na zachytenie informácie o zlučovaní (0) alebo nezlučovaní (1) skúmaného štvorčeka. Z

obrázka ďalej vidíme, že veľkosť tohto poľa je o jedno väčšia než je veľkosť obrázka, teda v tomto prípade 9×9. Pole L_2 predstavuje v podstate pixely obrázka, preto má rovnakú veľkosť ako obrázok samotný, čiže 8×8 pixelov. Po vytvorení mriežky sú pre nás dôležité už len niektoré hodnoty z tohto poľa, konkrétne sú to reprezentatívne farby všetkých buniek. Hodnotu reprezentatívnej farby pre nejakú väčšiu oblasť počítame ako priemer hodnôt všetkých pixelov, ktoré sa v tejto oblasti nachádzajú a zapisujeme ju do ľavého dolného rohu príslušného štvorčeka v poli L_2.

Spôsob, akým sa testuje, či sa štvorček zlúči alebo nie, je robený pomocou porovnávania minimálnej a maximálnej hodnoty farebnej intenzity zo všetkých pixelov v tejto oblasti. Ak platí, že $|max - min| < \varepsilon_1$, potom kvadranty môžme zlúčiť, pretože hodnota $|max - min|$ predstavuje najväčší možný rozdiel farieb pixelov v danej oblasti. Pokiaľ dôjde k zlúčeniu do väčšej oblasti, je výhodné si pre túto oblasť odpamätať hodnoty min a max, pretože by sa v ďalších krokoch mohla stať súčasťou zlúčenia do ešte väčšej oblasti. Ak by sa neskôr rozhodovalo napríklad nad zlúčením štyroch štvorčekov o veľkosti 16×16 pixelov do jedného o veľkosti 32×32, nemusí sa hľadať minimum a maximum zo všetkých $32^2 = 1024$ pixelov. Keďže pre každý zo štyroch štvorčekov 16×16 máme odpamätanú minimálnu a maximálnu hodnotu, stačí nájsť najmenšie z týchto štyroch miním a najväčšie zo štyroch maxím. Skutočne bude platiť, že nájdené min a max bude minimálna a maximálna hodnota celej oblasti 32×32 a ich rozdiel môžeme spokojne použiť pre testovanie s hodnotou ε_1. Na toto uchovávanie miním a maxím pre jednotlivé elementy využívame pomocné pole AUX, ktoré má rovnaké rozmery ako pole L_2. Samotné hodnoty miním a maxím, ako aj informácie o delení/nedelení a hodnoty farieb elementov sa ukladajú na špeciálne pozície, ktoré sú popísané na obrázku 1.2.



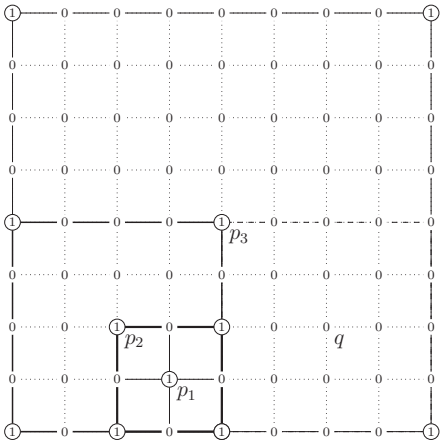
Obr. 1.2: Špeciálne pozície štvorčeka

Program pri vytváraní mriežky vidí obrázok ako súbor kvadrantov, ktorých veľkosť závisí na úrovni, na ktorej sa počíta. Aby sme ale pre konkrétny štvorček na konkrétnej úrovni vedeli povedať, ktoré miesta v poliach L_1, L_2 a AUX mu patria, vytvorili sa pre tento účel 4 pomocné funkcie. Na vstup dostanú informáciu takéhoto typu: skúma sa kvadrant, ktorý je v poradí k-tý zdola a l-tý zľava na úrovni i. Ich výstupom sú súradnice určujúce pozície v príslušných poliach pre skúmaný kvadrant. Funkciu `corner` využíva najmä pole $L2$, pretože vracia súradnice ľavého dolného rohu daného kvadrantu, čo sa využíva pri odpamätávaní si reprezentatívnej hodnoty pre kvadrant v prípade zlúčenia. Funkcia `encode` vracia súradnice stredu štvorčeka, čo je veľmi užitočné pri práci s L_1, pretože presne na túto pozíciu si zapíšeme informáciu o zlúčení, resp. nezlúčení detí skúmaného kvadrantu. Zvyšné 2 pozície slúžia pre pomocné dočasné pole AUX, v ktorom v prípade zlúčenia na pozíciu `pos1` uchovávame minimum a na pozíciu `pos2` maximum z hodnôt pixelov pre daný kvadrant. Okrem týchto funkcii na určenie pozícii v poliach máme vytvorených ešte niekoľko ďalších pomocných funkcií. Na hľadanie priemernej hodnoty v oblasti slúži funkcia `mean`, ktorá je veľmi jednoduchá. Vždy sa totiž počíta priemerná hodnota iba zo 4 hodnôt. Na najvyššej úrovni v prípade zlúčenia počítame priemernú hodnotu zo 4 pixelov a odpamätáme si ju do L_2 pre zlúčenú oblasť. V prípade, že má nastať zlúčenie 4 kvadrantov na nejakej nižšej i-tej úrovni, platí, že každý z týchto 4 kvadrantov musel byť vytvorený zlúčením ešte menších štvorčekov na úrovni $i + 1$. To znamená, že pre každý z nich mám odpamätanú priemernú hodnotu, a teda miesto počítania priemernej farby všetkých pixelov v skúmanej oblasti nám stačí vypočítať priemer štyroch odpamätaných priemerných hodnôt kvadrantov, ktoré zlučujeme. Vstup do funkcie `mean` je teda pozícia ľavého dolného rohu oblasti, ktorej deti chceme zlúčiť a hodnota predstavujúca polovicu dĺžky jej strany. Funkcia potom jednoducho vypočíta priemer zo štyroch hodnôt získaných z poľa L_2, pričom tieto hodnoty sú v skutočnosti ľavé dolné rohy každého zo zlučovaných 4 kvadrantov. Na tomto mieste teda nájdeme ich reprezentatívne farby a ich spriemerovaním dostaneme novú reprezentatívnu farbu pre novovzniknutý element, ktorú si odpamätáme na príslušnom mieste v L_2. Veľmi podobným spôsobom pracujú aj pomocné funkcie `minimum` a `maximum`. Vstup majú veľmi podobný ako funkcia `mean`, no ich úlohou je nájsť minimálnu, resp. maximálnu spomedzi 4 hodnôt. Tieto hodnoty môžu byť buď priamo farby jednotlivých pixelov, ak sa jedná o najvyššiu úroveň, alebo

na nižších úrovniach odpamätané minimá, resp. maximá 4 kvadrantov, ktoré sa testujú na zlúčenie.

1.3.3 Zjednodušenie programu z hľadiska delenia oblastí

Zlučovanie štvorčekov sa dá do istej miery (pre všetky úrovne okrem najvyššej) zjednodušiť. Platí totiž, že ak sa snažíme spojiť 4 deti nejakého štvorčeka do jedného celku, môže k tomu dôjsť len za predpokladu, že každé z týchto štyroch detí je súvislá nedelená oblasť. Ak by čo i len jedno dieťa bolo rozdelené na menšie štvorčeky, k spojeniu nedôjde. V skutočnosti nie je potrebné pre tento štvorček hľadať maximá a minimá. Dá sa vytvoriť jednoduchá, ale užitočná úprava. Pri nedelení nejakého skúmaného elementu vložíme jednotku nielen do stredu oblasti, ale i do všetkých jeho 4 vrcholov. Týmto sa prenesie informácia o nedelení na nižšiu úroveň, pretože vždy je jeden z vrcholov nejakého štvorčeka na úrovni i stredom štvorčeka na úrovni $i-1$. Inými slovami, deti nejakého rodičovského kvadrantu zdieľajú jeden spoločný vrchol, ktorý je v skutočnosti stredom rodičovskej oblasti. Ak čo i len jedno dieťa ostane rozdelené na menšie časti, do stredu rodiča sa dostane 1. Skôr, ako začneme čokoľvek súvisiace so zlúčením na rodičovi počítať, skontrolujeme jeho stred v poli L_1. Ak sa tam nachádza 1, vynechá sa niekoľko výpočtov a prejde sa rovno na vetvu programu starajúcu sa o nezlúčenie. Ak sa tam nachádza 0, každé z detí je súvislá oblasť a program pokračuje testovaním minima a maxima.



Obr. 1.3: Posielanie jednotiek do rohov

Obrázok 1.3 názorne ukazuje spôsob fungovania tejto myšlienky. Označenia p_1, p_2,

p_3 a q predstavujú názvy kvadrantov. Vidíme, že na elemente p_1 sa nám nepodarilo pixely zlúčiť, preto sme do stredu tohto elementu, rovnako ako do jeho vrcholov zapísali v poli L_1 hodnotu 1. Keď sa prejde na element p_2, program v prvom rade skontroluje či má v strede zapísane číslo 1. V tomto prípade má, čo znamená k zlučovaniu nedôjde, takže sa nehľadajú minimá ani maximá, rovno sa prejde do časti kódu, ktorá má na starosti nezlučovanie. Keďže k zlučovaniu nedochádza, do stredu a do vrcholov elementu p_2 zapíšeme 1. Podobne sa postupuje pre element p_3. Keďže má v strede hodnotu 1, zlúčiť ho nemôžeme, a tak opäť vykonáme potrebné zápisy do stredu a vrcholov tohto elementu. Takouto jednoduchou kontrolou sa vyhneme niektorým výpočtom, čo zohráva úlohu hlavne pri veľkých obrázkoch a ušetríme výpočtový čas.

