Čestné prehlásenie

Prehlasujem, že som bakalársku prácu vypracoval sám, len za pomoci citovanej literatúry a s odbornou pomocou vedúceho práce.

Bratislava 16. mája 2013 .....

Vlastnoručný podpis

Poďakovanie

Chcem sa poďakovať vedúcemu práce Ing. Róbertovi Špirovi za pomoc, venovaný čas počas konzultácií, odborné vedenie a cenné rady pri vypracovaní bakalárskej práce. Osobitné poďakovanie patrí mojej rodine za podporu a spolužiakom za spoločné štúdium.

Abstrakt

V tejto práci sme sa zamerali na efektívnu implementáciu vizualizácie rôznych matematických dát, ktoré sú reprezentované na obdĺžnikovej mriežke. Program na vykresľovanie používa grafickú kartu počítača a je teda veľmi rýchly, je možné plynule zobrazovať aj dátové súbory skladajúce sa z viac ako 8 miliónov bodov. Program umožňuje ľubovoľné natáčanie a približovanie vizualizovaných dát, úpravu zobrazenej farebnej škály a takisto zobrazenie iba určitej podmnožiny dát. Okrem toho je možné zobrazovať aj animáciu z viacerých dátových súborov a zobraziť rezíduá medzi dvoma súbormi. Pri načítavaní a práci s rozsiahlymi vstupnými súbormi sa nám podarilo paralelizovať a zefektívniť väčšinu výpočtových úkonov, vďaka čomu je práca s programom rýchla aj pri súboroch s veľkosťou viac ako 100 megabajtov.

Abstract

In this work we focus on the efficient implementation of various mathematical data visualization, which are represented on the rectangular grid. The program is rendering the data on computer graphics processing unit and is therefore very fast, smoothly displaying datasets with more than 8 million points. The program allows free rotation and zooming of displayed data, modification of the colour scale and displaying only selected data subset. Besides that the program can display animation of multiple data files and residuals between two files. During reading and calculations with large input files we were able to parallelize and optimize most of the operations and program usage is fast even with files with the size of more than 100 megabytes.

Obsah

1 Úvod

Už od mladého veku sa nám všetko snažia vysvetliť pomocou vizuálnych pomôcok, preto je pochopiteľné, že väčšina ľudí je vizuálne orientovaná. Tak isto, ako aj v ostatných oblasti života, aj vo vede nám pomôžu grafy, obrázky a modely pochopiť získané výsledky. Práve preto sa snažíme v tejto práci vytvoriť prostredie, ktoré umožní dlhý zoznam čísel premeniť na trojrozmerný model, podľa ktorého aj laik pochopí, aký význam majú dáta získané náročným výpočtom.

V dnešnej dobe je samozrejme najlepšie vytvorenie modelu na počítači. Existuje mnoho vizualizačných softvérov, avšak ich najväčšia výhoda, že sa dajú použiť s množstvom formátov súborov a typov dát je zároveň aj ich najväčšia nevýhoda, pretože s rozširovaním funkcionality prichádza aj znižovanie výkonu takýchto softvérov. Riešením je vytvoriť si vlastný vizualizačný softvér, ktorý môžeme uspôsobiť a optimalizovať iba na množinu dát, s ktorou pracujeme a dosiahnuť vizualizáciu veľmi detailných a rozsiahlych modelov, pri ktorých už bežné softvéry zlyhávajú.

Takto vytvorený softvér nemusí umožňovať iba jednoduché vykresľovanie dát. Môžeme pridať rôzne funkcie, ktoré nám poskytujú možnosti na manipuláciu s modelom aj samotnými dátami.

2 Dáta

Dáta, pre ktoré primárne vznikol tento program na tvorbu trojrozmerných modelov, predstavujú zoznam geodetických súradníc a hodnoty poruchového potenciálu. Sú uložené do súborov s príponami .dat. V týchto súboroch sa v každom riadku nachádzajú tri čísla, ktoré predstavujú zemepisnú šírku (latitude), zemepisnú dĺžku (longitude) a poruchový potenciál ($m^2.s^{-2}$).

```
-79.9583 0.0417 -43.4565
-79.9583 0.1250 -43.0557
-79.9583 0.2083 -42.6513
-79.9583 0.2917 -42.2435
-79.9583 0.3750 -41.8331
-79.9583 0.4583 -41.4202
-79.9583 0.5417 -41.0049
-79.9583 0.6250 -40.5875
-79.9583 0.7083 -40.1679
-79.9583 0.7917 -39.7463
-79.9583 0.8750 -39.3228
-79.9583 0.9583 -38.8977
-79.9583 1.0417 -38.4713
-79.9583 1.1250 -38.0445
-79.9583 1.2083 -37.6175
-79.9583 1.2917 -37.1910
-79.9583 1.3750 -36.7654
-79.9583 1.4583 -36.3407
-79.9583 1.5417 -35.9169
```

Obr. 2.1 Vzor dát vo vstupnom súbore

Naše dáta sú získané riešením problému geodetickej okrajovej úlohy so zmiešanými okrajovými podmienkami (Geodetic Boundry Value Problem with Mixed Boundry Condition) metódou konečných objemov (Finite Volume Method).

Formulácia problému je nasledovná [1]:

Majme výpočtový objem Ω ohraničený aproximovaným povrchom Zeme, hornej umelej hranice nachádzajúci sa v určitej vzdialenosti od Zeme a štyrmi umelými bočnými hranicami. Dolnú hranicu, ktorá predstavuje reálnu časť povrchu Zeme, označme Γ, potom geodetická okrajová úloha bude definovaná nasledovne:

$$-\Delta \mathrm{T}(x) = 0\,,\ x \in \Omega\,, \tag{2.1}$$

$$\frac{\partial T}{\partial n_\Gamma} = -\delta \mathrm{g}^*(x) = -\delta \mathrm{g}.\cos\mu\,,\ \mu = \angle(\bar{s}, \bar{n}_\Gamma) \ \ \text{na} \ \ \Gamma\,, \tag{2.2}$$

$$T(x) = T_{\mathrm{SAT}}(x),\ \text{na}\ \partial\Omega - \Gamma\,, \tag{2.3}$$

kde $T(x)$ je poruchový potenciál definovaný ako rozdiel medzi reálnym W(x) a normálovým U(x) gravitačným potenciálom v bode x, T_{SAT} predstavuje poruchový potenciál, ktorý bol generovaný zo satelitného geopotenciálneho modelu, $\delta\mathrm{g}(\mathrm{x})$ je gravitačná porucha, $\mu = \angle(\bar{s}, \bar{n}_{\Gamma})$ je uhol medzi normálou k ekvipotenciálnemu povrchu U(x) = konštanta a normálou ku Γ.

