SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Stavebná fakulta

Zobrazovanie kvadrantových a oktantových stromov vo vizualizačnom softvéri ParaView

Bakalárska práca

Študijný program: Matematicko-počítačové modelovanie
Študijný odbor: Aplikovaná matematika

Vedúci bakalárskej práce:
RNDr. Zuzana Krivá, PhD.

Autor práce:
Tomáš Zachar

Bratislava 2008

Čestné prehlásenie

Vyhlasujem, že som bakalársku prácu vypracoval samostatne s použitím citovanej literatúry a s odbornou pomocou vedúceho práce.

Bratislava 23.05.2008 . . . .

Vlastnoručný podpis

1. Obsah

1. Obsah 4
2. Úvod a motivácia 7
   - 2.1. ParaView 7
   - 2.2. Ciele bakalárskej práce 7
3. Základy stromov a programovania 8
   - 3.1. Kvadrantové a oktantové stromy 8
   - 3.2. Príklad vytvorenia stromov 10
   - 3.3. Kvadrantové stromy v ParaView 11
   - 3.4. Programovanie 11
   - 3.5. Základy rekurzie 11
   - 3.6. Programovací jazyk 12
4. Program 12
   - 4.1. Jadro programu 12
   - 4.2. Základy pre obrázky (2D štruktúry) 13
   - 4.3. Výstup pre ParaView a rôzne dátové štruktúry 13
   - 4.4. Tvorba základného výstupu v programe 17
   - 4.5. Optimalizácia vrcholov 20
   - 4.6. Úrovňovanie (stratový výstup) 20
   - 4.7. Atribúty a farebná paleta 21
   - 4.8. Prahovanie 23
   - 4.9. Spracovanie objemových telies (3D štruktúry) 24
   - 4.10. Doplnky programu 25
5. Užívateľské pracovanie s programom 26
   - 5.1. Rozhranie 26
   - 5.2. Možnosti 27
6. Záver 28
   - 6.1. Záver 28
   - 6.2. Poďakovanie 28
7. Súhrn 29
8. Summary 29
9. Použitá Literatúra 30

2. Úvod a motivácia

2.1. ParaView

Vizualizácia je súčasťou nášho každodenného života, prepracované predpovede počasia a dokonalá grafika zábavného priemyslu sú len jednými z mála príkladov. V tejto práci pod vizualizáciou budeme chápať transformáciu dát alebo informácie do obrázkov. Vizualizácia využíva potenciál ľudského mozgu pre spracovanie obrázkovej informácie akokoľvek zložitej aj objemnej a súčasne môže byť považovaná za efektívny prostriedok komunikácie. Je často nevyhnutná k tomu, aby sme vedeli spracovať obrovské množstvo informácie, ktoré dnešné počítače dokážu poskytnúť a zrozumiteľne ju sprostredkovať druhým. Jedným z nástrojov, ktorý poskytuje prostriedky pre spracovanie dvojrozmerných a trojrozmerných dát a vizuálne ich interpretovať rozumným spôsobom, je ParaView [1]. Má zabudované nástroje, napríklad na zobrazovanie kontúr (tzv. izočiary v 2D a izoplochy v 3D), rezov, vektorových polí, alebo vytváranie animácii. Je založený na objektovo orientovanej knižnici ***VTK*** [2], ktorej vstupný formát dát vtk ďalej budeme využívať a čiastočne ho popíšeme.

Typickým príkladom vizualizácie v ParaView môže byť napríklad spracovanie dát získaných v nemocnici počítačovým tomografom. Obyčajne to býva skupina dvojrozmerných obrázkov, z ktorých sa po načítaní do ParaView [4] vytvorí objem dát a pomocou funkcie pre zobrazovanie izoplôch môžeme dať zobraziť napríklad lebku, kožu, zuby a podobne.

2.2. Ciele bakalárskej práce

Cieľ tejto bakalárskej práce je nasledovný. Máme vstupné dáta, zadané na pravidelnej mriežke, ktoré vie ParaView načítať automaticky. Našou úlohou je tieto dáta zapísať úspornejším spôsobom, ktorý je založený na metóde kvadrantových a oktantových stromov. Ďalšou hlavnou úlohou je výstup programu konvertovať na vstup do ParaView – t.j. nájsť vhodné dátové formáty a dáta v týchto formátoch úsporne zapísať. Súčasťou bakalárskej práce bolo aj porovnanie týchto formátov, preskúmanie vizualizačných možností, ktoré ponúkajú a preskúmanie práce s vlastnou farebnou paletou, ktorá nie je, alebo je nedostatočne dokumentovaná v dostupnej literatúre.

Podstatnou časťou bakalárskej práce je program, ktorý je schopný:

- vytvoriť kvadrantový strom a zobraziť ho
- vytvoriť vstup pre ParaView
- tento vstup optimalizovať
- úrovňovať
- pracovať s farebnou paletou
- robiť prahovanie
- vytvoriť jednoduchú štatistiku

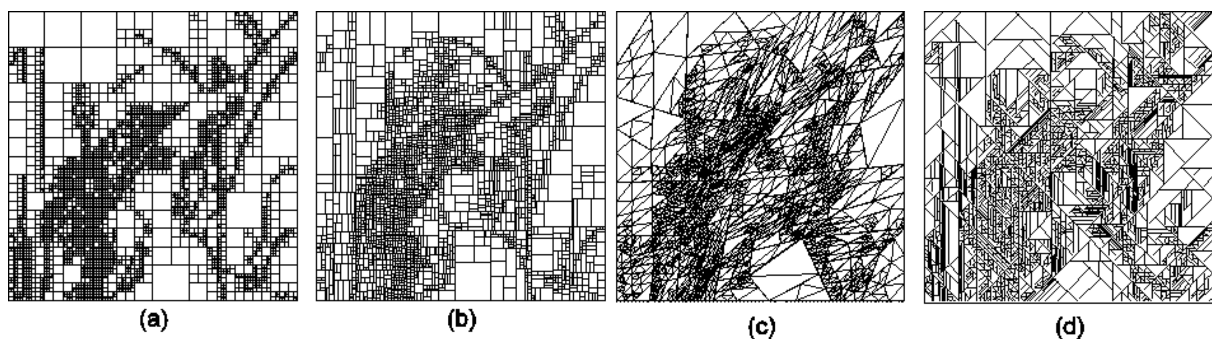
3. Základy stromov a programovania

3.1. Kvadrantové a oktantové stromy

Častou úlohou v praxi býva rozdeliť zadanú oblasť na menšie útvary - napríklad trojuholníky, obdĺžniky, štvorce tak, aby sa pretínali iba v hranách a vrcholoch. Súbor takýchto útvarov – elementov, budeme nazývať mriežka. Vo všeobecnosti rozlišujeme dva základné typy mriežok [3]:

- pravidelné (štruktúrované, structured)
- nepravidelné (neštruktúrované, unstructured)

Príkladmi nepravidelných mriežok sú: HV (horizontálno-vertikálne) delenie (3.1b), trojuholníkové delenie (3.1c), polygonálne delenie (3.1d) a nakoniec naše kvadrantové (3.1a) delenie [5].

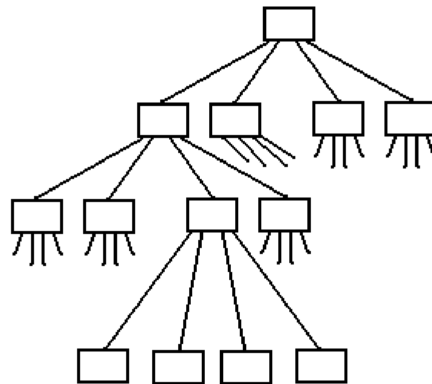


Obr. 3.1

Pravidelnou mriežkou je napríklad mriežka štvorcová. Okrem hraníc má každý štvorček štyroch susedov, ktorých pozície sa dajú vypočítať [3]. V súlade s terminológiou popisu vstupného formátu vtk, budeme takéto mriežky nazývať štruktúrované.

V nepravidelnej mriežke pozície elementov nemožno vypočítať, ale musia byť explicitne uložené a vyžadujú viac počítačovej pamäte. Navyše počet susedov sa môže meniť. Jej častou výhodou býva, že lepšie charakterizuje popisovanú oblasť. Takéto nepravidelné mriežky sa často nazývajú aj adaptívne mriežky. Na obrázku 3.1 sú uvedené príklady neštruktúrovaných mriežok.

Kvadrantový strom alebo ***Quadrant Tree***, či ***Quadtree*** je rekurzívne delenie dátovej oblasti veľkosti 2^N x 2^N na štyri časti veľkosti 2^{N-1} x 2^{N-1}. Toto delenie je riadené ukončovacou podmienkou [3]. Našou ukončovacou podmienkou bude rovnaká farba pre celú časť alebo rozmer 1x1 (jeden pixel obrázku, jedna farba). Takýmto delením vzniká štruktúra, ktorá sa v teórii grafov nazýva strom. (Obr. 3.2)



Obr. 3.2

Najvyšší prvok stromu sa nazýva ***koreň,*** alebo vrchol a posledné – koncové prvky sa volajú ***listy***. Každá podčasť stromu sa nazýva ***podstrom***. Podprvok sa nazýva ***syn*** a nadprvok ***otec***. V našom kvadrantovom strome má každý prvok práve štyroch alebo nula synov. Každý prvok ma práve jedného otca, okrem koreňa. Prvky nachádzajúce sa vedľa seba budeme nazývať susedné a jeden riadok prvkov budeme nazývať vrstvou alebo stupňom. Dĺžkou stromu nazývame počet stupňov stromu. V našom prípade (v podstate skoro vždy) nebudú listy len v poslednej vrstve a strom bude ukončovaný v rôznych prvkoch.