1.3.4 Modifikácia mriežky - vyvážený kvadrantový strom

Doteraz popísaným spôsobom vytvárame všeobecnú mriežku. My ale z praktických dôvodov budeme používať vyvážený kvadrantový strom, čo znamená, že od ľubovoľných 2 susedných štvorčekov požadujeme, aby ich strany boli v pomere 1:2, 1:1 alebo 2:1. Treba teda nejakým vhodným spôsobom takýto pomer dosiahnuť. V skutočnosti sa nejedná o nič komplikované, stačí využiť fakt, že pri nezlučovaní detí elementu pošleme hodnoty 1 do rohov rodiča. Okrem toho, že vrcholy elementov na úrovni i sú stredmi niektorých elementov na úrovni $i-1$, pre niektoré elementy tejto istej úrovne môžu predstavovať stredy strán. Ak sa pozrieme na obrázok 1.3, vidíme, že pravý horný roh elementu p_1 predstavuje stred ľavej strany elementu q. Element q by ale vzniknuť nemal, pretože dĺžka jeho hrany by bola 4 pixely, a keďže sa element p_1 nezlúčil, tak je tvorený jednopixelovými elementami. Dostávame teda susedov s pomerom strán 4:1, čo nie je v súlade s tým čo chceme. Okrem doterajších podmienok treba pridať ešte jedno kritérium, ktoré rozhodne o zlúčení. Do časti programu, kde kontrolujeme, či sa v strede skúmaného elementu nachádza hodnota 1, vložíme ďalšie podmienky, ktoré skontrolujú, či sa i v stredoch jeho strán nenachádza číslo 1. Pokiaľ je aspoň v jednom strede strany zapísaná 1, v kóde sa prejde do vetvy, v ktorej sa nezlučuje. V opačnom prípade sa pokračuje v kóde. Ak by sme mali zhrnúť poradie kritérií, ktoré boli zatiaľ použité, potom pre nejaký element p, ktorého deti sa snažíme zlúčiť, postupujeme nasledovne:

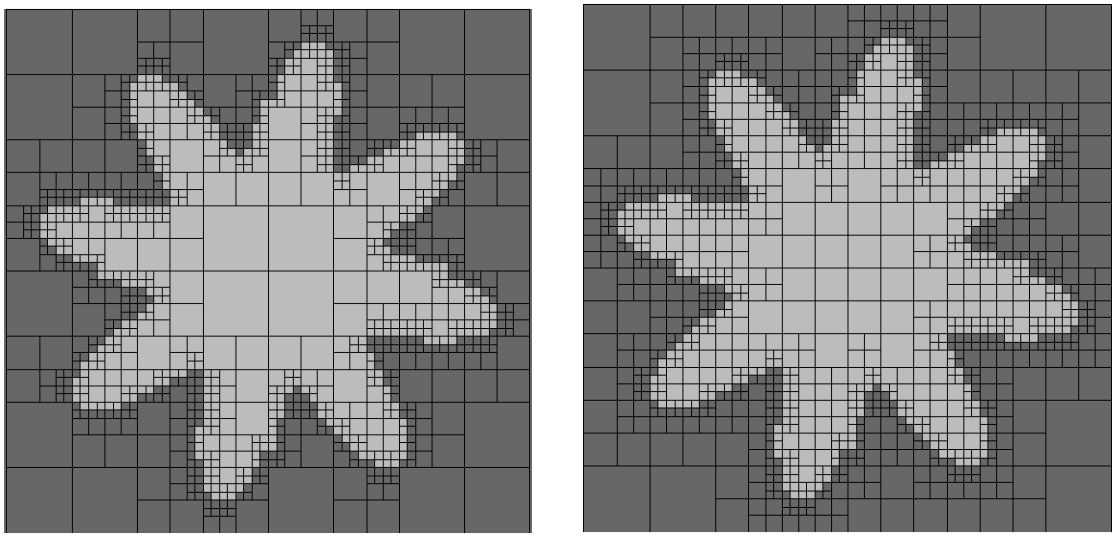
1. Skontroluj hodnoty v strede a na stredoch strán elementu p na príslušných

pozíciach v poli L_1 . Ak je aspoň jedna z týchto piatich hodnôt rovná 1, prejdi na časť programu v ktorej sa nezlučuje. Ak sú všetky hodnoty rovné nule, pokračuj ďaľej na bod 2.

2. Nájdi *min* a *max* elementu p a skontroluj, či je splnené kritérium z hľadiska intenzít farieb $|max - min| < \varepsilon_1$. Ak nie je, preskoč na časť programu v ktorej sa nezlučuje, v opačnom prípade pokračuj na bod 3.

3. Vypočítaj novú reprezentatívnu farbu pre element p ako priemer reprezentatívnych farieb jeho detí a zapíš ju na príslušné miesto do L_2 na pozíciu ľavého dolného rohu štvorčeka p. Taktiež si odpamätaj vypočítané *min* a *max* elementu p do poľa AUX.

Jedinou úlohou časti programu, ktorá ma na starosti nezlučovanie detí elementu p, je tá, že do stredu a do rohov štvorčeka p zapíše na príslušné pozície v poli L_1 jednotky. Takto je slovne popísaný algoritmus pre základnú mriežku, z ktorej budeme ďalej vychádzať. Neskôr ešte pribudnú ďalšie kritéria na zlúčenie, no dôvod ich zavedenia bude pomerne jasný aj z intuitívneho hľadiska.



(a) Všeobecný kvadrantový strom **(b)** Vyvážený kvadrantový strom

Obr. 1.4: Porovnanie adaptívnych mriežok

1.4 Zavedenie ďalších súčastí programu

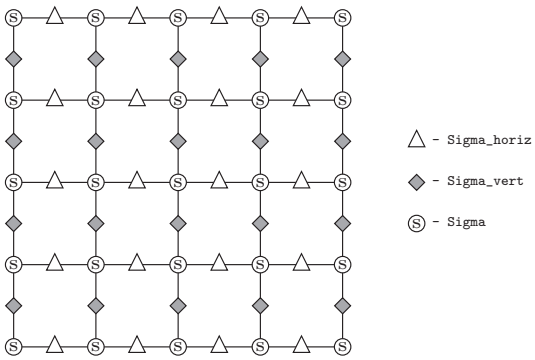
1.4.1 Zadefinovanie sigiem

Ako už bolo spomenuté, pri numerickom počítaní diferenciálnej rovnice pri väčšine metód potrebujeme pristupovať k okolitým hodnotám jednotlivých elementov oblasti. Sledujeme pritom nejaký tok intenzít farieb medzi časťami obrázka popísaný nejakou diferenciálnou rovnicou. Vzhľadom na to, že používame namiesto pravidelnej diskretizačnej siete adaptívnu, pristupovanie k susedným elementom už nie je také triviálne. V predchádzajúcom algoritme tento prístup uľahčuje zvolený rozmer polí L_1 a L_2 a požiadavka na vyváženosť kvadrantového stromu.

V posledných rokoch boli vytvorené algoritmy, ktoré využívajú stredy hrán štvorčekových elementov s tým, že mnohé operácie ako napríklad výpočet gradientu sa dajú urobiť na samotnom štvorčeku bez nutnosti pristupovať k susedom. Za týmto účelom si pre každý element zaznamenávame na každej jeho hrane špeciálne hodnoty, ktoré budeme nazývať sigmy. Nachádzajú sa na stredoch hrán elementov a ich úlohou je zachytiť akýsi vzťah medzi dvoma susednými elementami, ktoré túto hranu zdieľajú. Môžeme ich chápať ako hodnoty, ktoré pre každý element môžu charakterizovať nejaký prestup farebnej intezity medzi ním a jeho okolím cez jednotlivé jeho hrany. Hodnoty sigiem je možné vo výpočte vyeliminovať, v tom prípade však potrebujeme častejšie pristupovať k susedom. V predkladanom algoritme kombinujeme obe metódy - pre nerovnaké susedné štvorčeky využívame hodnoty sigiem a pre rovnaké ich z výpočtu eliminujeme. Jednou z výhod je, že takto zostavená rovnica má fixný počet členov - 1 diagonálny a 4 mimodiagonálne členy.

Na odpamätanie hodnôt jednotlivých sigiem používame špeciálne polia. Keďže potrebujeme mať uchované sigmy na stredoch hrán i pre jednopixelové elementy, zadefinovali sme pre tento účel 2 nové polia. Prvé z nich, *sigma_horiz*, uchováva stredy jednopixelových elementov na horizontálnych hranách mriežky, kdežto pole *sigma_vert* na vertikálnych hranách. Tretie pole s názvom *sigma* uchováva štvorice sigiem pre ostatné elementy o veľkosti strany 2 a viac.

Obrázok 1.5 ukazuje, ako by vyzerali spomenuté polia pre vstupný obrázok 4×4 pixelov. Vo všeobecnosti pre obrázok $n \times n$ potrebujeme, aby pole *sigma_horiz* malo veľkosť $(n+1) \times n$, pole *sigma_vert* veľkosť $n \times (n+1)$ a pole *sigma* definujeme o rozmeroch $(n+1) \times (n+1)$ prvkov. Takéto zadefinovanie nám bez problémov a



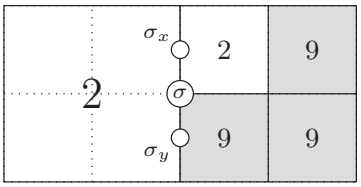
Obr. 1.5: Rozloženie polí pre sigmy

jednoznačne dokáže pre konkrétny element určiť všetky jeho 4 sigmy.