Tento problém budeme riešiť pomocou metódy konečných objemov. Pre formuláciu riešenia rovnice (2.1) musíme celú výpočtovú oblasť rozdeliť na konečný počet malých objemov, ktoré označíme $p_{i,j,k}$. Potom integračné rovnice na konečných objemoch sú založené na slabej formulácii diferenciálnej rovnice (2.1).

$$-\int_p \Delta\mathrm{T}\, dx\, dy\, dz = -\int_{\partial p} \nabla\mathrm{T}.n\, d\sigma = -\int_{\partial p} \frac{\partial T}{\partial n} d\sigma = 0 \qquad (2.4)$$

Deriváciu pozdĺž hranice objemu p môžeme aproximovať nasledovne

$$\frac{\partial T}{\partial n_{\mathrm{pq}}} \approx \frac{T_q - T_p}{d_{\mathrm{pq}}}$$

a

$$\mathrm{d}\sigma = m(e_{\mathrm{pq}})$$

aby sme dostali

$$-\sum_{\mathrm{q}\epsilon\mathrm{N}_p} \frac{T_q - T_p}{d_{\mathrm{pq}}} m(e_{\mathrm{pq}}) = \sum_{\mathrm{q}\epsilon\mathrm{N}_p} \frac{m(e_{\mathrm{pq}})}{d_{\mathrm{pq}}} (T_q - T_p) = 0 \qquad (2.5)$$

kde d_{pq} je vzdialenosť od stredu objemu p ku stredu susedného objemu q.

3 Implementácia

Program 3D KmaDG Vizualizátor je naprogramovaný v jazyku C# pomocou vývojového prostredia Visual Studio 2012 s využitím knižnice SlimDX [2], ktorá umožňuje .NET aplikáciám používať Microsoft DirectX APIs (application programming interfaces) priamo v spravovanom kóde. Avšak správa pamäte pre DirectX objekty vzhľadom na zvýšenie výkonu tejto knižnice nie je automatická, a preto je potrebné si dávať pozor na ich správne uvoľňovanie.

Ako základ sme použili existujúci program, ktorého jedinou funkcionalitou bolo vykreslenie formuláru s panelom, spustenie prázdneho renderovacieho cyklu v rámci formulárovej pumpy na zachytávanie udalostí a správ a nastavenie matice perspektívy zobrazenia. Tento základ bol použitý z programu pôvodne vytvoreného v [3].

3.1 Jazyk C#

C# bol vyvinutý firmou Microsoft. Jeho vývojový tím vedie Anders Hejlsberg a jeho najnovšia verzia je C# 5.0. C# je jednoduchý objektovo orientovaný programovací (OOP) jazyk, ktorý bol odvodený od jazykov C a C++. C# bol navrhnutý tak, že všetky funkcie a vlastnosti jazykov C a C++, ktoré splnili svoje účely a programátorská komunita bola s nimi spokojná, sa zachovali aj v tomto jazyku. Programátori, ktorí už majú skúsenosti s programovaním v jazyku C a C++, si rýchlo zvyknú aj na C# . V C# sa všetko správa ako objekt. Zjednotením typového systému je zabezpečené to, aby sa mohlo zaobchádzať so všetkými dátami ako s objektmi. Základmi sú vlastnosti, metódy a udalosti.

Jazyk C# bol vybraný kvôli jeho jednoduchej použiteľnosti. C# obsahuje automatickú správu pamäte a namiesto smerníkov používa referencie na objekty. Správca pamäte potom na základe počtu aktívnych referencií vie sám zistiť ktoré objekty sa ešte používajú a ktoré je už bezpečné z pamäte odstrániť. Programovanie sa takto stane omnoho intuitívnejšie, pretože programátorovi odpadá starosť so správnou alokáciou a dealokáciou pamäte, čo býva často najkomplikovanejšia úloha, ktorá pritom nijako nesúvisí s priamou funkcionalitou daného programu. S využitím knižnice SlimDX na prístup

k volaniam DirectX nám jazyk C# poskytuje všetky potrebné predpoklady na vytvorenie nášho softvéru.

3.1.1 Objektovo orientované programovanie v C#

C# podporuje všetky princípy OOP. Centrálnou myšlienkou OOP sú objekty. Tak ako aj v bežnom živote, v OOP sú problémy riešené pomocou objektov. Tento spôsob je omnoho intuitívnejší ako práca s dátovými štruktúrami, akú poznáme pri bežných programovacích jazykoch. Keď chceme nájsť nejaké dáta, nemusíme sa zaoberať tým, ako ich náš objekt zistí. Objekt obsahuje rovno metódy, funkcie a princípy, pomocou ktorých vie pracovať s dátami. To nám dovoľuje, aby sme premýšľali o problémoch prirodzenejším spôsobom.

Predlohou objektu je trieda. Triedu si môžeme predstaviť ako typ, ktorému sú priradené určité funkcie. Objekt je konkrétnym príkladom typu alebo triedy. Objekt bude definovaný s tým istými funkciami ako zdrojová trieda, t.j. objekt je jednou inštanciou triedy.

Základné princípy objektovo orientovaných programovacích jazykov sú:

- Zapuzdrenie
- Dedičnosť
- Polymorfizmus

3.1.2 Zapuzdrenie

Zapuzdrenie (“ukrývanie informácií“ – information hiding) je schopnosť objektu ukryť svoje vnútorné dáta a metódy pred užívateľom a poskytnúť rozhranie pre prístup k premenným, s ktorými môže užívateľ priamo manipulovať. Vykonáva sa nastavením prístupových práv. Tieto práva sa odlišujú pomocou špecifikátorov:

- public (verejné)
- private (súkromné)
- protected (chránené)
- internal (vnútorné)

Zapuzdrenie predstavuje hranicu medzi vonkajším rozhraním objektu a vnútornými procesmi. Rozhranie umožňuje užívateľom prístup k informáciám a funkciám ktoré potrebuje, ale skryje vnútorné činnosti triedy. Prvky triedy, ktoré sú označené špecifikátorom public, sú prístupné každej časti programu. K prvkom,

ktoré sú označené ako private, má prístup len samotná trieda. K protected prvkom majú prístup aj odvodené triedy.