Oktantový strom alebo ***OctTre***, sa využíva pri trojrozmerných dátach. Oktantový znamená osmitý, teda tak ako sme 2D obrázok/dáta delili na štyri štvorce, 3D teleso/dáta budeme podobným systémom deliť na osem častí, teda osem kociek.

Kvadrantové a oktantové stromy sa obťažnejšie používajú na dáta, ktoré nemajú štvorcový charakter, teda ich veľkosť nie je mocnina dvojky, teda 2, 4, 8, 16... Obrázok sa však môže na takýto rozmer doplniť, napríklad čiernou farbou. Takto doplnené časti väčšinou tvoria štvorce/kocky väčších rozmerov a ich spracovanie je rýchle.

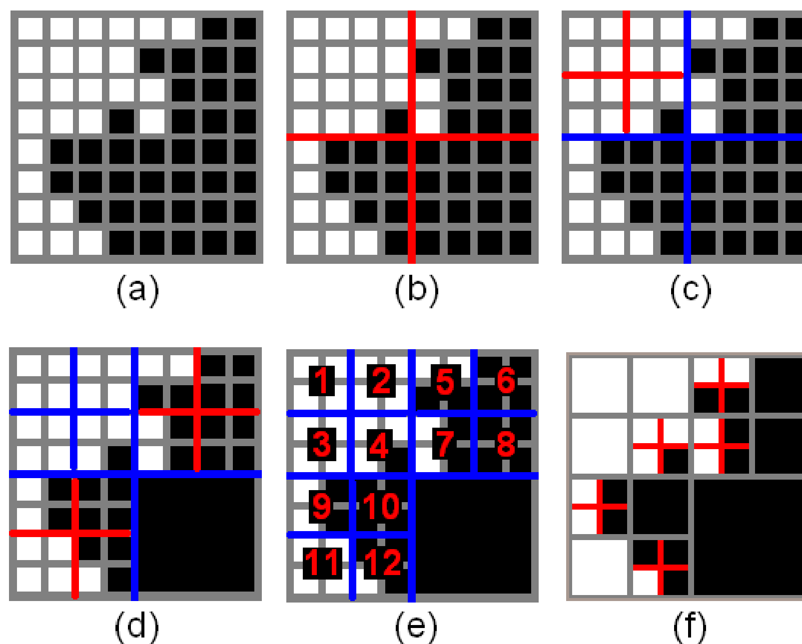
3.2. Príklad vytvorenia stromov

Najprv si ukážeme obrázok pred rozdelením (3.3a), potom po rozdelení (3.3b).

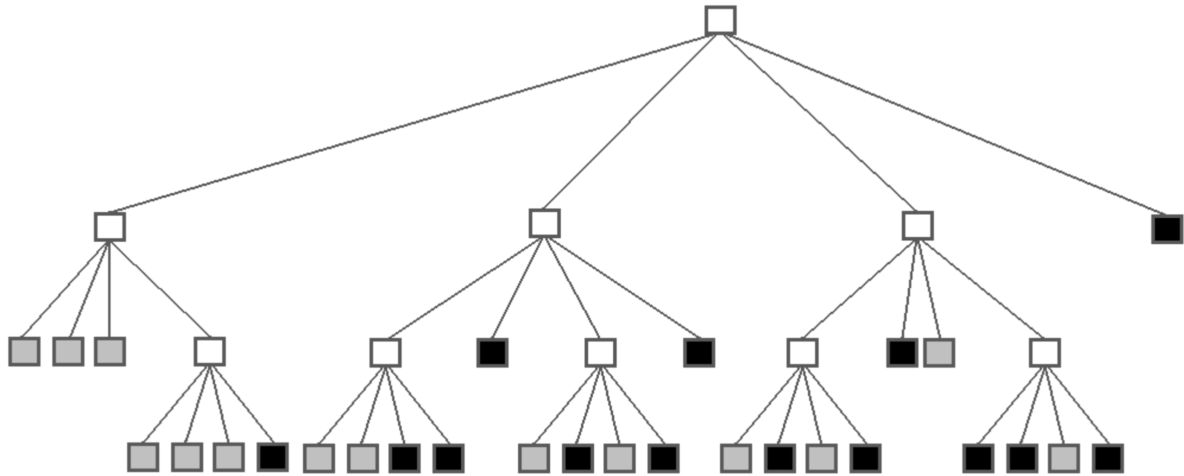
Obr. 3.3

Obrázok 3.3 má rozmer 8x8, kde osem je tretia mocnina dvojky. To, že osem je tretia mocnina dvojky zároveň napovedá, že maximálny stupeň (dĺžka) stromu bude 3+1.

Obrázok 3.4a rozdelíme (červenou čiarou) na štyri rovnaké časti (3.4b). Ak prvá časť (vľavo hore) nepozostáva z jednej farby, znova ju rozdelíme (3.4c). Rovnako postupujeme pre zvyšné tri časti (3.4d). Teraz namiesto 16 častí máme len 13, pretože pravú dolnú časť sme nerozdelili, keďže pozostávala z jednej farby. Ďalej treba preskúmať ešte 12 častí a v prípade nutnosti ich rozdeliť. Posledný obrázok 3.4f prezrádza, že sme rozdelili ešte päť častí, a teda spolu nám vzniklo 27 častí. Na obrázku 3.5 je znázornená stromová štruktúra pre obrázok 3.4f.



Obr. 3.4



Obr. 3.5

3.3. Kvadrantové stromy v ParaView

Našim cieľom je spraviť program, ktorý spraví kvadrantový strom pre výstup z dát zadaných na pravidelnej štvorcovej mriežke a spraví vstup do ParaView.

3.4. Programovanie

Počítač nie je človek, nemá oči a myslenie ako my, a to je náš základný problém. Počítač nedokáže z pohľadu povedať či daná časť pozostáva len z jednej farby alebo z viacerých. Musí si prejsť každý jeden pixel v danej časti a až pri poslednom vie vyhodnotiť, či časť pozostáva len z jednej farby. V lepšom prípade počas skúmania farieb narazí na inú farbu a môže sa hneď rozhodnúť pre ďalšie rozdelenie. Počítač je síce rýchly, no takýto postup je naozaj pomalý. Týmto spôsobom kontrolujeme niektoré body viackrát, v najhoršom prípade až ***n*** krát, kde ***n*** je dĺžka stromu mínus jedna (v predchádzajúcom príklade trikrát). To možno tiež nie je veľa, no čo ak obrázok je ***3D*** teleso s rozmermi 1024*1024*1024? Potom n=10 (2^{10}=1024), pixelov bude takto jedna miliarda a ak má väčšinu prejsť desaťkrát, tak to už je naozaj aj pre lepšie počítače dosť. My budeme náš obrázok deliť vždy na štyri rovnaké časti, až kým sa nedostaneme na najmenšiu veľkosť a postupne smerom naspäť budeme zisťovať, či sú oblasti tvorené jednou farbou. Oba postupy sa dajú spraviť cez rekurziu, no druhý spôsob je oveľa šikovnejší.

3.5. Základy rekurzie

Rekurzia je pojem [6], ktorý vysvetlíme na rekurzívnej funkcii. Je to funkcia, ktorá volá samu seba vo vlastnom tele. Typický príklad na vysvetlenie je faktoriál. Faktoriál je matematická funkcia, ktorá z nejakého kladného čísla ***n*** vráti číslo ***n*(n-1)*(n-***

2)*...*(2)*(1).Teda pre číslo 3 to bude 6, pre 4 to bude 24 a pre 5 to bude 120. Majme funkciu ***faktorial(i)*** kde ***i*** je číslo. Telo funkcie by vyzeralo takto:

```
Faktorial(i);
Begin
  result := faktorial(i-1)*i;
End;
```

Takto naprogramovaná funkcia by sa nikdy nezastavila, dokonca akonáhle by raz dosiahla nulu, tak by už bola stále iba nulová. Preto každá rekurzívna funkcia musí mať nejakú zastavovaciu podmienku. Po pridaní zastavovacej podmienky bude mať funkcia takýto tvar:

```
Faktorial(i);
Begin
  if (i>1) then
    result := faktorial(i-1)*i
  else
    result := 1;
End;
```

3.6. Programovací jazyk

Je viacej programovacích jazykov. Pre nás sú zaujímavé viac-menej len dva: ***C*** a ***Pascal***. Každý z nich má svoje pre aj proti. Jazyk C je veľmi rýchly, no veľmi zložitý a často označovaný ako „read only language" – teda je problém čítať kód po niekom inom, alebo po dlhšej dobe. Naopak, Pascal je jazyk veľmi jednoduchý, ľahko čitateľný, no bohužiaľ, pomalší. Pre jednoduchosť a čitateľnosť sme si zvolili programovací jazyk ***Pascal*** a prostredie ***Delphi***.

4. Program

4.1. Jadro programu

Jadrom celého programu je rekurzívna funkcia. Je to jednoduchá funkcia, ktorá má na vstupe štvorcovú oblasť a stupeň (hĺbku rekurzie) a ako výstup vracia farbu, alebo hodnotu, ktorá hovorí, že oblasť pozostáva z viacerých farieb. V tele funkcie nájdeme podmienku, či sme na najnižšom stupni (oblasti zodpovedá jeden pixel), alebo sme vo vyššom stupni. V prípade, že je program na najnižšom stupni, vráti farbu pixelu. V opačnom prípade sa funkcia odvolá sama na seba štyrikrát pre štyri štvorcové podoblasti. Nakoniec, v prípade, ak sú všetky štyri farby rovnaké, vráti túto farbu, ak nie, tak vráti informáciu, že farby sú rôzne. Výstup údajov sa vytvára vo funkcii ***vystup_udajov()***.