Samotné prvky týchto polí treba na začiatku, pred vytváraním mriežky, nejako naplniť. Pole *sigma_vert* sa inicializuje tak, že hodnoty na ľavej a pravej strane tohto poľa (na bočných hranách) naplníme samotnými hodnotami pixelov obrázka, na ktorých sigmy ležia (z dôvodu reflexie). Všetky ostatné sigmy z tohto poľa sú umiestnené na hrane, ktorú vždy zdieľajú 2 pixely obrázka. Naplnenie každej takejto sigmy potom spočíva v spriemerovaní hodnôt dvoch pixelov, medzi ktorými sa nachádzajú. Analogicky sa inicializuje aj pole *sigma_horiz*. Pole *sigma* je na počiatku naplnené nejakou zápornou hodnotou, napr. -10. Keďže polia *sigma_vert* a *sigma_horiz* máme v tomto momente už inicializované, využijeme ich hodnoty k naplneniu hodnôt poľa *sigma*. Z obrázka 1.5 si možno všimnúť, že na každej hrane nejakého väčšieho elementu sa nachádzajú 2 hodnoty sigiem jeho detí. Ich jednoduchým spriemerovaním získame hľadanú hodnotu sigmy rodičovského elementu pre konkrétnu hranu. Ak si teda z obrázka 1.5 vezmeme napríklad ľavý dolný element veľkosti 2×2 pixelov, potom hodnotu sigmy pre jeho dolnú hranu vypočítame jednoducho ako priemer sigiem na dolných hranách jeho dvoch dolných jednopixelových detí, teda ako priemer prvých dvoch prvkov z prvého riadku poľa *sigma_horiz*.

Zavádzame ďalšie kritérium pre zlučovanie elementov. Potrebujeme totiž, aby v oblastiach obrázka, kde je väčší rozdiel intenzít farieb ostala hustejšia sieť z dôvodu presnejšieho výpočtu. Spomínaný priemer sigiem vypočítame a zapíšeme len v prípade, ak je rozdiel 2 sigiem, ktoré chceme spriemerovať, menší ako nejaká predpísaná hodnota, ktorú označíme ε_2. Vzhľadom na to, že hodnoty pixelov sú vždy nezáporné, sigmy, ktoré vychádzajú z ich priemerov, sú taktiež nezáporné. Tu sa vysvetľuje dôvod, prečo

sme pole *sigma* na začiatku naplnili zápornými číslami. Samotné zlúčenie prebehne len v prípade, že sú všetky sigmy skúmaného elementu nezáporné. Ak je čo i len jedna jeho sigma záporná, znamená to že nedošlo k zapísaniu priemeru, t.j. aspoň pre jednu jeho hranu nie je splnená podmienka s ε_2, a teda k zlúčeniu nemôže dôjsť. Na obrázku



Obr. 1.6: Ukážka deliaceho kritéria pre sigmy

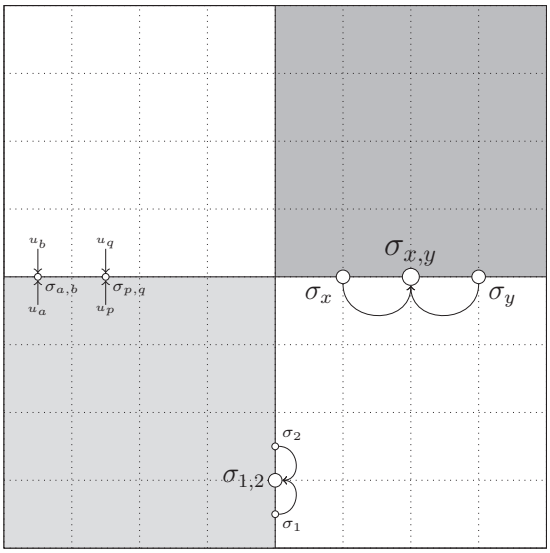
1.6 skúmame, či možno zlúčiť deti ľavého štvorčeka veľkosti 2×2 pixelov. Vidíme, že z hľadiska doterajších kritérií by k zlúčeniu dôjsť mohlo. Farebné intenzity jeho detí sú tie isté, rovné hodnote 2, rovnako pomer strán 2:1 by bol akceptovateľný. Hodnotu σ_x z obrázku vypočítame ako priemer hodnôt pixelov, medzi ktorými leží, čo je v tomto prípade 2. Podobne vypočítame σ_y spriemerovaním hodnoty 2 a 9, čo je rovné 5.5. Ak zvolíme napríklad $\varepsilon_2 = 3$, potom hodnotu sigmy σ na pravej strane skúmaného 2×2 štvorčeka by sme vypočítali spriemerovaním hodnôt σ_x a σ_y, t.j. čísel 2 a 5.5. Skôr, než budeme nejaký priemer počítať, je treba skontrolovať, či je rozdiel priemerovaných čísel σ_x a σ_y menší ako ε_2. Vidíme, že nerovnosť $5.5 - 2 < 3$ neplatí, takže sigma na pravej strane rodičovského štvorčeka kritériom neprejde a namiesto nezáporného priemeru tam ostane záporné číslo. Štvorček teda nezlúčime, pretože sigma na jeho pravej hrane bude záporná. Ak by sme takéto kritérium nezaviedli, štvorček by sa zlúčil. V takom prípade by ho cez jeho pravú hranu ovplyvňovali 2 štvorčeky s výrazne rozdielnymi hodnotami farby. Takto by sme mali problém popísať množstvo situácií, medzi ktorými je napríklad správne vystihnutie rozmazania hrán, ktoré by malo vzniknúť pri použití rovnice vedenia tepla, najmä v prípade, že by išlo o susediace štvorčeky rôznej veľkosti.

Ďalej zavedieme pomocné pole POR o veľkosti $n \times n$. Po vytvorení mriežky používame na prechádzanie stromu rekurzívny algoritmus, ktorý prehľadáva kvadrantový strom do hĺbky. To znamená, že každý element mriežky sa pri jednom prechádzaní navštívi/spracuje práve raz. Význam poľa POR spočíva v tom, že si v ňom pre jednotlivé elementy na pozíciu ľavého dolného rohu odpamätávame poradie, v ktorom sa daným štvorčekom

pri prechádzaní stromu prešlo. Získame tak pre každý element jedinečné poradové číslo. V prípade, že si bude treba pamätať pre štvorček viacero údajov, budeme vyhradzovať jednorozmerné polia, ktorých dĺžka bude zodpovedať počtu elementov. Vďaka pomocnému poľu POR máme jednoznačný vzťah medzi použitými dvojrozmernými a jednorozmernými poliami. Toto môžeme využiť napríklad pri odpamätávaní gradientov do poľa $GRAD$ pre každý element. Ak napríklad uvažujeme nejaký element, ktorý má na ľavom dolnom rohu v poli POR uloženú hodnotu k, potom má gradient uložený v poli $GRAD$ na mieste s indexom k. Naopak, nejaký k-tý prvok poľa $GRAD$ zodpovedá veľkosti gradientu toho elementu mriežky, ktorý bol k-tý v poradí pri prechádzaní stromu. Takýto prístup nám okrem práce s gradientom neskôr tiež umožní pomerne jednoducho vytvoriť sústavu lineárnych rovníc.

1.4.2 Posledné zlučovacie kritérium

Ako sme spomenuli, z hľadiska matematického výpočtu je niekedy vhodné vytvoriť v okolí hrán - častí obrázka s rôznou intezitou farby, čo najjemnejšiu sieť, napríklad keď riešime difúziu pomocou rovnice vedenia tepla. Vo veľkej miere k tomu napomáha posledné kritérium s kontrolou sigiem, no v praxi sme zistili, že pre rovnicu vedenia tepla nezachytí úplne všetky potrebné prípady, čo sa môže pri numerickom počítaní negatívne prejaviť na výstupných obrázkoch. Zavádzame teda ešte jedno posledné kritérium, ktoré tento problém pomôže vyriešiť.