Pri programovaní je vždy najlepšie členské premenné triedy označiť ako private. V bežných OOP jazykoch, ako je aj C++ , sa na manipulácie s premennými používajú tzv. prístupové metódy. C# ponúka účinnejší mechanizmus vo forme vlastností. Vlastnosť (property) pozostáva z deklarácie premennej a prístupových metód. Takto vyzerá vlastnosť v samotnom kóde :

```
class Trieda
{
        private string premenna;
        public string Premenna
        {
                get { return premenna; }
                set { premenna = value;}
        }
}
```

3.1.3 Dedičnosť

Dedičnosť predstavuje možnosť vytvárania odvodených tried založených na existujúcich triedach. Odvodené triedy môžeme ďalej upravovať a vytvárať z nich nové objekty. Nové triedy označujeme ako odvedené (potomkovia). Triedy, z ktorých vzniknú potomkovia sa volajú základné (bázické, rodič). Odvodené triedy dedia všetky premenné základných tried [4].

3.1.4 Polymorfizmus

V jazyku C# môžeme polymorfizmus implementovať pomocou dedičnosti alebo pomocou rozhrania. Polymorfizmus znamená, že inštancie základnej a odvodenej triedy sú schopné odpovedať rozličným spôsobom. Podstata polymorfizmu realizovaného dedičnosťou spočíva v prekrytí člena základnej triedy v tele triedy odvodenej.

Aby sme mohli v podtriede pokryť metódu základnej triedy, musíme vykonať tieto činnosti:

- V definícii metódy základnej triedy sa musí nachádzať modifikátor virtual:

  public virtual void Metóda() { }

- V definícii rovnomennej metódy odvodenej triedy musí byť uvedený modifikátor override:

  public override void Metóda() { }

Prekrývajúca metóda podtriedy s modifikátorom override musí byť s prekrývanou (virtuálnou) metódou bázovej triedy zhodná v týchto aspektoch:

- signatúra (zoznam formálnych parametrov)
- dátový typ návratovej hodnoty,
- prístupový modifikátor [5].

3.1.5 Výnimky

Pri zlyhaní normálneho priebehu kódu sa môžu vyskytnúť chybové stavy. Tieto chybové stavy sa nazývajú výnimkami. Taká výnimka môže byť napríklad pri načítavaní dát zo súboru, keď operačný systém objaví chybu na disku a súbor sa kvôli tomu neotvorí. Zvyčajne sú metódy tried vytvorené na vykonávanie jednej činnosti a nevedia vyriešiť výnimky samé. Z tohto dôvodu existuje ošetrovanie výnimiek. Základná syntax ošetrovania výnimiek sa skladá z troch príkazov : try, catch, throw. Ako príklad používania výnimiek uvedieme časť kódu nášho programu :

```
try
{
    SetUpColor(name);
    GenerateBitmap();
    CopyToBuffers();
    render = true;
}
catch (Exception e)
{
    MessageBox.Show(e.Message,"Chyba",MessageBoxButtons.OK);
    return;
}
```

V prípade , keď niektorá z uvedených metód narazí na výnimočnú situáciu, t.j. nevie vykonať jej cieľ, pomocou príkazu throw ohlási výnimku volajúcej metóde. Aby sme mohli volať výnimku, musíme mať k dispozícii objekt typu System.Exception. Ohlásenie výnimky potom môže vyzerať nasledovne :

```
throw new Exception("zlyhalo kopírovanie dát do bufferov!")
```

Aby sme mohli zachytiť výnimky, metódy musia byť uvedené v bloku try. Príkazy v tomto bloku sa spracúvajú priebežne, kým niektorý z nich neohlási chybu. Výnimku po bloku try potom zachytá príkaz catch. Po tomto príkaze je v krútených zátvorkách uvedené, ako sa má výnimka ktorá sa vyskytla vyriešiť. V našom prípade sa pomocou statickej triedy MessageBox v novom okne vypíše správa výnimky. Okrem týchto dvoch blokov existuje aj blok finally, pričom príkazy v tomto bloku sa vykonajú bez ohľadu na to či bola vyvolaná výnimka [4].

3.2 Vývojové prostredie

3D KMaDG Vizualizátor bol vytvorený vo vývojovom prostredí Visual Studio. Program Visual Studio 2012 nám poskytuje najlepšie prostredie pre jazyk C# z jednoduchého dôvodu - bol vytvorený tou istou spoločnosťou ako primárne vývojové prostredie pre jazyk C#. Zabudovaný Intelli-sense (intelligent sense) veľmi urýchli programovanie. Táto technológia sa prejaví pri snahe automatického ukončenia príkazov počas písania.

S Visual Studiom sa dajú ľahko vytvoriť GUI aplikácie (aplikácie s grafickým užívateľským rozhraním). Na vytvorenie nášho programu sme z dostupných projektov vo Visual Studiu zvolili projekt typu Windows Forms Application. Pri takýchto projektoch sú vo Visual Studiu rovno naprogramované všetky elementy, ktoré potrebujeme na vytvorenie užívatelsky prívetivých aplikácií . Pri vytvorení grafického rozhrania v záložke Toolbox zvolíme komponent (napr. tlačidlo, panel, menu bar) ktorý potrebujeme a umiestnime ho v návrhári výzoru softvéru. Všetky komponenty obsahujú tzv. event handlers, pomocou ktorých môžeme určiť správanie sa programu pri jednotlivých udalostiach, ako napr. stlačenie komponentu myšou.

3.3 3D knižnica DirectX

Microsoft DirectX je zbierka rozhraní pre programovanie aplikácií (application programming interfaces – API) na riešenie úloh súvisiacich s multimédiami, najmä programovanie hier a videí, na platformách spoločnosti Microsoft.

Direct3D (3D grafické API v DirectX) je široko používaný pri vývoji videohier pre Microsoft Windows, Microsoft Xbox 360. Direct3D sa používa aj v iných softvérových aplikáciách pre vizualizáciu a grafické úlohy, akými sú CAD / CAM softvéry.

DirectX Software Development Kit (SDK) sa skladá z runtime knižníc v redistribuovateľnej binárnej forme, sprievodnej dokumentácie a hlavičkových súborov pre použitie pri programovaní. SDK je k dispozícii na bezplatné stiahnutie.