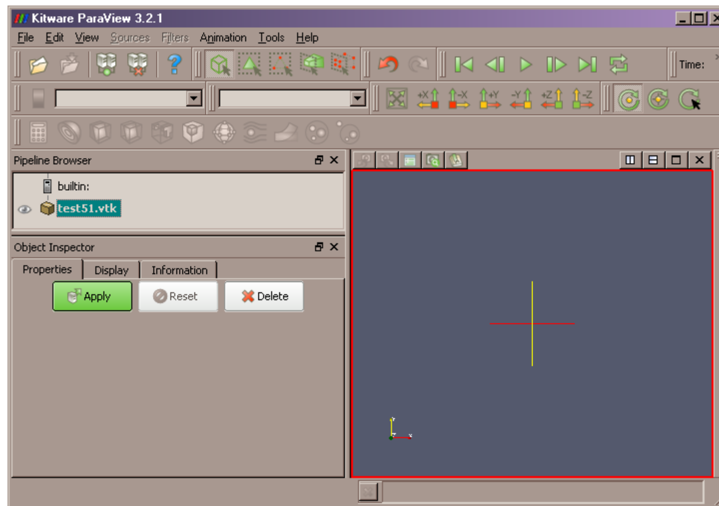
```
Jadro(o: oblast;stupen:integer):integer;
Begin
  If (stupen=0) then
    farba := farba_pixlu(o)
  Else Begin
    f1 := Jadro(hornaLava_podoblast_z(o),stupen-1);
    f2 := Jadro(hornaPrava_podoblast_z(o),stupen-1);
    f3 := Jadro(dolnaLava_podoblast_z(o),stupen-1);
    f4 := Jadro(dolnaPrava_podoblast_z(o),stupen-1);
    If (rovnaka_farba(f1,f2,f3,f4)) then
      farba := f1
    Else
      Vystup_udajov(o,f1,f2,f3,f4);
      farba := rozne_farby;
    End;
  Result := farba;
End;
```

4.2. Základy pre obrázky (2D štruktúry)

V našom programe pre kvadrantové stromy sú hlavným vstupom klasické obrázky vo formáte ***BMP*** alebo ***TXT*** súbory, ktoré obsahujú rozmer a údaje o dátach (tu farby). Ako už bolo spomínané, dáta by mali byť štvorcového rozmeru, kde dĺžka strany je prirodzenou mocninou dvojky. My v programe načítaný obrázok prevedieme vždy do dvojrozmerného poľa farieb. Farba každého pixelu je reprezentovaná číslom od 0 po 2147483648, kde nula je čierna a maximálna hodnota je biela. Teda pole farieb bude vlastne pole čísiel. Delphi má svoje vlastné funkcie, ktoré dokážu z každého tohto čísla určiť zložku červenej, zelenej a modrej farby. Možno ešte ako zaujímavosť je dobré dodať, že tento program pri spustení načíta automaticky obrázok ***defin.bmp***.

4.3. Výstup pre ParaView a rôzne dátové štruktúry

Výstup pre ParaView bude v našom prípade jednoduchý textový súbor s príponou ***vtk*** kódovaný v ASCII formáte ***vtk***. Súbor pozostáva z hlavičky a dát. V hlavičke sa nachádzajú štyri údaje: verzia ***vtk*** formátu (verzia 1.0 a 2.0 sú kompatibilné s verziou 3.0, ktorú budeme používať my), vlastný komentár, formát súboru (***ASCII***, ***BINARY***) a typ štruktúry dát. Štruktúru dát si podrobnejšie vysvetlíme nižšie. V našom prípade veľkosť súboru závisí najmä od toho ako často sa opakujú niektoré farby, teda či obrázok/dáta obsahuje väčšie plochy jednej farby. Takýto súbor už v ParaView otvoríme jednoducho, spustíme si ParaView, v hlavnej ponuke nájdeme ***file*** a tam ***open***, ďalej už len nájdeme a vyberieme náš súbor. Po otvorení sa však obrazec nezobrazí (ParaView verzia 3.2.1), treba ešte kliknúť na zelené tlačidlo ***apply*** (ako je na obrázku 4.1).



Obr. 4.1

Ako už bolo spomenuté, v ParaView môžeme zobrazovať viac typov dátových štruktúr. My sa budeme zaoberať len štyrmi typmi:

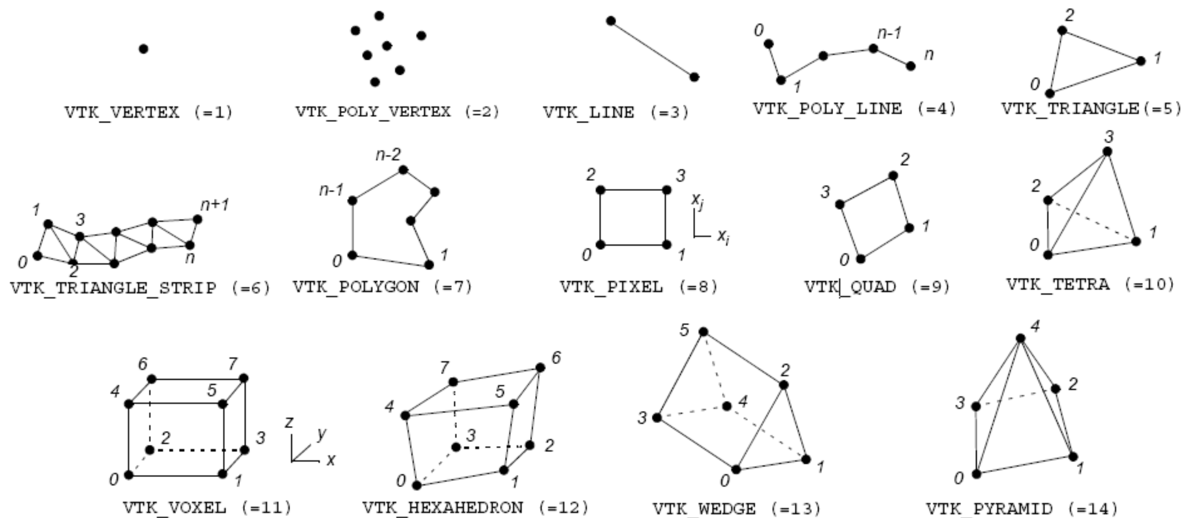
- ***unstructured gri***d (neštruktúrovaná mriežka) - ***VTK_QUAD***,
- ***unstructured gri***d **-** ***VTK_POLYGON***,
- ***unstructured gri***d **-** ***VTK_HEXAHEDRON*** (pre 3D)
- ***polygonal data***

Navzájom sú veľmi podobné, takže si rozoberieme jednu z nich, konkrétne ***VTK_POLYGON,*** a potom už len uvedieme rozdiely. Na základe zvoleného typu dátovej štruktúry bude výstup pozostávať z piatich častí/sekcií: vrcholy, bunky (štvorce), typy buniek a atribúty, ktorými sú zväčša farby a farebná paleta. Prehľad súborov vtk formátu:

```
# vtk DataFile Version 3.0
náš popis alebo komentár súboru
ASCII alebo BINARY
DATASET type
...
POINT_DATA n
...
CELL DATA n
...
Atribúty...
```

(Príklad pozri strana 19)

Každá z tých sekcií pozostáva z dvoch ďalších častí: hlavička a telo. Pre zaujímavosť si môžeme ukázať na obrázku 4.2 všetky formáty neštruktúrovanej mriežky.



Obr. 4.2, [7]

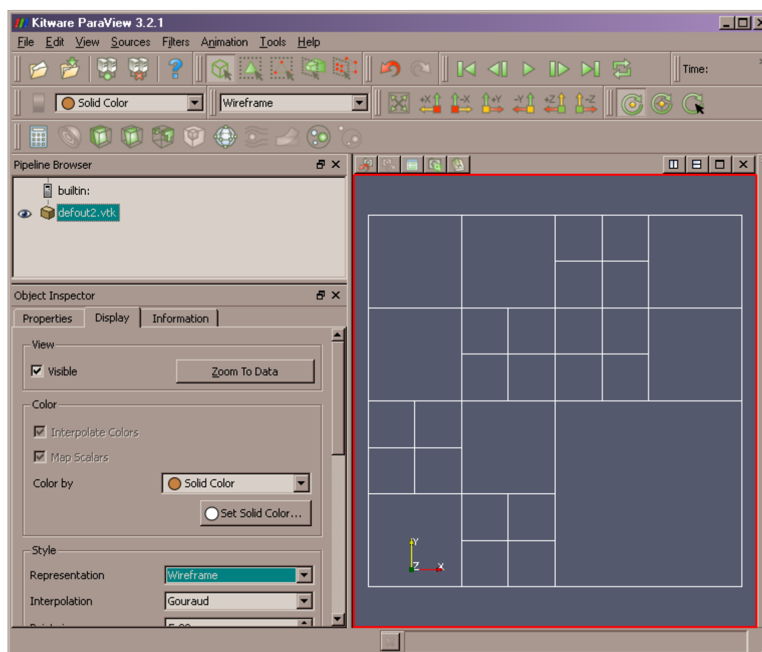
Hlavička vrcholov obsahuje slovo ***POINTS,*** potom počet vrcholov, ktoré budú nasledovať a typ, teda či celočíselné ***(Int)*** alebo reálne ***(Float)***. Sú aj iné typy: ***bit***, ***unsigned_char***, ***char***, ***unsigned_short***, ***short***, ***unsigned_int***, ***unsigned_long***, ***long***, ***double***. My však budeme používať len dva uvedené vyššie. V tele budú už len súradnice vrcholov. V prípade ***2D*** to budú tri čísla, kde tretie je rovné nule.