Obr. 1.7: Posledné deliace kritérium a ukážka výpočtu sigiem

K ukážke takéhoto problematického príkladu nám poslúži obrázok 1.7. Možno na ňom vidieť, aká mriežka sa vytvorí pri použití doterajších kritérií pre daný vstupný obrázok rozmerov 8×8 pixelov. Výsledná sieť bude pozostávať zo 4 kvadrantov veľkosti 4×4 pixelov. Skutočne, program nemá dôvod ponechať nejaké ďalšie delenie, pre každý kvadrant totiž platí, že intenzita farieb jeho vnútorných pixelov je rovnaká, že so všetkými susednými elementami má pomer strán 1:1 a že sigma na každej jeho hrane prejde podmienkou na zápis. V obrázku vidíme, ako sa dopracujeme k výpočtu sigmy pre element veľkosti 4×4. V konečnom dôsledku jej hodnotu (v obrázku $\sigma_{x,y}$) vypočítame ako aritmetický priemer sigiem na tej istej hrane menších štvorčekov (v obrázku σ_x a σ_y). Tieto 2 hodnoty sú však rovnaké, pretože boli vypočítané spriemerovaním sigiem jednopixelových elementov, ktorých hodnoty máme uložené v poli *sigma_horiz* (na obrázku je tento proces znázornený pomocou $\sigma_{1,2}$). Každá jednopixelová sigma tejto vodorovnej hrany má ale rovnakú hodnotu, pretože vznikla spriemerovaním tých istých dvoch farieb (na obrázku $\sigma_{a,b}$ a $\sigma_{p,q}$ majú rovnakú hodnotu). Z toho dôvodu bude vždy rozdiel dvoch priemerovaných sigiem nulový, čo je menšie ako hodnota ε_2, o ktorej predpokladáme že je kladná. Ak by sme počítali napríklad rovnicu vedenia tepla, intuitívne očakávame akési rozmazanie hrán jednotlivých farebných oblastí. To, čo pri použití takejto mriežky získame, je len pomalá zmena farieb jednotlivých 4 kvadrantov, čo veľmi nezodpovedá očakávaniam. Vec, ktorá by určite pomohla, je zjemnenie siete v okolí hrán týchto štyroch elementov. Na to sme zaviedli kritérium, ktoré kontroluje rozdiel hodnôt sigiem elementu s hodnotou jeho reprezentatívnej farby. Ak je aspoň jeden z týchto 4 rozdielov väčší, ako nejaká predpísaná hodnota ε_3, potom zlúčenie nepovolíme. Vďaka nesplneniu podmienky vieme, že farba elementu sa výrazne líši od nejakej časti jeho okolia, čo znamená, že niekde v jeho blízkosti sa nachádza hrana, pri ktorej potrebujeme jemné delenie. Zlúčenie detí nejakého elementu sa teda povolí len v prípade, ak sú hodnoty všetkých sigiem zhruba rovnaké ako reprezentatívna farba štvorčeka, pričom ich rozdiel môže byť maximálne ε_3. Takto je vytvorená výsledná adaptívna mriežka, s ktorou sa ďalej počíta a na ktorej budeme zostavovať systém lineárnych rovníc.

1.5 Zhrnutie prvej kapitoly

K tvorbe mriežky sme použili niekoľko zlučovacích kritérií. Zhrňme si teda, aké parametre môžu okrem samotného počiatočného obrázka ovplyvniť vzhľad výslednej mriežky:

- ε_1 (v programe `eps1`) - Hodnota, ktorá predpisuje maximálny možný rozdiel farebných intenzít pixelov na elemente. Zväčšovaním tejto hodnoty kladieme menšie nároky na rovnakosť farieb pri rozhodovaní o zlučovaní.

- ε_2 (v programe `eps2`) - Využíva sa pri zlučovaní z hľadiska sigiem. Ak inicializujeme sigmy pre nejaký rodičovský element priemerovaním sigiem jeho detí, k zlúčeniu dôjde len v prípade že sú tieto priemerované sigmy v rozmedzí predpísanom hodnotou ε_2.

- ε_3 (v programe `eps4`) - Ohraničuje maximálny možný rozdiel intenzity farby elementu s jednotlivými hotnotami jeho sigiem. Znižovaním tejto hodnoty, rovnako ako hodnoty ε_2 dostávame hustejšiu sieť v okolí hrán, čo je dôležité z hľadiska výpočtu najmä pri riešení rovnice vedenia tepla, ktorý sa snažíme na mriežke realizovať.

Popis implementácie základného algoritmu s použitím dvojrozmerného kódovania sú popísané v [1]. Oproti tomuto algoritmu sú v tejto práci nasledovné zmeny:

1. sú použité stredy strán štvorčekov, tzv. *sigmy*, ako je popísané v texte, pričom v prípade nerovnakých susedných štvorčekov je ich rozptyl riadený parametrom ε_2

2. celkovo je tvorba mriežky riadená väčším počtom parametrov

3. je vyhradené pole POR, kde sa kóduje poradie prechodu a štvorčeky sa číslujú, čo je dôležité pre ukladanie informácii o elementoch

Pri štandardných neadaptívnych numerických schémach sa v prvom kroku sigmy nastavujú na priemer hodnôt elementov, ktoré ich zdieľajú - tak, ako je popísané v tomto algoritme. V ďalšom časovom kroku sa využívajú hodnoty sigiem vypočítaných počas riešenia lineárneho systému. V predkladanom algoritme zatiaľ nie je vytvorený v jednotlivých časových krokoch prechod zo starej mriežky na novú s využitím hodnôt sigma z aktuálneho časového kroku. Program teraz v každom časovom kroku vytvára mriežku

a počíta sigmy nanovo z výstupného obrázka predošlého časového kroku, čo je korektné len pre rovnicu vedenia tepla. Napriek tomu sme urobili aj niekoľko experimentov pre krivostný filter.

2 Matematické výpočty na mriežke

2.1 Rovnica vedenia tepla

Ako základ pre ďalšie výpočty vezmeme klasickú rovnicu vedenia tepla s nulovou pravou stranou. Budeme teda uvažovať počiatočno-okrajovú úlohu tvaru:

$$\partial_t u(t,x) - \Delta u(t,x) = 0 \quad \text{na } Q_T \equiv I \times \Omega, \tag{2.1.1a}$$

$$\partial_{\vec{\nu}} u(t,x) = 0, \quad \forall x \in I \times \partial\Omega, \tag{2.1.1b}$$

$$u(0,x) = u_x^0, \ \forall x \in \Omega, \tag{2.1.1c}$$

kde $\Omega \subset R^2$ je priestorová oblasť, v našom prípade obrázok ako taký, $I = [0,T]$ je časový interval, na ktorom počítame, rovnica 2.1.1b je okrajová podmienka a rovnica 2.1.1c predstavuje počiatočnú podmienku.

2.1.1 Diskretizácia výpočtovej oblasti

Použitá schéma vychádza z metódy konečných objemov s použitím štvorcových elementov, na ktorých bude konštantná hodnota riešenia. Predpokladajme najprv pravidelnú mriežku štvorcov - konečných objemov, zvyčajne označovaných malými písmenami p a q. Označenie ε_p bude predstavovať množinu všetkých hrán konečného objemu p. Veľkosť hrany σ budeme značiť ako $|\sigma|$. Ak označíme x_p ako stred konečného objemu p a stred jeho hrany σ ako x_σ, potom kolmú vzdialenosť stredu elementu od jeho hrany budeme značiť $d_{p\sigma} = |x_\sigma - x_p|$ a vektor $n_{p,\sigma}$ predstavuje vonkajši normálový vektor elementu p na hrane σ. Označením $m(p)$ budeme rozumieť mieru elementu p, čo bude v našom prípade obsah štvorčeka. Hodnota k reprezentuje veľkosť zvoleného časového kroku. Na časovú deriváciu použijeme aproximáciu pomocou spätnej diferencie a deriváciu v smere vektora $n_{p,\sigma}$ aproximujeme ako

$$\nabla u \cdot n_{p\sigma} \approx \frac{\left(u_\sigma^{n+1} - u_p^{n+1}\right)}{d_{p\sigma}}. \tag{2.1.2}$$

Základná rovnica, ktorú odvodíme pre rovnicu vedenia tepla pomocou metódy konečných objemov a vyššie uvedeného označenia potom vyzerá nasledovne:

$$\left(\frac{m(p)}{k} + \sum_{\sigma\in\varepsilon_p} \frac{|\sigma|}{d_{p\sigma}}\right) u_p^{n+1} - \sum_{\sigma\in\varepsilon_p} \frac{|\sigma|}{d_{p\sigma}} u_\sigma^{n+1} = \frac{m(p)}{k} u_p^n \tag{2.1.3a}$$

$$\frac{m(p)}{k}\left(u_p^{n+1} - u_p^n\right) = \sum_{\sigma\in\varepsilon_p} \frac{|\sigma|}{d_{p\sigma}} (u_\sigma^{n+1} - u_p^{n+1}). \tag{2.1.3b}$$

Obe rovnice 2.1.3b aj 2.1.3a sú tie isté, len prepísané v inom tvare. Prvý tvar sa používa pri riešení a v druhom sú vyjadrené toky cez jednotlivé hranice. Z rovnosti tokov medzi 2 elementami p a q ďalej vyplýva, že

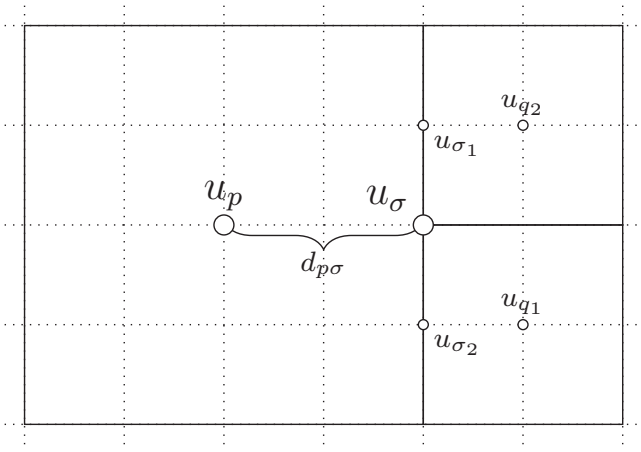
$$(u_\sigma - u_p) = (u_q - u_\sigma) \tag{2.1.4a}$$

$$u_\sigma = \frac{u_p + u_q}{2}. \tag{2.1.4b}$$

Po dosadení výrazu 2.1.4b napríklad do rovnice 2.1.3a eliminujeme členy obsahujúce σ a dostaneme upravený tvar

$$\left(\frac{m(p)}{k} + \sum_{q\in N(p)} 1\right) u_p^{n+1} - \sum_{q\in N(p)} u_q^{n+1} = \frac{m(p)}{k} u_p^n, \tag{2.1.5}$$

pričom označením $N(p)$ sa myslí množina všetkých susedných elementov skúmaného štvorčeka p.