Direct3D 9Ex, Direct3D 10 a Direct3D 11 sú k dispozícii len pre Windows Vista a novšie, pretože každá z týchto nových verzií bola postavená tak, aby závisela na novom Windows Driver Display Model, ktorý bol zavedený pre Windows Vista. Nová Vista / WDDM grafická architektúra zahŕňa nového grafického správcu pamäti podporujúceho virtualizáciu grafického hardvéru pre rôzne aplikácie a služby, ako je Desktop Window Manager [6].

V našom programe využívame verziu rozhrania DirectX 9.0, vďaka čomu máme zabezpečenú bezproblémovú podporu naprieč verziami operačných systémov Windows XP až Windows 8 a veľkého množstva grafických kariet.

Na rozdiel od ostatných časti kódu si pri používaní Direct3D objektov musíme dávať väčší pozor počas programovania. Ako sme už spomenuli, pre Direct3D objekty nie je poskytovaný automatický správca pamäti. O správne uvoľnenie pamäťe sa musíme postarať sami pomocou metódy Dispose(). Konkrétny príklad použitia metódy v našom programe vyzerá takto:

```
foreach (VertexBuffer vbuf in vb)
{
        try
        {
                vbuf.Dispose();
        }
        catch { }
}
```

```
foreach (IndexBuffer ibuf in ib)
{
        try
        {
                ibuf.Dispose();
        }
        catch { }
 }
```

V tejto časti kódu vb predstavuje pole typu VertexBuffer, v ktorom sú uložené vrcholy 3D modelu a ib predstavuje pole typu IndexBuffer obsahujúce indexy vrcholov pre trojuholníkovú sieť modelu. Metódu Dispose() musíme zavolať pre každú časť poľa a na to používame cyklus foreach. Vzhľadom na to, že niektoré objekty v poli nemusia byť nutne inicializované, tak jednotlivé volania metódy Dispose sú v bloku try-catch.

3.4 Optimalizácia

Keďže výpočtovo najnáročnejšia časť nášho programu je samotné načítanie a spracovanie vstupného súboru, zamerali sme sa na optimalizáciu tejto časti. Optimalizáciou jednotlivých krokov pri postupnom spracovaní súboru sa nám nakoniec podarilo výrazne znížiť časovú náročnosť programu. Spracovanie vstupného súboru v softvéri 3D KMaDG Vizualizátor prebieha nasledovne:

- Čítanie dát zo súboru
- Rátanie vrcholov pre model
- Nastavenie farby jednotlivých vrcholov
- Výpočet indexov pre trojuholníkovú sieť
- Výpočet normál jednotlivých vrcholov
- Kopírovanie vrcholov a indexov do bufferov
- Generovanie bitmapy pre Legendu
- Nastavenie kamery pre zobrazenie modelu

Keď sa všetky kroky vykonajú, začína sa renderovanie a zobrazí sa trojrozmerný obraz. V tabuľke 3.1 môžeme vidieť, aké sú rozdiely medzi časmi pred a po optimalizácií. Testovali sme spracovanie súboru na notebooku so 4-

jadrovým procesorom na dátovom súbore s veľkosťou 212MB s 24 miliónmi číselných hodnôt.

Tab. 3.1: Čas jednotlivých operácií v milisekundách

Postup pri otváraní súboru	Čas potrebný na jednotlivé kroky pred optimalizáciou (ms)	Čas potrebný na jednotlivé kroky po optimalizácii (ms)
Čítanie dát	12184	5725
Rátanie vrcholov	2531	304
Nastavenie farby	756	165
Výpočet indexov	70	71
Výpočet normál	4477	443
Kopírovanie bufferov	245	236
Generovanie bitmapy	5	5
Nastavenie kamery	19	18
Celkový čas	20290	6971

Samotné spracovanie súboru po optimalizácií programu prebieha nasledovne. Po výbere súboru sa načíta celý súbor do poľa reťazcov pomocou príkazu File.ReadAllLines(“meno súboru“). Jednotlivé časti poľa predstavujú jeden riadok súboru. Keď už máme uložené všetky hodnoty zo súboru, musíme zistiť rozmery nášho modelu, čiže koľko bodov sa nachádza v riadkoch a stĺpcoch. Analyzujeme vždy prvú hodnotu v každom riadku, ktorá predstavuje zemepisnú šírku. Keď sa táto hodnota zmení, znamená to, že sa už nachádzame v novom riadku, čiže rátame len koľkokrát sa mení prvá hodnota. Takto dostaneme počet našich riadkov. Počet stĺpcov dostaneme tak, že celkovú dĺžku pola podelíme počtom riadkov.

Uložené reťazce musíme konvertovať na číselné hodnoty. Každý riadok rozdelíme na 3 float hodnoty a potom ich uložíme do troch polí. To robíme v cykle. Tu, ako aj na ďalších miestach kódu, sme dosiahli optimalizáciu tým, že pôvodný for(;;) cyklus sme zamenili na Parallel.For(,, =>{}), ktorý je súčasťou .net frameworku vo verzii 4.0 . Tento príkaz vykoná jednotlivé iterácie for(;;) cyklu paralelne na viacerých procesorových jadrách. V tejto časti sa ešte hľadá aj maximálna a minimálna výška, ktorú budeme potrebovať pri Legende, priradení farieb a preškálovaní výšok.

Pri optimalizácií sme ešte presunuli do časti čítanie dát preškálované výšky, ktoré budú predstavovať z-ové súradnice nášho modelu a výpočet súradníc pre zobrazenie na guľovej ploche. Takto môžeme aj tieto hodnoty rátať v paralelnom cykle a nemusíme to robiť pri vytváraní vrcholov modelu. Takto sa nám v kroku, kde vypočítame vrcholy, znížil potrebný čas, lebo už rátame len x a y súradnice na rovinnej ploche, čo robíme jednoduchým súčtom. Z-ové súradnice a súradnice pre guľovú plochu už len kopírujeme z poľa.

Po vytvorení vrcholov nasleduje nastavenie farieb. Každý vrchol má vlastnú farbu nastavenú podľa farebnej škály. Pôvodne sme na výpočet výslednej farby používali zabudované metódy objektu Color, čo však bolo príliš pomalé. Preto sme to zamenili za využitie bitových operácií priamo na jednotlivých farebných RGB zložkách výslednej farby, čím sme dosiahli výrazné zrýchlenie.