Bunky majú hlavičku podobnú, prvé je slovo ***CELLS***, potom počet buniek a za ním počet čísiel v sekcii. Telo obsahuje riadky s údajmi o bunkách, teda štvorčekoch. Každý jeden riadok má v našom prípade päť čísiel. Prvé číslo určuje, koľko čísiel bude nasledovať. Ďalšie sú indexy vrcholov z predchádzajúcej sekcie ***POINTS***, kde prvý vrchol má index 0. Číslo, ktoré určuje, koľko čísiel bude nasledovať, je v našom prípade stále štyri. No keďže bunka je polygón (trojuholník je polygón s troma vrcholmi, obdĺžnik je polygón so štyrmi vrcholmi), môže byť pospájaná aj z viacerých čiar. ***VTK_HEXAHEDRON*** má toto číslo rovné 8, pretože to je ***3D*** štruktúra, a kocky majú 8 vrcholov. Celkovo je teda v riadku 9 údajov.

Typy buniek majú v hlavičke slovo ***CELL_TYPES*** a počet zápisov typov rovnaký ako počet buniek. Typ je určovaný jedným číslom: pre ***VTK_POLYGON*** je to *7*, pre ***VTK_QUAD*** je to ***9*** a pre ***VTK_HEXAHEDRON*** je to ***12*** (toto možno vidieť aj na obrázku 4.2). Štruktúra ***Polygonal data*** túto časť vôbec neobsahuje, čím sa objem dát zmenší. Treba pripomenúť, že pri výbere štruktúry ***Polygonal data*** treba v hlavičke súboru vedľa slova ***DATASET*** napísať ***POLYGONAL***. V každom inom našom prípade tam bude slovo ***UNSTRUCTURED_GRID***.

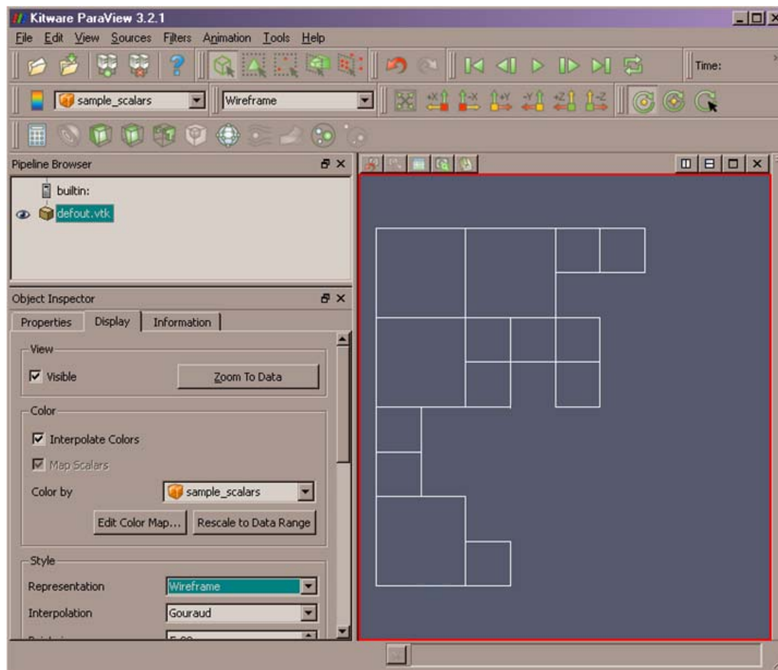
Zápis atribútov bunky (v našom prípade vlastne farby) je zložitejší. Hlavička má až tri riadky. Prvé slovo je ***CELL_DATA*** a po ňom ide počet zápisov farieb, teda buniek (lebo každá bunka má svoju farbu). Druhý riadok má tento tvar: ***SCALARS názov_atributov float 1*** (bunka môže mať aj viac atribútov, no keďže v našom prípade zobrazujeme farbu, tá je vždy len jedna) a tretí riadok je ***LOOKUP_TABLE my_table***. Po hlavičke nasleduje telo, a tým je ***n*** indexov farieb, kde ***n*** je počet buniek. Indexy farieb sú z tabuľky farieb, ktorá nasleduje ďalej. Slovo index v tomto prípade nie je úplne presné, je to skôr intenzita. Predstavme si 10 farieb, najlepšie stupne šedej, od tmavej po bielu: index 0.0 je čierna, index 1.0 je biela a index 0.5 je stredne šedá. V našom prípade však nebudeme mať takúto peknú škálu farieb, ale budeme mať konkrétne hodnoty, ktoré sa vyskytujú v obrázku, napríklad: biela, čierna, červená, modrá, zelená. To je 5 farieb. Biela bude mať intenzitu 0.0, čierna 0.25, červená 0.5 modrá 0.75 a zelená 1.0. ParaView je stále vo vývoji a je možné, že skutočnosť už predbehla tu uvádzané.

V prípade, že vynecháme atribúty (treba vynechať posledné dve časti), obrázok v ParaView sa bude defaultne javiť ako jeden šedý štvorec. Po prepnutí reprezentácie na ***wireframe*** v ***display*** sa zjaví jeho mriežka (obr. 4.3). Táto mriežka je vlastne zjednotenie hraníc všetkých buniek/polygónov. Takto sa nezobrazuje vnútro polygónu – atribúty. Presný výstup v textovom formáte je uvedený na konci ďalšej kapitoly.



Obr. 4.3

V programe je možné nastaviť, aby čiernu farbu „nebral" ako farbu, ale aby ju ignoroval. V takom prípade by sme v ParaView videli tento obraz:



Obr. 4.4

4.4. Tvorba základného výstupu v programe

Po predchádzajúcom oboznámení sa s výstupom a po ukážkach zostáva uviesť postup zápisu v našom programe. Údaje budeme postupne vpisovať do piatich textových okien, ktoré nakoniec pospájame do jedného a navyše pridáme hlavnú hlavičku súboru. Väčšina dát sa vytvára vo funkcii ***Vystup_udajov(o,f1,f2,f3,f4)***. Hlavičky sa vytvárajú až nakoniec, pretože až vtedy vieme, koľko je buniek. V našom prípade ich je ***n=28*** a vrcholov ***v=112***.

Vystup_udajov(o,f1,f2,f3,f4) je zložitá funkcia a v samotnom programe je zapísaná trochu inak. Pre jednoduchosť budeme mať oblasť štvorca jednoducho označovanú ako ***o*** a súradnice vrcholov ako ***x1***, ***y1***, ***x2***, ***y2***, ***x3***, ***y3***, ***x4***, ***y4*** (pre každú farbu je iná oblasť). Údaje o bunke sa zapisujú, len ak má bunka rôznu farbu od ostatných troch buniek, alebo ak jej farba je odlišná od hodnoty ***rozne_farby***. Farby sú reprezentované číslom od 0 do 2147483648. Hodnota ***Rozne_farby*** sa dá reprezentovať číslom -1. Ak teda majú tieto štyri bunky navzájom inú farbu, zapíšeme údaje o každej z nich, ale to len v prípade, že farba danej bunky je rôzna od ***rozne_farby***. Potom farbu všetkých buniek nastavíme ako ***rozne_farby***, takže pri postupe vyššie v rekurzii (v strome) sa už nestane, že by ďalšie štyri mali rovnakú farbu.

To bola logika, „kedy“ sa zapisujú údaje o jednej bunke. Teraz si ukážeme ako sa tvorí výstup. Zapisujú sa dve hlavné veci: do prvého textového pola sú to súradnice vrcholov

bunky a do druhého textového poľa sú to indexy vrcholov, ktoré budú pospájané do zvolenej štruktúry a ich počet (tento počet sa bude uvádzať na začiatku riadku – pred indexmi). Zápis je už naozaj jednoduchý. Poznáme všetkých 8 súradníc (4 vrcholy, ***x*** a ***y***), takže postupne súradnice vrcholov pozapisujeme. Pri tom nemožno zabúdať, že aj v prípade ***2D*** je treba 3 súradnice pre každý vrchol, kde tretiu súradnicu zvyčajne nastavujeme na nulu. Po zápise všetkých 4 vrcholov do prvého textového poľa zapíšeme ešte do druhého textového poľa do nového riadku číslovku 4 (pretože naša štruktúra štvorček pozostáva zo štyroch vrcholov) a 4 indexy. Indexy vrcholov sa zisťujú jednoducho. Prvý vrchol ma index ***pocet_vrcholov***, druhý má ***pocet_vrcholov+1***, tretí má ***pocet_vrcholov+2*** a posledný má ***pocet_vrcholov+3***. Po tomto zápise ešte zvýšime premennú ***pocet_vrcholov (v)*** o 4, a ***pocet_buniek (n)*** o 1. Dôležité je, že po ukončení celej rekurzie treba ešte vytvoriť hlavičku pre prvé a druhé textové pole.