Obr. 2.1: Susedné elementy s nerovnakou veľkosťou

Ak ďalej rozšírime schému o to, že pripustíme elementy nerovnakej veľkosti, potom podľa obrázka 2.1 pre hodnotu u_σ máme vzťah:

$$u_\sigma = \frac{u_{\sigma_1} + u_{\sigma_2}}{2}. \tag{2.1.6}$$

Použijúc opäť myšlienku o zachovaní tokov musí platiť, že to, čo pritečie/vytečie cez pravú hranu elementu p musí pritiecť/vytiecť cez ľavé hrany elementov q_1 a q_2. A teda dostávame vzťah:

$$(u_\sigma - u_p) = (u_{q_1} - u_{\sigma_1}) + (u_{q_2} - u_{\sigma_2}). \tag{2.1.7}$$

Po spojení 2.1.6 a 2.1.7 dostaneme:

$$\begin{aligned} \tfrac{3}{2}u_{\sigma_1} + \tfrac{3}{2}u_{\sigma_2} &= u_{q_1} + u_{q_2} + u_p && \text{(2.1.8a)} \\ \tfrac{u_{\sigma_1}+u_{\sigma_2}}{2} &= \tfrac{1}{3}(u_{q_1} + u_{q_2} + u_p) && \text{(2.1.8b)} \\ u_\sigma &= \tfrac{1}{3}(u_{q_1} + u_{q_2} + u_p) && \text{(2.1.8c)} \end{aligned}$$

V rovnici 2.1.3b sa na pravej strane sumuje cez všetky 4 hrany elementu p. Ak má skúmaný štvorček nerovnakého suseda, tvar sčítanca sumy pre túto hranu môžeme upraviť dosadením vzťahu 2.1.8c:
$\frac{|\sigma|}{d_{p\sigma}}(u_\sigma^n - u_p^n) = 2(u_\sigma^n - u_p^n) = 2(\frac{1}{3}(u_{q_1}^n + u_{q_2}^n + u_p^n) - u_p^n) = \frac{2}{3}(u_{q_1}^n + u_{q_2}^n + u_p^n) - 2u_p^n = \frac{2}{3}u_{q_1}^n + \frac{2}{3}u_{q_2}^n - \frac{4}{3}u_p^n = \frac{2}{3}(u_{q_1}^n - u_p^n) + \frac{2}{3}(u_{q_2}^n - u_p^n)$
Schému zo vzťahu 2.1.3b môžeme pomocou predošlého vyjadrenia použitého v [2] zapísať nasledovne:

$$(u_p^{n+1} - u_p^n)m(p) = k \sum_{q \in N(p)} T_{pq}(u_q^{n+1} - u_p^{n+1}) \tag{2.1.9a}$$

$$T_{pq} = \begin{cases} 1, & \text{ak má susedný element rovnakú veľkosť} \\ \frac{2}{3}, & \text{inak} \end{cases} \tag{2.1.9b}$$

Napokon zvoľme

$$\begin{aligned} u_{\sigma_1} &= \tfrac{1}{3}u_p + \tfrac{2}{3}u_{q_1} && \text{(2.1.10a)} \\ u_{\sigma_2} &= \tfrac{1}{3}u_p + \tfrac{2}{3}u_{q_2} && \text{(2.1.10b)} \end{aligned}$$

Po dosadení 2.1.10a a 2.1.10b do 2.1.6 dostaneme vzťah $u_\sigma = \frac{1}{2}u_{\sigma_1} + \frac{1}{2}u_{\sigma_2} = \frac{1}{3}(u_{q_1} + u_{q_2} + u_p)$, čo je zhodné zo vzťahom 2.1.8c, a teda pre takto zvolené sigmy bude rovnosť tokov zachovaná aj medzi susednými elementami s nerovnakou veľkosťou.

Ak zostavujeme rovnicu 2.1.3a, resp. 2.1.3b pre element p, potom v prípade rovnakých susedov použijeme v sume sčítanca v tvare rovnice 2.1.5. V prípade odlišnej

veľkosti susedných štvorčekov stačí, ak pomocou poľa L_1 zistíme, že sused nie je rovnakej veľkosti, ako skúmaný element p. Namiesto toho, aby sme ďalej zisťovali niečo o tomto susedovi (či už presnú veľkosť alebo hodnotu), siahame práve na vypočítanú sigmu danej hrany elementu p a susedom sa ďalej nezaoberáme. Sčítanec ponecháme v tvare rovnice 2.1.3a, resp. 2.1.3b.

2.2 Krivostný filter

Uvažujme schému pre krivostný filter. Rovnica pre konečný objem p je podľa [3] nasledovná:

$$\left(\frac{m(p)}{kf_p^n} + \sum_{\sigma\in\varepsilon_p}\frac{|\sigma|}{d_{p\sigma}f_p^n}\right)u_p^{n+1} - \sum_{\sigma\in\varepsilon_p}\frac{|\sigma|}{d_{p\sigma}f_p^n}u_\sigma^{n+1} = \frac{m(p)}{kf_p^n}u_p^n \tag{2.2.1}$$

$$\frac{m(p)}{f_p^n}\frac{u_p^{n+1}-u_p^n}{k} - \sum_{\sigma\in\varepsilon_p}\frac{|\sigma|}{d_{p\sigma}}\frac{1}{f_p^n}\left(u_\sigma^{n+1}-u_p^{n+1}\right) = 0, \tag{2.2.2}$$

kde f_p^n je veľkosť gradientu elementu p z časového kroku n a platí

$$f_p = \sqrt{|\nabla u_p|^2+\varepsilon_4^2} \tag{2.2.3}$$

$$|\nabla u_p|^2 = \frac{1}{m(p)}\sum_{\sigma\in\varepsilon_p}\frac{|\sigma|}{d_{p\sigma}}\left(u_\sigma-u_p\right)^2, \tag{2.2.4}$$

čo pri štvorčeku s dĺžkou strany h môžeme zjednodušiť na vzťah

$$|\nabla u_p|^2 = \frac{2}{h^2}\sum_{\sigma\in\varepsilon_p}\left(u_\sigma-u_p\right)^2. \tag{2.2.5}$$

Pri výpočte použijeme podobnú úvahu ako pri rovnici vedenia tepla, pričom dostaneme nasledovné vzorce pre toky cez hranice konečného objemu p:

$$\text{Sused } q \text{ tej istej veľkosti:} \quad \frac{2}{f_p^n+f_q^n}\left(u_q^{n+1}-u_p^{n+1}\right) \tag{2.2.6}$$

$$\text{Sused } q \text{ nerovnakej veľkosti:} \quad \frac{2}{f_p^n}\left(u_\sigma^{n+1}-u_p^{n+1}\right), \tag{2.2.7}$$

kde hodnota u_{σ_1} podľa obrázka 2.1 bude aktualizovaná vzorcom

$$u_{\sigma_1}^{n+1} = \frac{f_{q_1}^n u_p^{n+1} + 2f_p^n u_{q_1}^{n+1}}{f_{q_1}^n + 2f_p^n} \tag{2.2.8}$$

Ak zvolíme veľkosť gradientu pre každý element rovný 1, potom dostaneme presne vzťah 2.1.3a, teda rovnicu vedenia tepla. Pri numerickom počítaní s touto rovnicou

navyše nepotrebujeme posledné kritérium s hodnotu ε_3, pretože táto rovnica pracuje na rozdiel od rovnice vedenia tepla na báze krivosti. Má vyrovnávací vplyv na hranách v častiach obrázka, kde je hodnota krivosti vysoká. Z tohto dôvodu tu nedostávame problém, ktorý je popísaný v sekcii 1.4.2. Ak použijeme vstupný obrázok 1.7, nestane sa s obrázkom pre túto rovnicu nič, pretože sú tu len vodorovné a zvislé rovné hrany, ale žiadne zakrivenie. Na riadenie presnosti výpočtu nám teda stačia hodnoty ε_1 a ε_2.

2.3 Sústavy rovníc

Pre každý element nám vznikne rovnica, v konečnom dôsledku teda dostávame sústavu m rovníc, pričom m je počet elementov. Neznáme sú hodnoty pixelov, no v prípade nerovnakých susedných štvorčekov sú to aj sigmy. Neznámych teda môžeme mať viac ako rovníc, preto pridávame ešte jednu sústavu, ktorá sa rieši spolu s prvou. V programe vieme presne určiť, koľko prípadov nerovnakých susedov sa nám v mriežke vyskytlo, a teda vieme vopred presne povedať, aké množstvo pamäte treba alokovať. Ako je vyššie spomenuté, pre každý takýto prípad používame tvar toku podľa rovnice 2.1.3b, teda pristúpime k hodnote sigmy. Akonáhle sa nejaká sigma dostane medzi neznáme hlavnej sústavy, vytvárame v druhej sústave trojicu ďalších rovníc pre neznámu sigmu. Ak nastane situácia, že skúmaný štvorček p má menších susedov, ako je zaznačené na obrázku 2.1, potom do druhej sústavy pri počítaní pre krivostný filter pridáme trojicu rovníc:

$$u_{\sigma_1}^{n+1} = \frac{f_{q_1}^n u_p^{n+1} + 2 f_p^n u_{q_1}^{n+1}}{f_{q_1}^n + 2 f_p^n} \tag{2.3.1}$$

$$u_{\sigma_2}^{n+1} = \frac{f_{q_2}^n u_p^{n+1} + 2 f_p^n u_{q_2}^{n+1}}{f_{q_2}^n + 2 f_p^n} \tag{2.3.2}$$

$$u_\sigma = \frac{1}{2} u_{\sigma_1} + \frac{1}{2} u_{\sigma_2} \tag{2.3.3}$$

Pri rovnici vedenia tepla položíme všetky hodnoty gradientov rovné 1, čím sa nám vzťahy 2.3.1, 2.3.2 a 2.3.3 zjednodušia. Obe sústavy máme uložené v poliach štruktúr, pričom prvky oboch sústav majú fixný počet elementov a preto sa ľahko spracovávajú. Na rozdiel od predkladaného algoritmu, v [2] sa pri nerovnakých susedoch zisťovali informácie o susednom elemente aj z väčšieho štvorčeka na menší, aj naopak. Počet susedov sa tam mohol pohybovať od 2 do 8 a bolo treba si ich odpamätať alebo ich špeciálne spracovať.