Následne vytvárame pole indexov pre trojuholníkovú sieť modelu. Do poľa ukladáme vždy po sebe idúce trojice indexov, ktoré nám indikujú vrcholy pre jednotlivé trojuholníky. Indexy budeme potrebovať aj pri výpočte normál na vrcholoch pre správne fungovanie nasvietenia modelu. Normály počítame tak, že sa vypočítajú normály na každom trojuholníku so spoločným vrcholom a sčítajú sa. Aj tu sa nám podarila optimalizácia. Najprv sme znovu použili paralelizáciu výpočtového cyklu. Oblasť nášho trojrozmerného modelu je rozdelená na viac častí kvôli renderovaniu a tým pádom aj rátanie normál pre jednotlivé časti môže prebiehať paralelne. Okrem toho sme zmenili aj spôsob výpočtu normál. Normály už nerátame pomocou zabudovanej metódy vektorového súčinu Vector3.Cross(“prvý vektor“, “druhý vektor“), ale rátame ich manuálne podľa vzorca vektorového súčinu, čo je rýchlejšie. V kroku výpočtov normál sme dosiahli najväčšie zrýchlenie. Podarilo sa nám znížiť potrebný čas o 90%. Na porovnanie, pri rátaní vrcholov sme dosiahli 87% a pri nastavovaní farieb 78% zníženie.

Teraz, keď už máme všetko potrebné vyrátané, vrcholy spolu s normálami a farbou môžeme skopírovať do objektu typu VertexBuffer a indexy do objektu typu IndexBuffer, ktoré sú následne odoslané do grafickej karty na renderovanie. Pred tým ako sa spustí renderovanie, vygeneruje sa bitmapa predstavujúca Legendu a nastaví sa kamera pre zobrazovanú scénu.

Teraz, po optimalizácií, je najdlhšia časť samotné čítanie súboru z disku. Keďže táto časť je sekvenčná a sme limitovaní rýchlosťou disku, nedá sa dosiahnuť väčšie zrýchlenie.

3.5 Vytvorenie trojuholníkovej siete

Na vytvorenie trojrozmerného modelu potrebujeme vygenerovať trojuholníkovú sieť. Táto trojuholníková sieť pozostáva z vrcholov, ktoré sú rovnomerne rozložené na výpočtovej sieti. Ich rozloženie môže vyzerať napríklad takto:

. . . . .

. . . . .

. . . . .

. . . . .

. . . . .

Obr.3.1 : Rozloženie vrcholov

Aby sme vedeli efektívne uložiť súradnice, farby a normály vrcholov vytvorili sme nasledujúcu štruktúru:

```
struct Vertex
{
        public Vector3 Position;
        public int Color;
        public Vector3 Normal;
}
```

Pri vytváraní vrcholov musíme najprv zvoliť veľkosť oblasti na ktorej bude náš model zobrazený, t.j. rozsah x-ových a y-ových súradníc. Z týchto rozmerov

a z údajov o počte vrcholov v smere x a y môžeme vyrátať vzdialenosti medzi vrcholmi nasledovne:

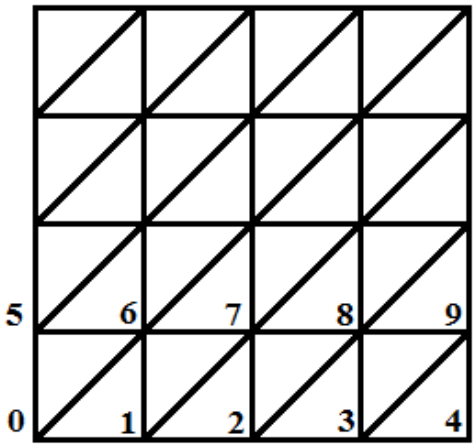
$$krokx = \frac{(maxx - minx)}{(pocetvrchX - 1)}, \quad kroky = \frac{(maxy - miny)}{(pocetvrchY - 1)} \qquad (3.1)$$

Pomocou týchto údajov môžeme rátať x-ové a y-ové súradnice vrcholov v cykle. Budeme ich rátať od ľavého dolného vrcholu a postupovať smerom doprava a hore.

```
poc = 0;
for (int y=0; y<pocetvrchY; y++)
{
        posx=minx;
        for (int x=0; x<pocetvrchX; x++)
        {
                vertex[poc].Position = new Vector3(posx, posy, z[poc]);
                posx+=krokX;
                poc++;
        }
posy+=krokY;
}
```

Hore vidíme, ako sme naprogramovali rátanie súradníc. Premenná poc nám určuje pre ktorý vrchol práve počítame pozíciu. Používame dvojitý cyklus, aby sme postupne prešli každý riadok vrcholov. Pri každom posunutí zvýšime x-ovú pozíciu (posx) o krokX. Keď skončí vnútorný cyklus, znamená to, že sme na konci riadku a teda y-ovú pozíciu (posy) zvýšime o krokY a posx zase vrátime na začiatočnú pozíciu. Takto prejdeme postupne celú sieť. Z-ové súradnice sme uložili do poľa (z[]) pri načítavaní súboru a preto ich už nemusíme rátať.

Po výpočte vrcholov musíme vytvoriť indexovanie trojuholníkovej siete. Indexovanie trojuholníkov bude vyzerať nasledovne:



Obr.3.2: Indexovanie vrcholov

Postupne do poľa (indis[]) ukladáme trojice indexov tak, aby sme vedeli z ktorých vrcholov pozostávajú jednotlivé trojuholníky. Pri ukladaní indexov začneme od ľavého dolného rohu. Najprv zoberieme horný trojuholník a indexy budeme ukladať v smere hodinových ručičiek. Podľa obrázku by postupnosť indexov vyzerala nasledovne: 0,5,6, 0,6,1, 1,6,7, 1,7,2....