Vytvorenie ***CELL_TYPES*** sa robí taktiež na konci rekurzie. Do tretieho textového okna napíšeme slovo ***CELL_TYPES***, počet buniek ***n*** a podľa toho aký typ buniek sme vybrali (7, 9, 12), do ďalších ***n*** riadkov napíšeme číslo typu. V prípade, že štruktúra ***DATASET*** je ***POLYGONAL*** (pozor, nemýliť si to s ***unstructured data*** - ***VTK_POLYGON***), sa táto časť úplne vynecháva.

Táto kapitola hovorí o tvorbe výstupu bez farieb. Vzhľadom na to, že výstup je príliš dlhý, rozhodli sme sa ho dať do stĺpcov. Pre ucelenosť sme sem pridali aj dva stĺpce s atribútmi/farbami, no farbám sa budeme venovať neskôr. V závere náš výstup pre obrázok ***4.3b*** bude vyzerať takto (ďalšia strana).

vtk DataFile Version 2.0 Vstup do ParaView, Bakalarka, Tomas Zachar ASCII DATASET UNSTRUCTURED_GRID

POINTS 112 float		CELLS 28 140	CELL_TYPES 28	CELL_DATA 28	LOOKUP_TABLE
		4 0 1 2 3	9	SCALARS	my_table 2
2 6 0	6 5 0	4 4 5 6 7	9	nase_atributy	1.0 1.0 1.0 1.0
3 6 0	6 4 0	4 8 9 10 11	9	float 1	0.0 0.0 0.0 1.0
3 5 0	5 4 0	4 12 13 14 15	9	LOOKUP_TABLE	
2 5 0	6 8 0	4 16 17 18 19	9	my_table	
3 6 0	8 8 0	4 20 21 22 23	9	0.0	
4 6 0	8 6 0	4 24 25 26 27	9	0.0	
4 5 0	6 6 0	4 28 29 30 31	9	0.0	
3 5 0	6 6 0	4 32 33 34 35	9	1.0	
2 5 0	8 6 0	4 36 37 38 39	9	0.0	
3 5 0	8 4 0	4 40 41 42 43	9	0.0	
3 4 0	6 4 0	4 44 45 46 47	9	0.0	
2 4 0	0 4 0	4 48 49 50 51	9	0.0	
3 5 0	1 4 0	4 52 53 54 55	9	0.0	
4 5 0	1 3 0	4 56 57 58 59	9	1.0	
4 4 0	0 3 0	4 60 61 62 63	9	1.0	
3 4 0	1 4 0	4 64 65 66 67	9	0.0	
0 8 0	2 4 0	4 68 69 70 71	9	1.0	
2 8 0	2 3 0	4 72 73 74 75	9	0.0	
2 6 0	1 3 0	4 76 77 78 79	9	1.0	
0 6 0	0 3 0	4 80 81 82 83	9	1.0	
2 8 0	1 3 0	4 84 85 86 87	9	1.0	
4 8 0	1 2 0	4 88 89 90 91	9	0.0	
4 6 0	0 2 0	4 92 93 94 95	9	1.0	
2 6 0	1 3 0	4 96 97 98 99	9	0.0	
0 6 0	2 3 0	4 100 101 102 103	9	1.0	
2 6 0	2 2 0	4 104 105 106 107	9	1.0	
2 4 0	1 2 0	4 108 109 110 111	9	1.0	
0 4 0	2 2 0			0.0	
4 8 0	3 2 0			1.0	
5 8 0	3 1 0			1.0	
5 7 0	2 1 0			0.0	
4 7 0	3 2 0			1.0	
5 8 0	4 2 0				
6 8 0	4 1 0				
6 7 0	3 1 0				
5 7 0	2 1 0				
4 7 0	3 1 0				
5 7 0	3 0 0				
5 6 0	2 0 0				
4 6 0	3 1 0				
5 7 0	4 1 0				
6 7 0	4 0 0				
6 6 0	3 0 0				
5 6 0	2 4 0				
4 6 0	4 4 0				
5 6 0	4 2 0				
5 5 0	2 2 0				
4 5 0	0 2 0				
5 6 0	2 2 0				
6 6 0	2 0 0				
6 5 0	0 0 0				
5 5 0	4 4 0				
4 5 0	8 4 0				
5 5 0	8 0 0				
5 4 0	4 0 0				
4 4 0					
5 5 0					

4.5. Optimalizácia vrcholov

Bunky – polygóny majú spoločné hranice, čo znamená, že niektoré vrcholy sú zdieľané viacerými štvorčekmi. Vrcholy v rohoch sú vždy len v jednom štvorčeku, kým vrcholy na okrajoch obrázku sú v dvoch bunkách a vrcholy vo vnútri obrázku sú v štyroch bunkách. Podstata je taká, že každý vrchol zapíšeme len raz, zapamätáme si ho v tabuľke (len počas tvorby výstupu) a následne pri tvorbe výstupu štvorčekov sa odkazujeme na vrcholy, ktoré už raz boli použité. Toto znamená, že vrcholy na okrajoch sa nebudú vyskytovať dvakrát, vrcholy vo vnútri štyrikrát, ale len jedenkrát. Pokiaľ je obrázok veľkosti 8x8 a viac, tak je vo vnútri viac vrcholov ako polovica (v prípade, ak by všetky štvorčeky mali rovnakú veľkosť – jeden pixel). Ak použijeme túto optimalizáciu pre náš prípad, tak sa nám zredukuje počet vrcholov zo 112 na 48, čo je menej ako polovica. Pre 3D dáta z obrázku 4.10 sa nám počet vrcholov zredukoval z 9640 na 2037, čo je úspora 79%. Navyše niektoré indexy neukazujú na vrcholy s trojciferným indexom, ale len s dvojciferným, čím sa tiež skráti výstup (menej bajtov). Táto optimalizácia síce nie je stratová, no spomaľuje rekurziu a taktiež si vyžaduje extra pamäť pre uchovávanie indexov vrcholov počas generovania. Táto extra pamäť je ***poleVrcholov*** a je inicializovaná na začiatku a naplnená hodnotami -1. ***Pocet_vrcholov*** je aktuálny počet vrcholov, ktorý sa pri každom zápise vrcholu zvýši o jedna. Na začiatku je počet vrcholov rovný nule.

Optimalizovaný kód by vyzeral takto (pre jeden vrchol štvorca):

```
indexV1 := poleVrcholov[x,y];
if (indexV1=-1) then begin
  indexV1 := pocet_vrcholov;
  zapis_do_vystupuPOINTS(x+' '+y+' 0');
  poleVrcholov [x,y] := indexV1;
  pocet_vrcholov = pocet_vrcholov + 1;
  end;
... (to isté pre ďalšie tri vrcholy)
zapis_do_vystupuCELLS('4 '+indexV1+' '+indexV2+
    '+indexV3+' '+indexV4)
```

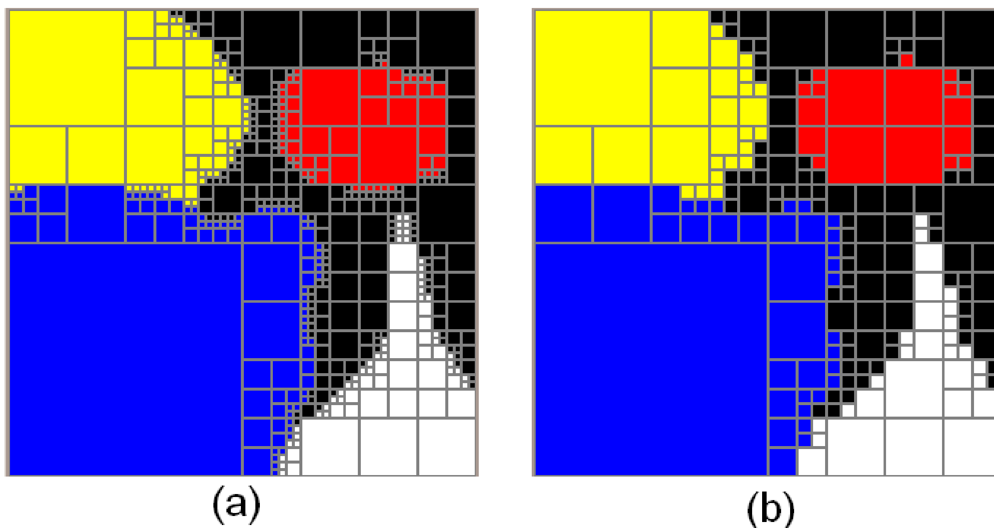
4.6. Úrovňovanie (stratový výstup)

Okrem predošlej optimalizácie, existujú ešte dva spôsoby ako zmenšiť výstup. Jeden z nich je úrovňovanie. Princíp spočíva v tom, že nepôjdeme s rekurziou na dáta na najnižšej úrovni – jeden pixel, ale len po určitú úroveň. To znamená, že ak úrovňovanie nastavíme na úroveň jedna (základný je nula), tak sa namiesto jedného pixelu v poslednom

kroku rekurzie budú analyzovať hneď štyri. Pre jednoduchosť a rýchlosť môžeme farbu týchto štyroch pixelov zameniť za farbu jedného z týchto štyroch (konkrétne pravého dolného) a určiť si, že to bude len jedna bunka so štyrmi vrcholmi. Takto vlastne ušetríme všetky údaje o zvyšných troch bunkách, päť vrcholov a tri bunky. Navyše rekurzia zbehne štyrikrát rýchlejšie. Existujú aj iné možnosti voľby kritéria. Poprvé: vrátiť priemer farieb. Tu by nám však vznikali nové farby, čo by bolo možné ošetriť zmenou farby na najviac podobnú farbu z existujúcich farieb. Druhá možnosť je nevybrať farbu z jedného rohu, ale tú, ktorá sa tam vyskytuje najčastejšie (modus). My sme sa pre tieto spôsoby nerozhodli pre spomalenie algoritmu a tiež preto, že samotná idea, že nahrádzame farby inými znamená, že sme si vedomí straty. Preto si myslíme, že takéto jednoduché riešenie je postačujúce, pozri obrázok 4.5. Obrázok 4.5a je bez úrovňovania, 4.5b je s úrovňovaním stupňa jedna (žiadne je nula).