Každý riadok našej matice sústavy má nenulové vždy maximálne 4 prvky, vznikne nám teda riedka matica, ktorá je naviac aj diagonálne dominantná. Na riešenie môžeme použiť iteračnú Gauss-Seidelovu metódu. Na zrýchlenie konvergencie riešenia sme sa rozhodli použiť SOR metódu.

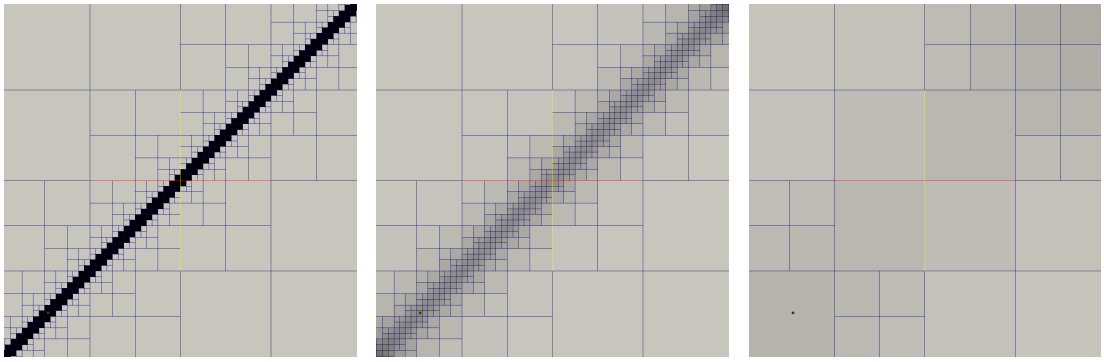
2.4 Popis hlavného programu

1. Deklarácia potrebných premenných + alokovanie dvojrozmerných polí
2. Načítanie/vytvorenie vstupného obrázka
3. Začiatok časového cyklu
    - Vytvorenie mriežky pomocou procedúry `grid()`
    - Po zistení počtu elementov sa alokujú jednorozmerné polia
    - Jednotlivé elementy sa očíslujú do poľa POR
    - Zistí sa počet nerovnakých hrán a alokuje sa pole pre druhú sústavu rovníc
    - V prípade riešenia level-set rovnice sa vypočítajú gradienty
    - Zostaví sa sústava rovníc a následne sa iteračne rieši
    - Vytvorí sa názov súboru a konečný výstup časového kroku sa uloží ako VTK súbor [4] (program má vytvorené aj funkcie na načítanie vstupných obrázkov a zápis výstupných obrázkov formátu PGM [5])
    - Prejde sa na nový krok časového cyklu

3 Prezentácia výsledkov

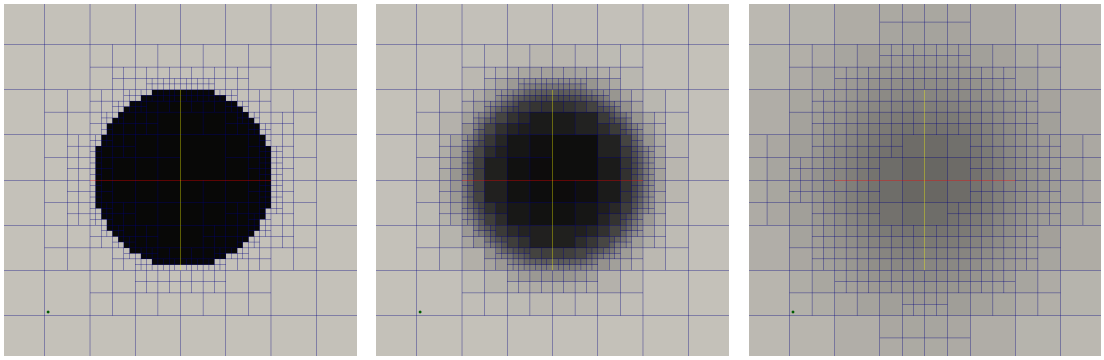
3.1 Rovnica vedenia tepla

Ako prvé sme modelovali rovnicu vedenia tepla na vstupnom obrázku rozmerov $2^N \times 2^N$. Je dôležité hodnoty ε_1, ε_2, ε_3 a časový krok nadstaviť tak, aby sme našli vhodný pomer rýchlosti výpočtu a kvality výstupného obrázka. Ak zvolíme napríklad hodnoty ε príliš malé, dostaneme síce presnejší výsledok, no výpočtový čas sa môže zväčšiť až natoľko, že program pracujúci na pravidelnej mriežke nám zbehne rýchlejšie a teda vytváranie adaptívnej mriežky v takom prípade stratí význam. Nasledovné ukážky prezentujú výstupy programu pre rôzne vstupné obrázky s parametrami $N = 6$, $tau = 10$, $\varepsilon_1 = 0.1$, $\varepsilon_2 = 0.025$ a $\varepsilon_3 = 0.02$.



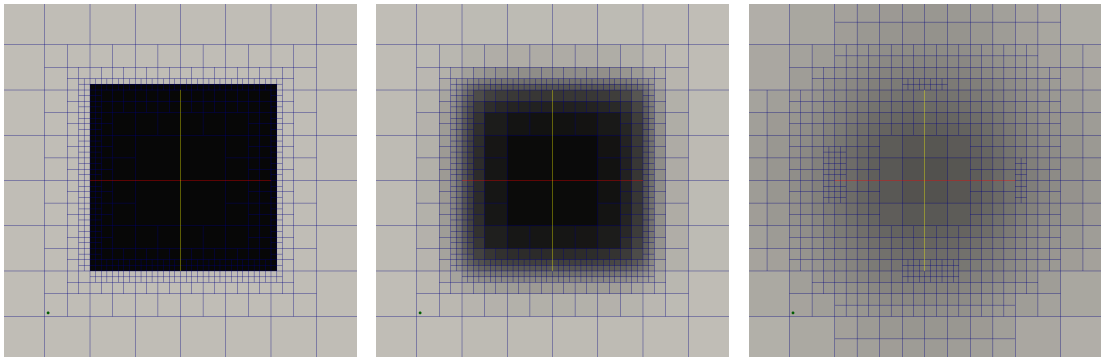
(a) Mriežka na vstupe **(b)** Po prvom časovom kroku **(c)** Po 10 časových krokoch

Obr. 3.1: Ukážka na obrázku diagonálnej čiary



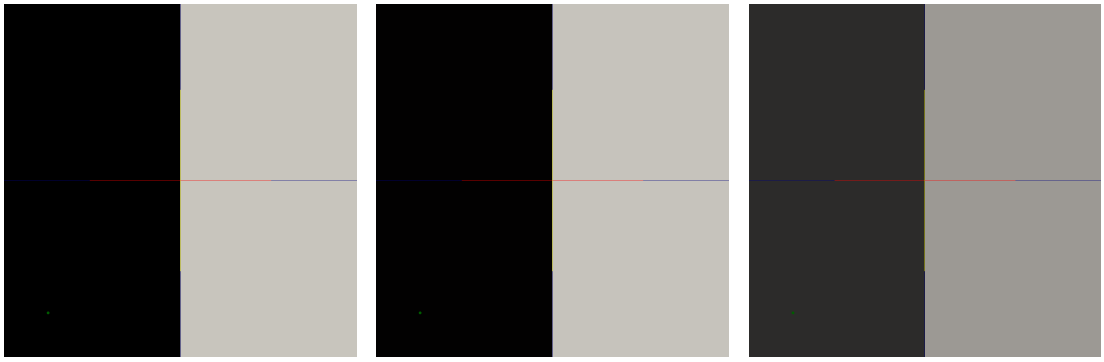
(a) Mriežka na vstupe **(b)** Po prvom časovom kroku **(c)** Po 10 časových krokoch