Časť kódu ktorý sa stará o správne indexovanie vyzerá takto:

```
poc = 0;
index = 0;
for (int y = 0; y < (pocetvrchY-1); y++)
{
        for (int x = 0; x < (pocetvrchX-1); x++)
        {
                // prvý trojuholník
                indis[poc++] = index;
                indis[poc++] = index + pocetvrchX;
                indis[poc++] = index + pocetvrchX + 1;
                // druhý trojuholník
                indis[poc ++] = index;
                indis[poc ++] = index + pocetvrchX + 1;
                indis[poc ++] = index + 1;
                index++;
        }
        index++;
}
```

4 GUI (graphical user interface) - grafické užívateľské rozhranie

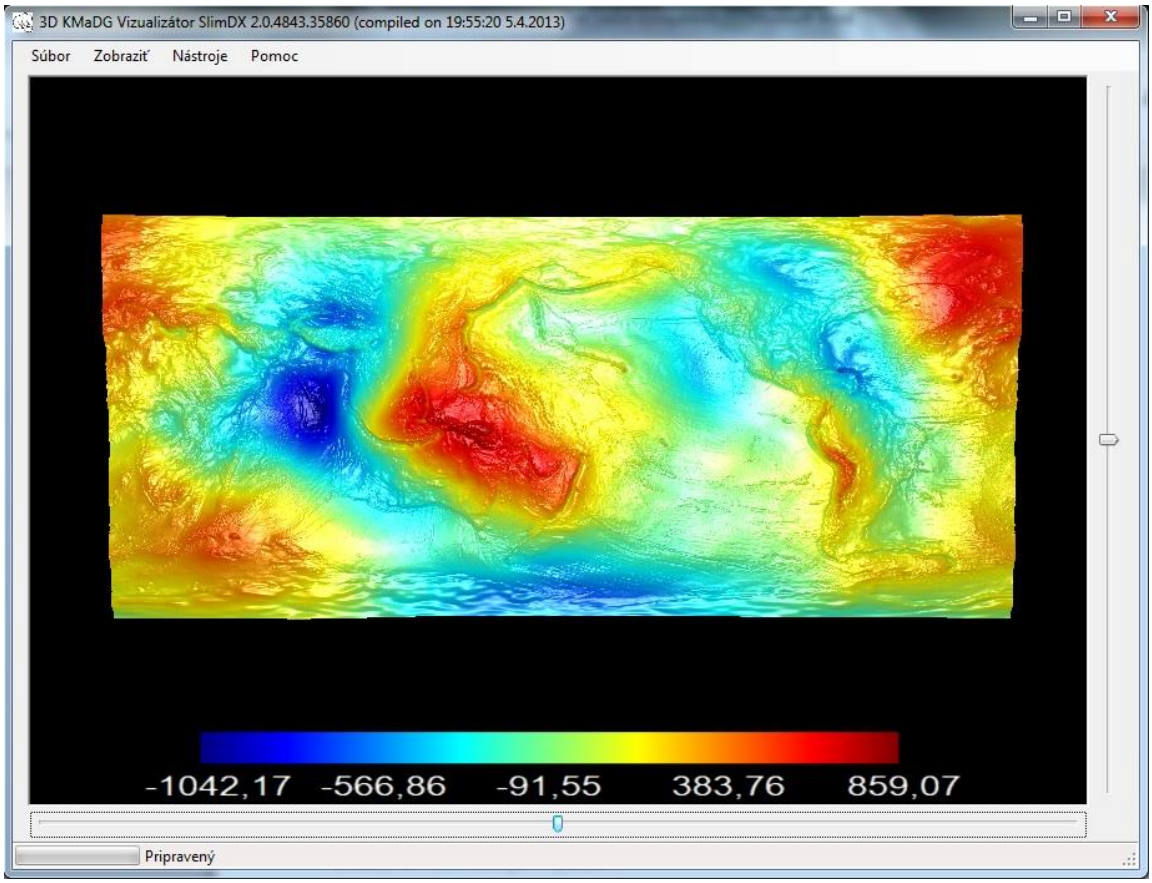
V tejto kapitole predstavíme celkový výzor programu a objasníme ako sa používa. Poskytneme stručnú príručku o tom, čo sa dá robiť s 3D KMaDG Vizualizátorom a načo slúžia jednotlivé položky v menu.

4.1 Vzhľad programu

Pomocou Vizualizátora môžeme zobraziť dáta ktoré boli už opísané v druhej kapitole. Na obrázku 4.1 je vidieť ako vyzerá program po otvorení nejakého súboru. Najväčšiu časť okna obsadí panel, na ktorom sú vizualizované naše dáta. V dolnej časti sa zobrazí legenda, na ktorej je znázornené, ako sú rozdelené z-ové hodnoty podľa farebnej škály. Tu je aj vypísaný číselný rozsah týchto hodnôt. Na ľavom konci je minimum a pravom maximum. Tieto dve hodnoty môžeme za behu programu nastaviť. Ako sa to robí objasníme neskôr.

Správanie sa 3D obrázku v paneli môžeme ovplyvniť pomocou myši. Keď stlačíme ľavé tlačidlo môžeme posúvať obrázok pohybom myši do ľubovoľného smeru. Stlačením pravého tlačidla a posunom myši hore sa môžeme priblížiť k vizualizovanej oblasti a naopak pri posunom dole sa môžeme vzdialiť. Ďalej môžeme s obrázkom manipulovať pomocou dvoch trackbarov, ktoré sa nachádzajú na okraji panela.

S posúvaním trackbaru pod panelom sa 3D obrázok otočí doprava, respektíve doľava. Ak posúvame trackbar vedľa panela, obrázok sa otočí nahor alebo nadol. Správanie sa pri jednotlivých akciách sa môže meniť pri niektorých nastaveniach.

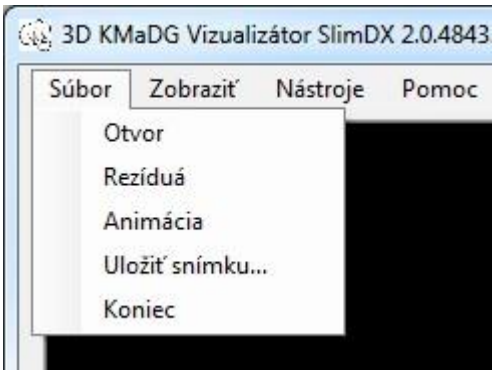


Obr. 4.1 : Vzhľad Programu

V ľavom hornom rohu sa nachádza menu bar s položkami: “Súbor“, “Zobraziť“, “Nástroje“ a “Pomoc“. Pod položkou “Pomoc“ nájdeme stručný popis na ovládanie programu.