Farba_pixelu je funkcia, ktorá štandardne vráti farbu jedného pixelu, ale pri úrovňovaní vráti farbu podľa nami daného kritéria, čiže farbu pravej dolnej podoblasti. Kód sa zmení len nepatrne.

```
Jadro(o: oblast):farba;
Begin
  If (stupen = minimalna_uroven) then
    farba := farba_pixelu(o);
  ...
```

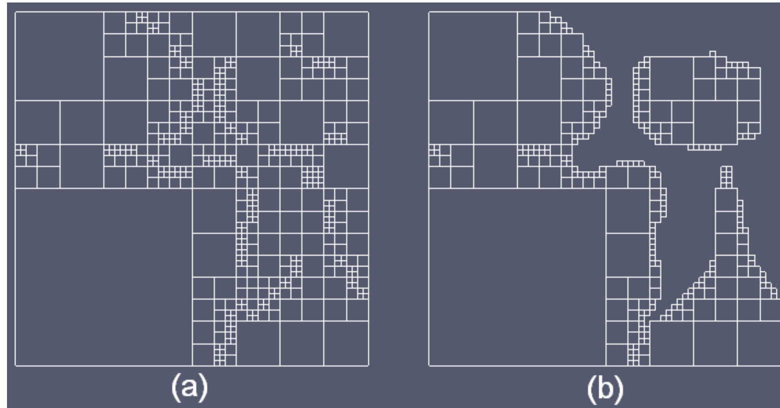


Obr. 4.5

4.7. Atribúty a farebná paleta

Výstup pre ParaView, ktorý nemá žiadne informácie o atribútoch, je vlastne len mriežka, ktorá je v ParaView štandardne reprezentovaná bielošedou farbou. V našom programe je možnosť nevytvárať farebnú paletu, teda nevytvárať údaje o atribútoch buniek.

Tento efekt dosiahneme jednoduchým pridaním podmienok pred tvorbu výstupu atribútov. Pre zaujímavejší efekt je tu možnosť vypnúť kreslenie čiernej farby. Výstup bez farieb je rýchlejší a taktiež zaberá menej priestoru na disku. Pre dáta použite v predchádzajúcej kapitole je výstup na obrázku 4.6a. Obrázok 4.6b je ten istý výstup bez zobrazenia čiernej farby.



Obr. 4.6

My farby požadujeme, preto je táto možnosť štandardne zapnutá. Výstup týkajúci sa farieb, aj odtieňov šedej, pozostáva z dvoch sekcií. Druhou sekciou je zoznam farieb (farebná paleta), ktoré sú použité v dátach a prvou je priradenie týchto farieb pre každú bunku. Sekcia začína slovami ***LOOKUP_TABLE meno_tabulky*** a ďalej nasledujú riadky s farbami. Je to vlastne zoznam farieb, kde každý riadok obsahuje štyri reálne čísla od ***0*** do ***1***. Sú to ***RGBA*** čísla - ***Red***, ***Green***, ***Blue*** a ***Alpha***, čo v preklade znamená červená, zelená, modrá a priesvitnosť. Priesvitnosť necháme vždy rovnú jednej. Uvediem aspoň pár príkladov: ***0 0 1 1*** je modrá, ***1 1 0 1*** je žltá, ***0 0.5 0 1*** je tmavozelená. Treba podotknúť, že desatinný oddeľovač je desatinná bodka. Výstup sa vytvára tak, že v podstate pri analýze buniek, teda počas rekurzie a zápisu dát, zapisujeme aj jednotlivé farby do vyhradenej tabuľky v pamäti (aby sme žiadnu farbu nezapísali dvakrát). Pokiaľ sa farba v tabuľke ešte nevyskytuje, zapíšeme ju aj do výstupu, teda do druhej sekcie (zoznamu, farebnej palety). Každá farba v delphi je reprezentovaná jedným číslom a na získanie jednotlivých farebných zložiek nám pomôžu zabudované funkcie, o ktorých sa píše podrobnejšie v kapitole Extra doplnky programu. Okrem zoznamu vytvárame tiež extra tabuľku (***PoleF***) farieb pre bunky, pretože každá bunka má svoju farbu. Sem zapisujeme indexy farieb zo zoznamu, teda čísla od nula po ***počet farieb-1***. Napríklad: prvá bunka ma nultú farbu, druhá ma tiež nultú farbu atď. V ParaView treba miesto indexov používať intenzity od 0 po 1, no my zatiaľ nevieme, koľko farieb bude spolu. Na konci programu, keď budeme vedieť počet farieb, ktoré obrázok obsahuje, môžeme prvky pola ***PoleF*** (indexy) predeliť ***počtom farieb-1***. Takže ak sme mali

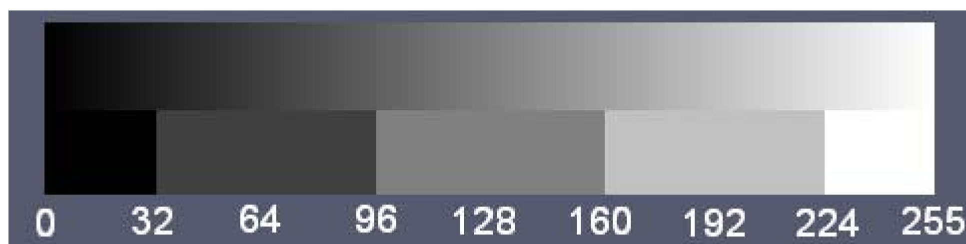
farby ***0 0 1 1 0 1 2 2 0***, tak počet je 3, ***počet-1*** je 2, a po predelení získame čísla ***0 0 0.5 0.5 0 0.5 1 1 0***. Toto nové pole zapíšeme do výstupu.

V prípade, že farby sú štyri, budeme deliť trojkou, čo bude dávať čísla ako 0.33333333333 alebo iné nepresné čísla. Rovnaký prípad môže nastať pri iných hodnotách. Považujeme za vhodné, aby množstvo typov farieb bol vždy nejaký vhodný počet, ako napríklad 2, 3, 5, 6, 11, 21, 31... Preto zoznam farieb je doplnený nulami do takéhoto počtu.

4.8. Prahovanie

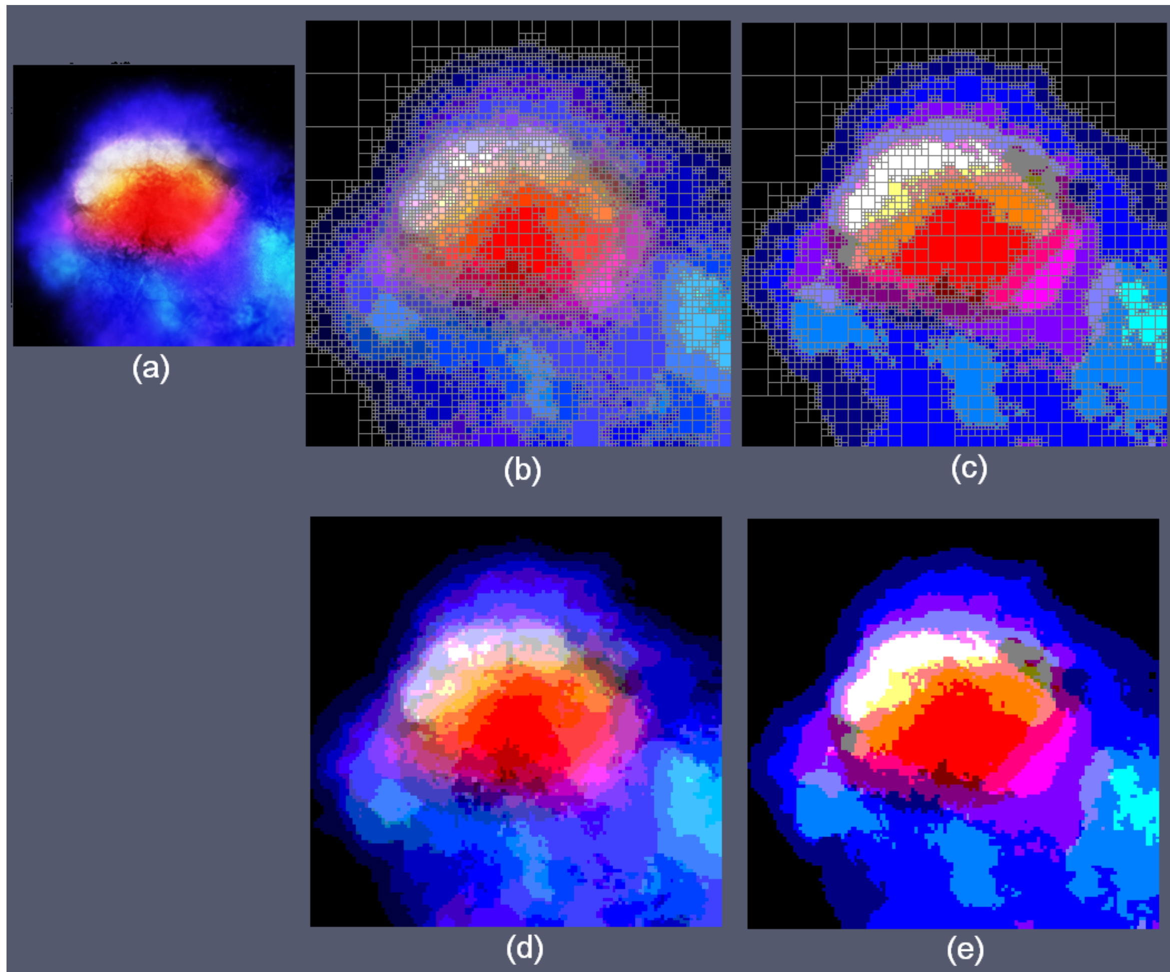
Chceli by sme zdôrazniť, že táto funkcia je stratová. Je to funkcia, ktorá nahradí podobné farby jednou farbou, a teda bunky s podobnými farbami zlúči. Jej výhodou je menšia veľkosť výstupu, teda menšie množstvo farieb, a súčasne je väčšia šanca vzniku väčších plôch/štvorčekov jednej farby. Prahovanie spočíva v tom, že farby ktoré majú podobné hodnoty zložiek sa nahradia jednou hodnotou. V našom prípade si môžeme túto hodnotu zvoliť. V programe ju nazývame ***prah***. Pre ***prah*** rovný „***64***“ (naše označenie) sa nám všetky zložky červenej v rozmedzí 0-31 nahradia hodnotou 0, červená 32-95 hodnotou 64, 96-159 hodnotou 128, 160-223 hodnotou 192 a 224-255 hodnotou 255 (viď obr. 4.7). Implementovanie do rekurzie je jednoduché. Riadok kódu ***farba := farba_pixelu(o);*** nahradíme riadkom ***farba := prahovanie(o);***.

```
function prahovanie(farba:integer):integer;
var r,g,b:integer;
begin
  result := farba;
  if (prah>1) then begin
    r := (round(getRvalue(farba) / prah)) * prah;
    if (r>255) then r:=255;
    g := (round(getGvalue(farba) / prah)) * prah;
    if (g>255) then g:=255;
    b := (round(getBvalue(farba) / prah)) * prah;
    if (b>255) then b:=255;
    result := RGB(r,g,b);
    end;
  end;
```