Obr. 3.2: Ukážka na obrázku kruhu



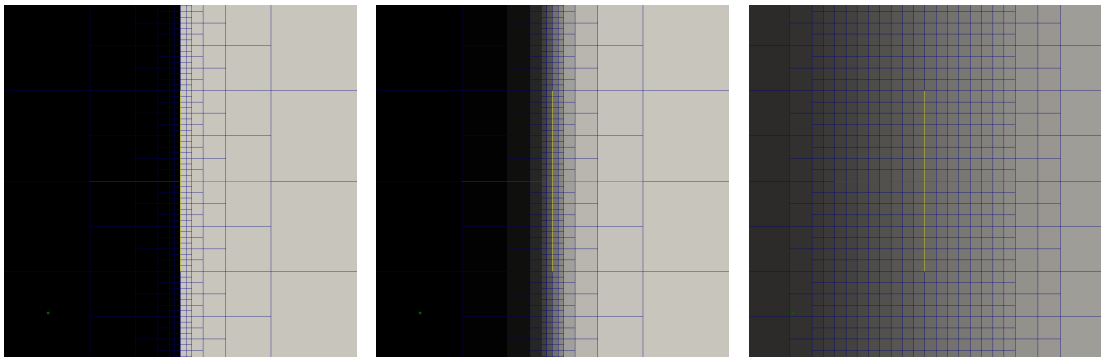
(a) Mriežka na vstupe **(b)** Po prvom časovom kroku **(c)** Po 10 časových krokoch

Obr. 3.3: Ukážka na obrázku štvorčeka



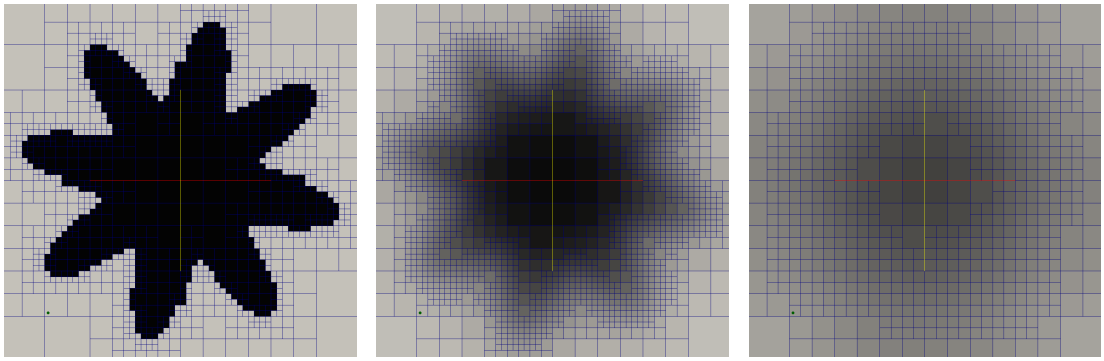
(a) Vstupný obrázok **(b)** Po prvom časovom kroku **(c)** Po 30 časových krokoch

Obr. 3.4: Ukážka problematického prípadu z časti 1.4.2 bez použitia ε_3



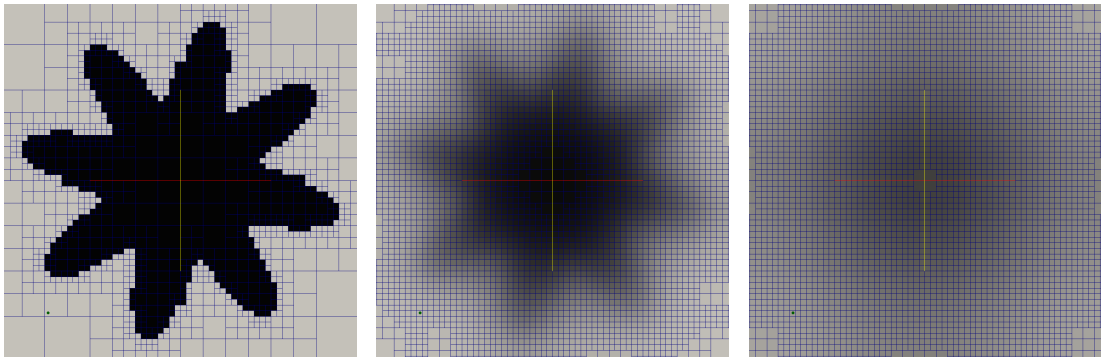
(a) Vstupný obrázok **(b)** Po prvom časovom kroku **(c)** Po 30 časových krokoch

Obr. 3.5: Rovnaký obrázok s použitím posledného kritéria s $\varepsilon_3 = 0.02$



(a) Vstupný obrázok **(b)** Po 2 časových krokoch **(c)** Po 10 časových krokoch

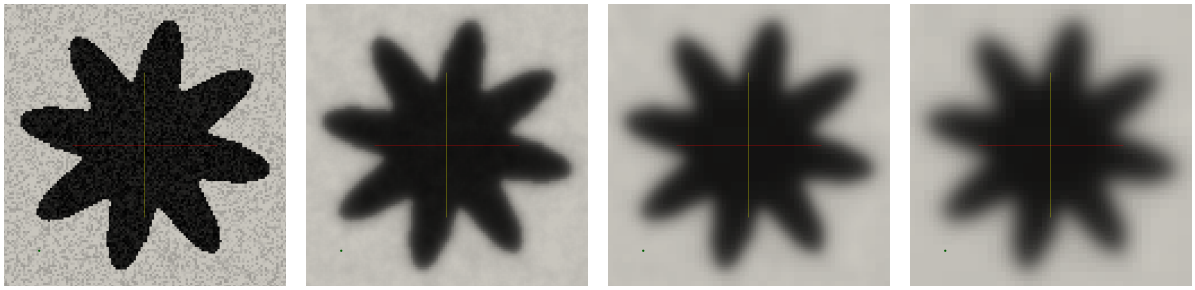
Obr. 3.6: Obrázok tzv. zubáča s použitím $\varepsilon_3 = 0.05$



(a) Vstupný obrázok **(b)** Po 2 časových krokoch **(c)** Po 10 časových krokoch

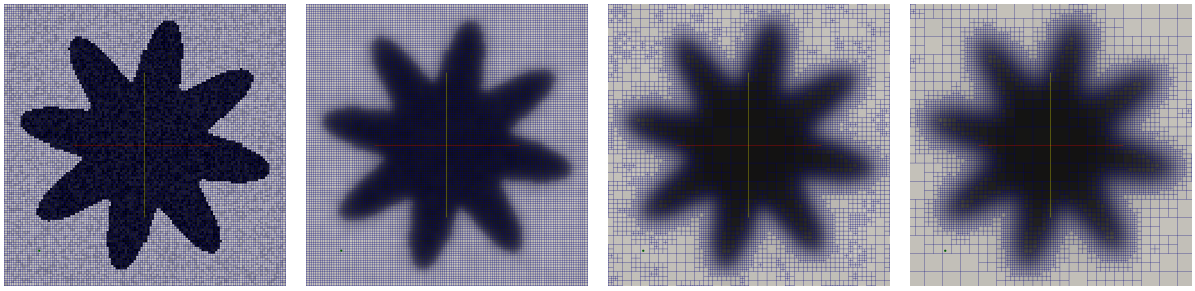
Obr. 3.7: Rovnaký vstupný obrázok s príliš malým $\varepsilon_3 = 0.005$

Rovnicu vedenia tepla môžeme použiť aj na odstránenie aditívneho šumu v obrázku. Nasledujúca séria obrázkov skúma úbytok elementov počas behu programu. Použili sme obrázok zubáča o veľkosti $2^7 \times 2^7$, $tau = 5$, $\varepsilon_1 = 0.1$, $\varepsilon_2 = 0.02$, $\varepsilon_3 = 0.05$.



(a) Vstupný obrázok **(b)** 1 časový krok **(c)** 2 časové kroky **(d)** 3 časové kroky

Obr. 3.8: Odstraňovanie aditívneho šumu v prvých 3 časových krokoch



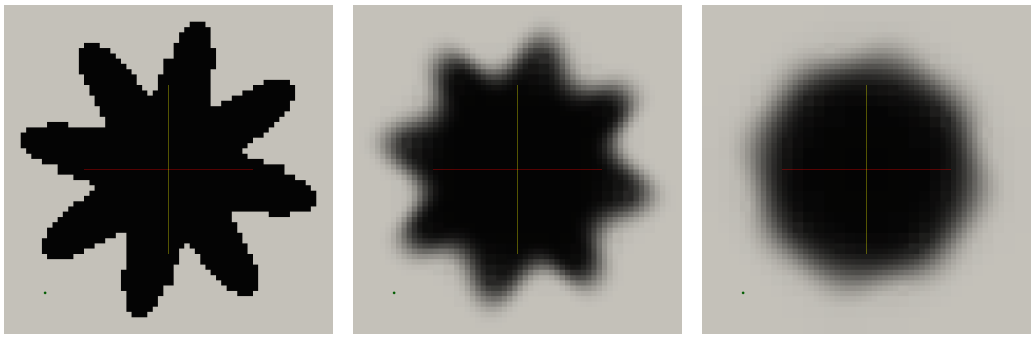
(a) 16378 elementov **(b)** 16378 elementov **(c)** 8104 elementov **(d)** 6577 elementov

Obr. 3.9: Úbytok elementov počas prvých 3 časových krokov

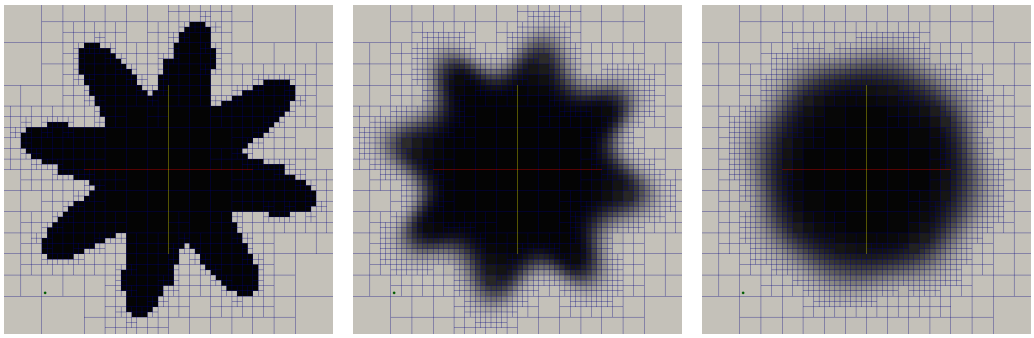
Z obrázkov vidíme, že šum bol odstránený už po prvom časovom kroku.