4.2 Položka “Súbor“

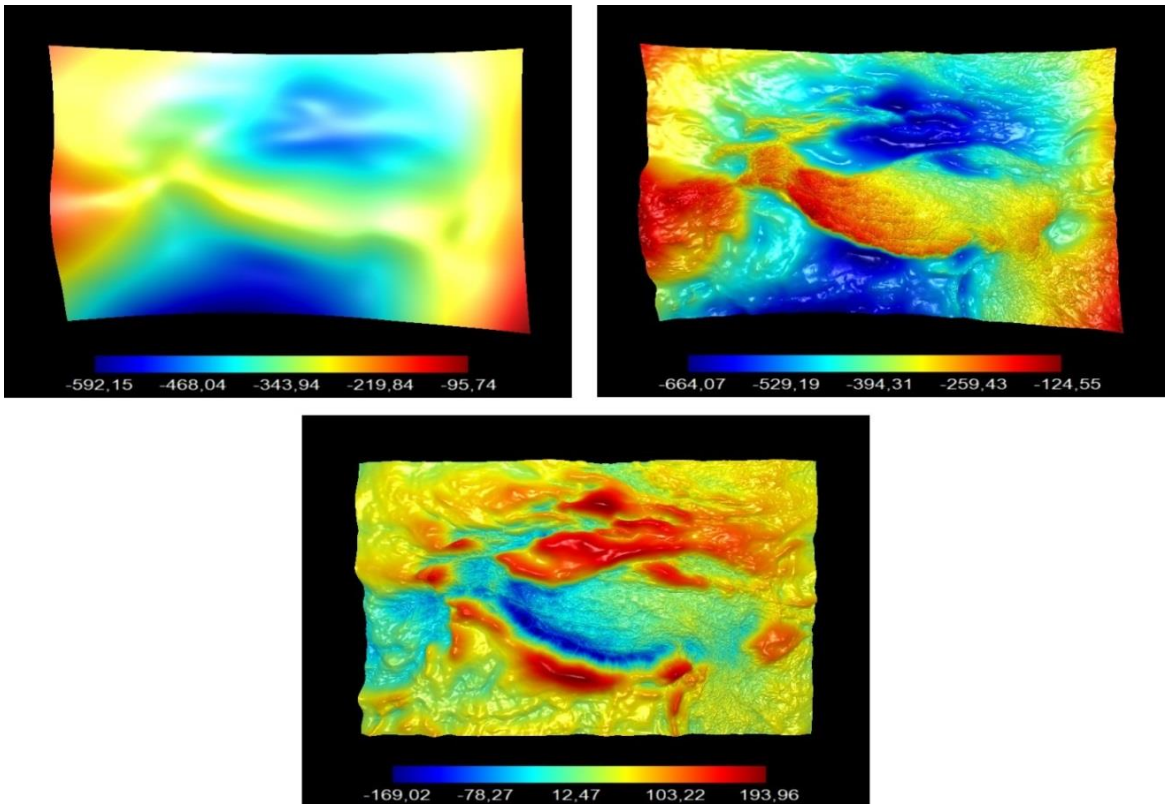
Na obrázku 4.2 vidíme aké máme možnosti pod položkou “Súbor“. Tu môžeme otvoriť súbory troma rôznymi spôsobmi, “Otvor” , “Rezíduá“ , “Animácia“ , a uložiť snímku. Pri všetkých štyroch voľbách sa nám otvorí okno na prácu so súbormi.



Obr. 4.2 : Menu bar “Súbor“

So stlačením “Otvor” a po vybraní súboru, softvér načíta dáta zo súboru do poľa vrcholov, z ktorých sa pri renderovaní vytvorí trojuholníková sieť, t.j. plocha, ktorú uvidíme na paneli.

Pri voľbe “Rezíduá“ môžeme vybrať ďalší súbor, ktorého z-ové hodnoty budú odrátané od z-ových hodnôt už otvoreného súboru. Pri tejto voľbe sa najprv skúma, či sa rozmery dát v súboroch zhodujú. Keď nie, tak program upozorní užívateľa na chybu. Pri správnej voľbe sa zobrazí rozdiel medzi dvoma dátovými súbormi. Takéto zobrazenie môže vyzerať napríklad takto :

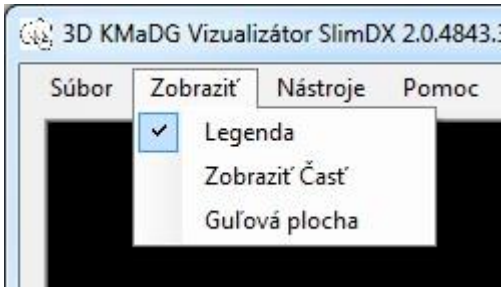


Obr. 4.3 : 3. obrázok dostaneme po odčítaní 2. od 1.

“Animácia“ nám poskytne možnosť na vytvorenie jednoduchej animácie. Pri otvorení súborov v tejto voľbe musíme otvoriť aspoň dva súbory. Program nám postupne zobrazí všetky súbory v takom poradí v akom sme ich otvorili. Pri zobrazení súborov sa prvý obraz pomaly stáva priehľadnejším, kým sa pod ním neobjaví ďalší obraz.

Položka “Uložiť snímku“ nám umožňuje ukladanie vizualizácie do obrázku vo formáte Portable Network Graphics (PNG).