Obr. 4.7

Pre názornú ukážku tu máme obrázok 4.8 s tromi prahovaniami. Bez prahovania (4.8a), ***prah = 64*** (4.8b, 4.8d) a ***prah = 128*** (4.8c, 4.8e). Obrázky 4.8d a 4.8e sú bez mriežky pre lepšie znázornenie prahovania.



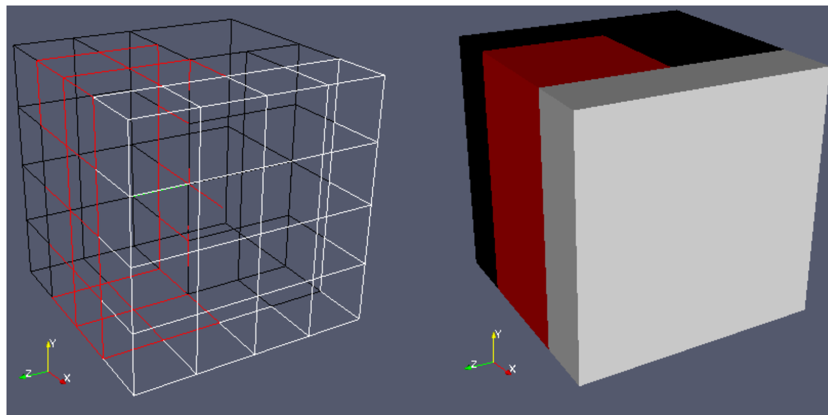
Obr. 4.8

Existuje ešte aj iné prahovanie, takzvané binárne. Je to prahovanie, ktoré ma dané dve hodnoty, a všetky farby pod prvou hodnotou sa nahradia prvou hodnotou, a všetky farby nad druhou hodnotou sa nahradia druhou hodnotou. Takéto prahovanie sa v programe nevyskytuje.

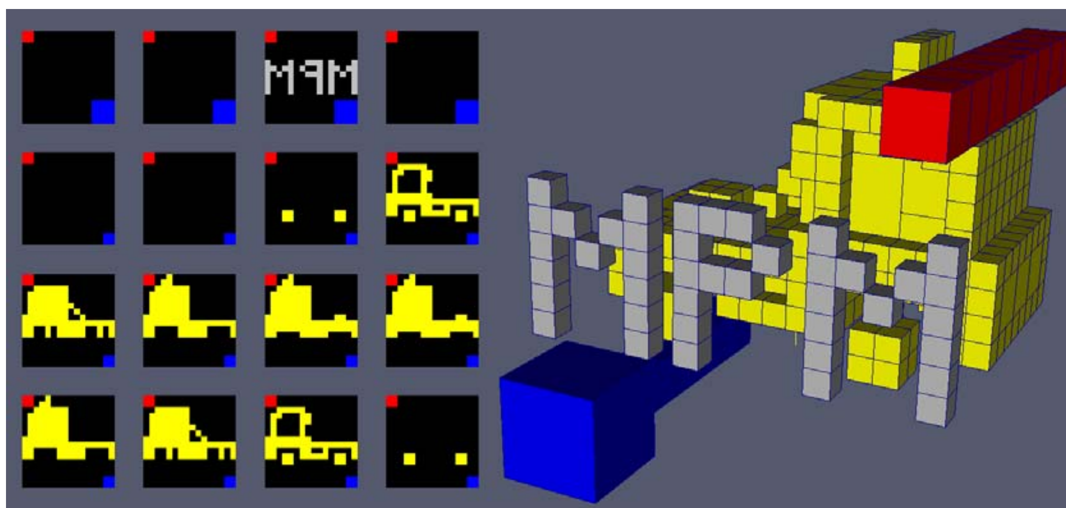
4.9. Spracovanie objemových telies (3D štruktúry)

3D telesá majú ako už skratka ***3D*** napovedá, tretiu dimenziu, teda tretí rozmer. Preto sa oblasti nebudú deliť na štyri časti ako to bolo u ***2D*** objektov (2^2), ale na osem častí (2^3). Preto sa stromy budú volať ***oktantové*** (***Octtree***). Princíp je ten istý, len rekurzívna funkcia sa bude volať osemkrát v sebe samej, bude sa porovnávať osem farieb, a pri výstupe nebude tretia súradnica rovná nule, ale bude vznikať tak ako prvé dve. Rozdiel je hlavne na vstupe, kde nemáme jeden obrázok ale sadu obrázkov (obrázok 4.10, bez čiernej farby), alebo

podobným štýlom dáta typu ***raw*** tak ako to bolo v ***2D***. Na obrázku 4.9 nájdeme príklad trojrozmernej mriežky.



Obr. 4.9



Obr. 4.10

4.10. Doplnky programu

V programe sa často opakuje funkcia s názvom ***Application.ProcessMessages;*** Je to funkcia, ktorá je už zabudovaná v delphi a slúži na to, aby si počítač počas behu programu vyhradil krátky čas aj na iné aplikácie, ktoré bežia na počítači.

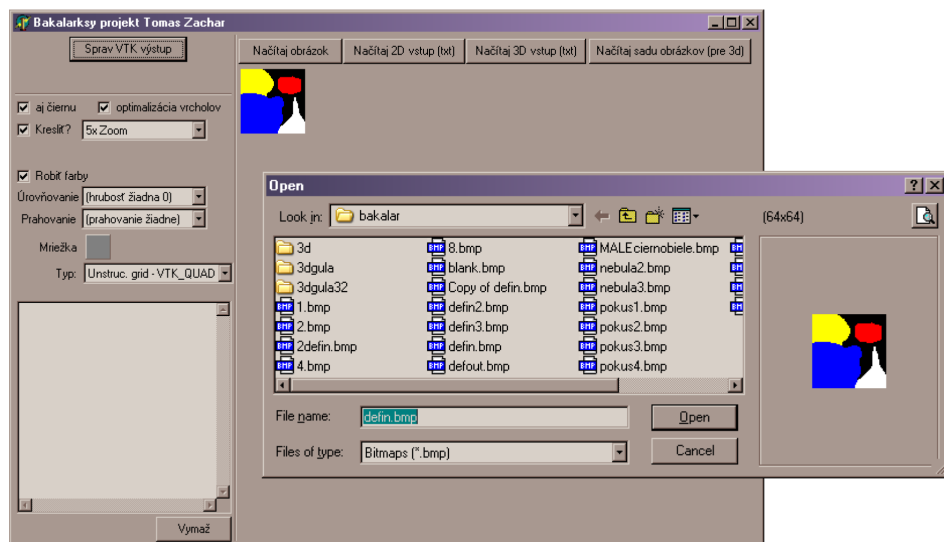
GetRvalue, ***GetGvalue***, ***GetBvalue*** sú funkcie ktoré dokážu z čísla získať len danú farebnú zložku. Teda ***GetRvalue*** nám vráti hodnotu ***0*** až ***255*** z daného čísla (respektíve farby), kde 0 znamená, že červená sa v čísle vôbec nevyskytuje, teda daná farba nemá červenú zložku. V našom programe však potrebujeme hodnoty od ***0*** po ***1***, teda jednoducho túto hodnotu predelíme číslom ***255***.

Power je funkcia, ktorá vráti ***x-tu*** mocninu čísla ***y***. Napríklad ***power(3,4)*** je tri na štvrtú a to je 81.