3.2 Krivostný filter

Na modelovanie nasledujúcich obrázkov nebolo použité posledné kritérium s hodnotou ε_3. Počiatočný obrázok sme zvolili veľkosti $2^6 \times 2^6$ a program spustili s hodnotami $tau = 10$, $\varepsilon_1 = 0.1$ a $\varepsilon_2 = 0.02$.

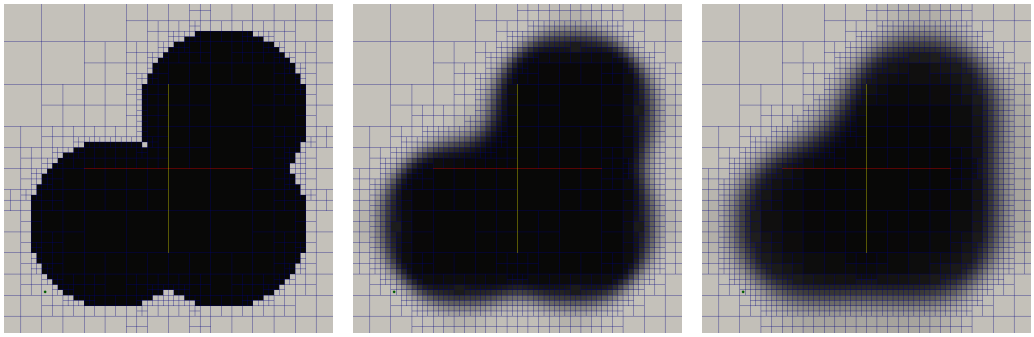


(a) Vstupný obrázok **(b)** Po 5 časových krokoch **(c)** Po 10 časových krokoch



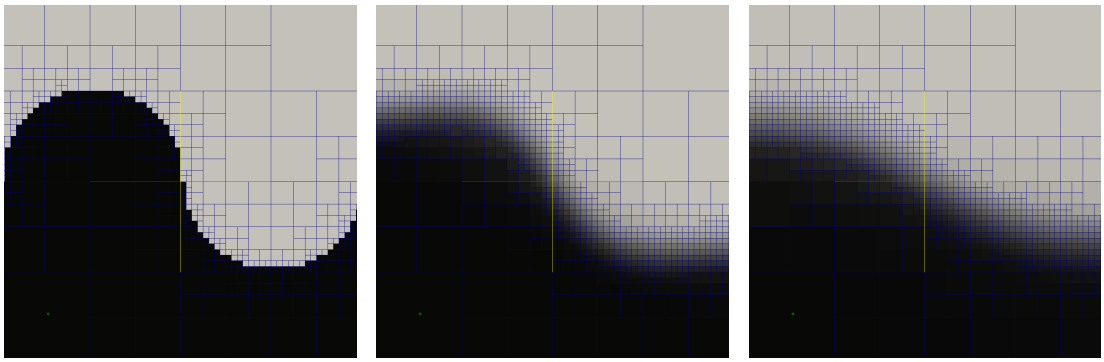
(d) Vstupný obrázok **(e)** Po 5 časových krokoch **(f)** Po 10 časových krokoch

Obr. 3.10: Krivostný filter a príslušné mriežky



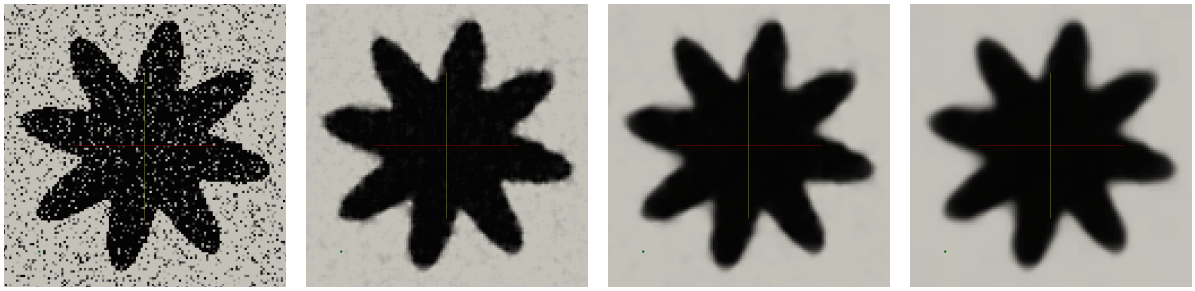
(a) Vstupný obrázok **(b)** Po 5 časových krokoch **(c)** Po 10 časových krokoch

Obr. 3.11: Krivostný filter pre obrázok 3 kruhov



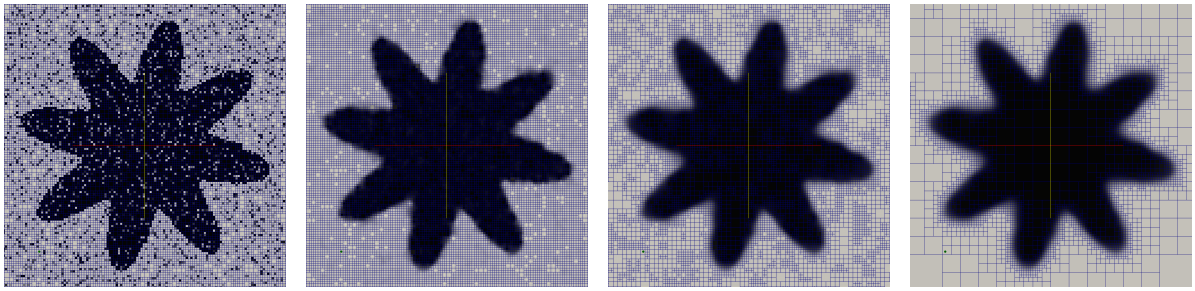
(a) Vstupný obrázok **(b)** Po 20 časových krokoch **(c)** Po 50 časových krokoch

Obr. 3.12: Krivostný filter pre obrázok vlnky



(a) Vstupný obrázok **(b)** 1 časový krok **(c)** 2 časové kroky **(d)** 3 časové kroky

Obr. 3.13: Odstraňovanie šumu *salt and pepper* v prvých 3 časových krokoch pre pre $N = 7$, $tau = 5$, $\varepsilon_1 = 0.1$, $\varepsilon_2 = 0.02$



(a) 15148 elementov **(b)** 15148 elementov **(c)** 9820 elementov **(d)** 4606 elementov

Obr. 3.14: Príslušné mriežky a úbytok elementov zašumeného obrázka počas prvých 3 časových krokov

Ciele ďalšieho výskumu

Cieľom ďalšieho výskumu bude urobiť vybudovanie mriežky v novom časovom kroku na základe mriežky z predchádzajúceho časového kroku a implementovať algoritmy so zložitejším výpočtom gradientov a difúznych koeficientov, prípadne skúsiť výpočty pre iné diferenciálne rovnice. V budúcnosti v programe môže pribudnúť príjemnejšie užívateľské prostredie a rozšírenie o možnosť pracovať s 3D dátami.

Záver

V práci je predstavená a podrobne popísaná metóda na skonštruovanie adaptívnej mriežky, ktorej hustota sa dá zo strany užívateľa ovládať niekoľkými parametrami. Na konkrétnych príkladoch je tu vysvetlených niekoľko rôznych problematických prípadov a k nim navrhnuté riešenia, ktoré sme v programe použili. V druhej kapitole je matematicky popísaná rovnica vedenia tepla, level set rovnica pre krivostný filter a ich diskrétny tvar, s akým sa ďalej pracuje a pomocou ktorého vytvárame sústavu lineárnych rovníc. Program sme implementovali v programovacom jazyku C, pričom výstupy z jednotlivých časových krokov sú ukladané do formátu VTK, ktorý sa dá ľahko vizualizovať napríklad v softvéri *ParaView* [6]. Kapitola 3 prezentuje dosiahnuté výsledky pre rovnicu vedenia tepla a taktiež ponúka niekoľko ukážok pre Level set rovnicu.

Literatúra

[1] KRIVÁ Z., *Adaptive Finite Volume Methods in Image Processing*, Edícia vedeckých prác, Zošit č. 15, Bratislava, 2004

[2] KRIVÁ Z., *Numerical Schemes In Image Processing*, Edícia vedeckých prác, Zošit č. 97, Bratislava, 2011

[3] EYMARD R., HANDLOVIČOVÁ A., MIKULA K., *Study of a finite volume scheme for the regularized mean curvature flow level set equation*, IMA Journal of Numerical Analysis Vol. 31, No. 3, pp. 813-846

[4] VTK File Formats, *http://www.vtk.org/VTK/img/file-formats.pdf*

[5] Reading/Writing PGM Files In C, *http://ugurkoltuk.wordpress.com/2010/03/04/an-extreme-simple-pgm-io-api*, 2010

[6] KRIVÁ Z., PROKOPEC P., *Vykresľovanie adaptívnych mriežok v programe ParaView*, v MAGIA 2010: Mathematics, Geometry and their applications, Conference Proceedings, Kočovce, November 2010