5. Užívateľské pracovanie s programom

5.1. Rozhranie

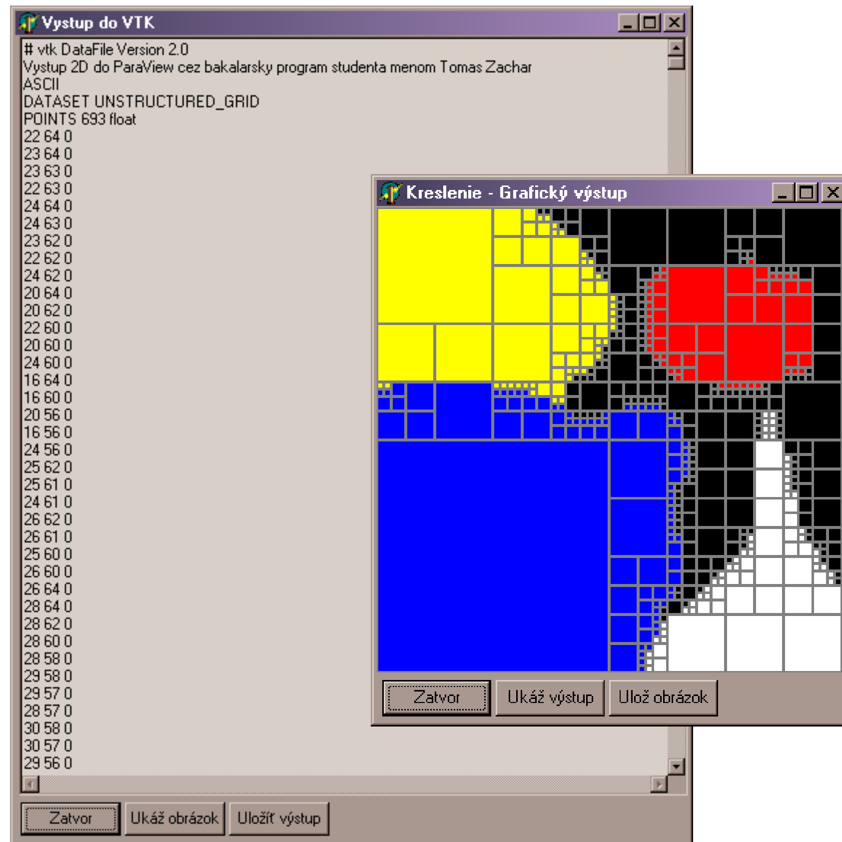
Po spustení programu sa automatický načíta vstup ***defin.bmp***, a zobrazí sa pod tlačidlom ***načítaj vstup***. V pravo hore sa nachádzajú štyri možnosti načítania. Prvou možnosťou je obrázok vo formáte ***BMP***, druhou sú dáta typu ***raw*** v ***ascii*** kóde. Treťou možnosťou je ***3D*** vstup, ktorý je podobný ako druhý. Posledný je tiež ***3D*** vstup. Pri nami ľubovoľne vybranom obrázku sa program následne pokúsi otvoriť sadu obrázkov. Takže, napríklad: ak otvoríme ***pokus1.bmp***, program zistí, že pokus1.bmp má rozmer ***4x4***, a pokúsi sa načítať dáta ***pokus1.bmp***, ***pokus2.bmp***, ***pokus3.bmp*** a ***pokus4.bmp***.



Obr. 4.11

Na pravej strane je hlavné tlačidlo ***Sprav VTK Výstup***, menu pre rôzne možnosti na výstupe a informatívne okno o výstupe. Možnostiam sa budeme venovať v ďalšej časti. Teraz si ukážeme, čo sa stane po vybratí ***2D*** vstupu a kliknutí na ***Sprav VTK Výstup***. Na obrázku 4.12 sa postupne otvoria dve okná (závisí od rýchlosti počítača, no v konečnom dôsledku by mali byť dve). Prvé okno je grafické (v ***3D*** sa nezobrazuje). Zobrazuje sa tu postupné vykresľovanie mriežky. Aj tu sa nachádzajú tri tlačidlá. ***Zatvor*** slúži na zatvorenie okna, ***ukáž výstup*** prepne do tretieho okna a ***ulož obrázok*** dá možnosť uložiť si výstup ako bmp obrázok. Tretie okno je samotný výstup, teda vzniknutý súbor. Jeho štandardné meno je ***defout.vtk***. Výstup je však možné uložiť aj pod iným názvom. V prípade nutnosti je možné vstup editovať (napríklad druhý riadok slúži ako popis, ktorý slúži len ako informácia). Taktiež je tu možnosť okno zavrieť. Po vygenerovaní takéhoto výstupu sa v ľavom dolnom rohu v hlavnom okne (obr. 4.11) zobrazia informácie o výstupe, ako napríklad počet

vrcholov, veľkosť súboru, typ štruktúry a iné. Celý program možno ukončiť aj stlačením klávesy ***ESC***.



Obr. 4.12

5.2. Možnosti

Program obsahuje 9 možností a všetky sa nachádzajú v ľavom paneli. ***Aj čiernu*** znamená, že program bude pracovať aj s čiernou farbou. ***Optimalizácia vrcholov*** znamená, že každý vrchol sa bude zapisovať len raz. Možnosť ***kresliť*** ukáže grafický výstup priamo počas generovania. Možnosťou ***zoom*** sa určí zväčšenie výstupu. Pri vynechaní možnosti kreslenie sa spracuje výstup oveľa rýchlejšie. Odškrtnutím voľby ***Robiť farby*** sa zamedzí programu tvoriť atribúty/farby vo výstupe. ***Úrovňovanie*** aj ***prahovanie*** je vysvetlené v predchádzajúcich kapitolách. ***Mriežka*** je len pre grafický výstup, pokiaľ je obrázok šedý, je lepšie použiť červenú mriežku. Avšak pri vypnutí možnosti kreslenie nemá táto farba mriežky význam. Možnosť ***Typ*** slúži k voľbe typu štruktúry.

6. Záver

6.1. Záver

Program je schopný zo vstupných dát vytvoriť výstupné dáta v štyroch štruktúrach, v prípade 2D v troch. Nepozorovali sme rozdiely v ich využití, teda všetky štyri majú rovnaké možnosti operácii na nich, akými sú ***clip*** (orezávanie), ***slice*** (robenie rezov) a ***treshold*** (zobrazenie len niektorých farieb). Kontúry nie sú možne zobraziť ani na jednej štruktúre. ***ParaView verzia 2.4.0*** na rozdiel od ***verzie 3.2.1*** nie je schopná na týchto dátach spraviť ***treshold***. Je možné, že s niektorými typmi pracuje počítač rýchlejšie, no toto sme tiež nepozorovali. Podarilo sa však ukázať, že typ ***polygonal data*** zaberá jednoznačne menej ako ostatné štruktúry, a to práve preto, že vynecháva typy jednotlivých buniek. Taktiež optimalizácia vrcholov ma kladný dopad na veľkosť výstupného súboru.

6.2. Poďakovanie

Ďakujem vedúcej práce RNDr. Zuzane Krivej, PhD. za usmerňovanie, podporu sebadôvery a aktivity, poskytnutie vstupných dát, materiálov a pomocnej literatúry.

Ďakujem rodičom za psychickú podporu, mame za kontrolu práce z hľadiska pravopisu a štylistiky.

Ďakujem spolužiakom za pomoc pri rozoberaní výsledku bakalárskej práce a kamarátom za pozitívne ohlasy pri sledovaní generovania výstupu.

7. Súhrn

Bakalárska práca sa zaoberá základmi kvadrantových a oktantových stromov, ich jednoduchým vysvetlením. Hlavným cieľom programu je vytvoriť zo vstupných dát výstupne pre program ParaView. Taktiež tu nájdeme krátke oboznámenie sa s programom ParaView a dôvod na použitie stromov. Ďalej nasledujú štruktúry štyroch vstupov ParaView, základy rekurzie a s ňou aj programátorské riešenie. V programe ďalej rozoberáme ako jednotlivé časti fungujú, teda ako sa vytvárajú jednotlivé časti výstupu. Postupne sú tu vysvetlené rôzne možnosti programu akými sú optimalizácia, úrovňovanie, farebná paleta, prahovanie a základy pre 3D. Bakalárku ukončujú 3D výstupy a prehľad užívateľského rozhrania bakalárskeho programu.

8. Summary

Bachelor project deals with basics of Quadrant Tree (Quadtree), Octant Tree (Octtree) and with their easy definition. The main aim of the program is to create an output from the input for application ParaView. Also you can find here short acquaintance with ParaView and the reason for using trees. Next section is about structures of four inputs in ParaView, basics of recursion and the programming solution. Next thing in program that we are analyzing is how the parts work, meaning how the each part of output is created. Step by step, different options of the program are explained, such as optimalization, leveling, color palette, threshold and the basics of 3D. Bachelor project is ended with the 3D outputs and overview at user interface of our program.

9. Použitá Literatúra

[1]: http://www.paraview.org/New/index.html#

[2]: http://www.vtk.org/

[3]: Marshall Bern, David Eppslein – Mesh generation and optimal triangulation

[4]: The ParaView Guide, Amy Henderson, 2004 Kitware, ISBN 1-930934-14-19

[5]: http://www.kaabel.name/fic/part1.htm

[6]: http://www.viliam.bur.sk/sk/2007-08-24/Rekurzia

[7]: http://www.vtk.org/pdf/file-formats.pdf