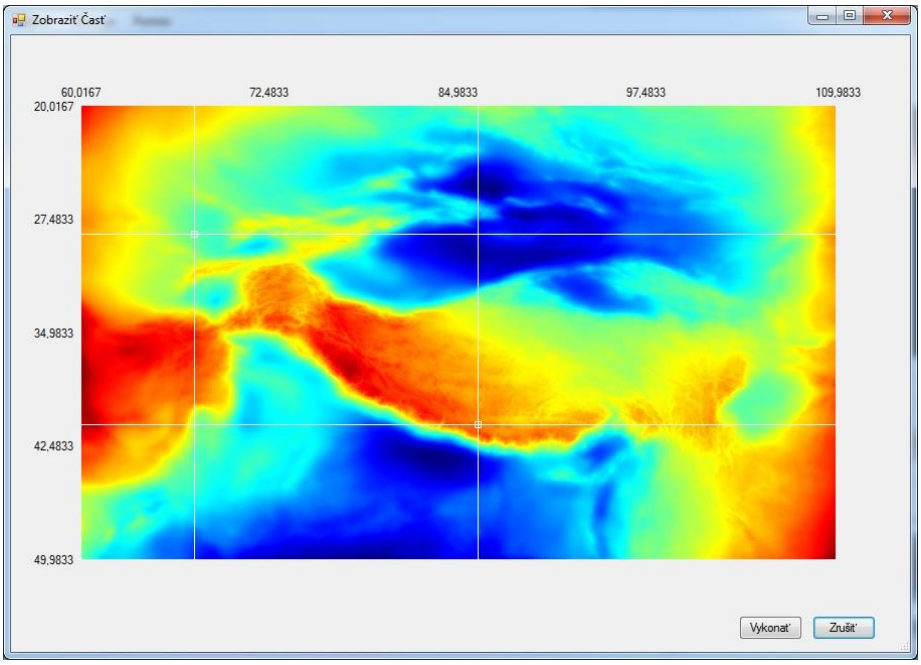
4.3 Spôsoby zobrazenia



Obr. 4.4 : Ponuka “Zobraziť“

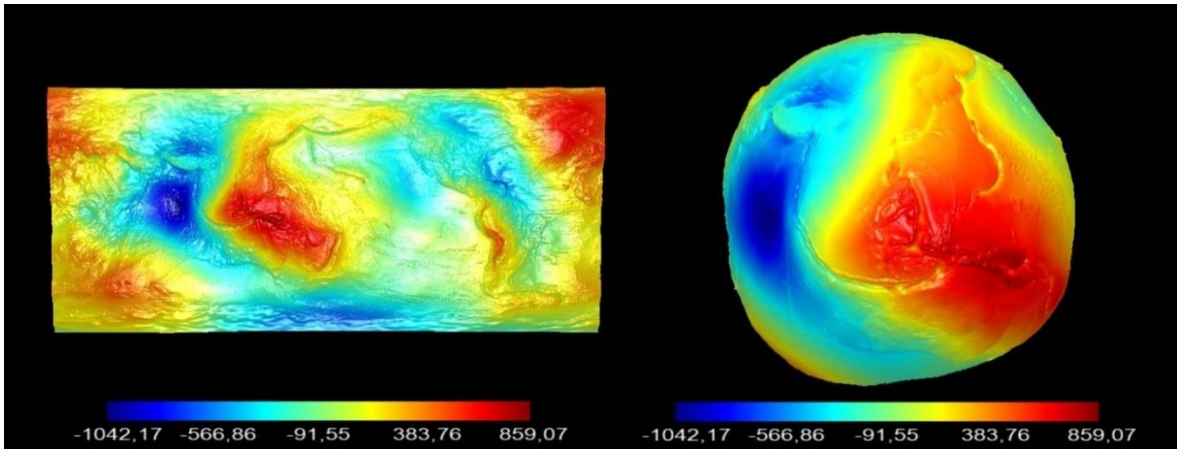
Keď chceme pozmeniť spôsob zobrazenia, môžeme to spraviť pomocou položky “Zobraziť“. Prvé čo môžeme ovládať je viditeľnosť legendy. Keď nechceme zobrazovať legendu alebo ju pri skúmaní našich dát nepotrebujeme, kliknutím na položku “Legenda“ ju môžeme vypnúť.

Keď nechceme zobraziť všetky dáta, ktoré obsahoval vstupný súbor, položkou “Zobraziť Časť“ máme možnosť zvoliť len časť pôvodnej oblasti. Po kliknutí na túto položku sa nám objaví okno, ktoré vidíme na obrázku 4.5 . V okne sa objaví obraz vizualizovaných dát na 2D ploche a na okraji obrazu sú vypísané zemepisné dĺžky a šírky, respektíve rozsah x a y súradníc. Pomocou dvoch pohyblivých štvorčekov môžeme vybrať časť obrázku, ktorú chceme zobraziť. Stlačením gombíka “Vykonať“ sa vykoná zobrazenie a s gombíkom “Zrušiť“ sa okno zatvorí bez zmeny pôvodného zobrazenia.



Obr. 4.5 : Okno na vybratie časti plochy

Z dôvodu, že naše dáta môžu predstavovať aj hodnoty na zemskom povrchu, môže byť vhodné ich zobrazenie na guľovej ploche. Preto sme do nášho programu pridali aj funkciu, s ktorou sa dajú naše dáta zobraziť na guli. Tento spôsob zobrazenia môžeme hocikedy zapnúť počas behu programu, aj keď už máme otvorený súbor, kliknutím na položku “Guľová plocha“. Na obrázku 4.6 vidíme zobrazenie tých istých dát na rovinnej a guľovej ploche.



Obr. 4.6 : Spôsoby zobrazenia

Na to, aby sa dali zobraziť dáta na guli, musíme súradnice vrcholov z našich pôvodných dát prepočítať do kartézskych súradníc. Hodnoty v jednom riadku súboru si označíme písmenami B,L,H, ako geodetické súradnice a výšku nad povrchom gule. Vzhľadom na to, že hodnoty H sú pomerne veľké, preškálujeme si

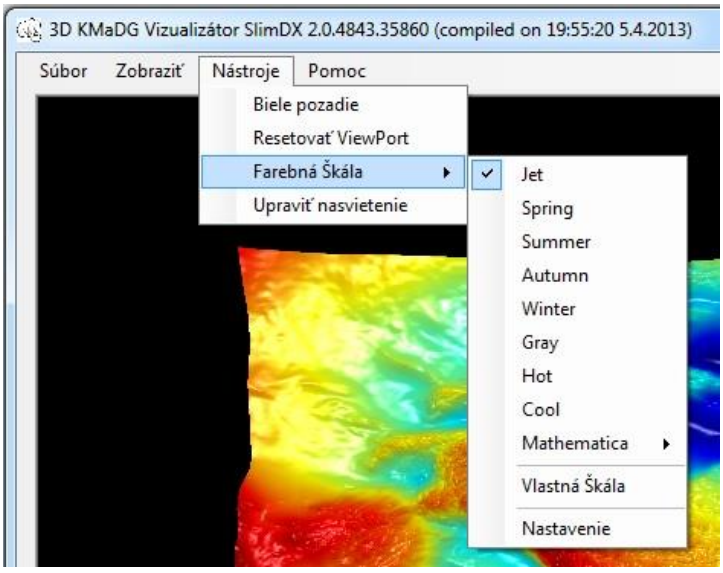
ich na hodnoty z intervalu (0,1). Polomer základnej, jednotkovej gule bude n. Súradnice pre zobrazenie na guli budú vyzerať nasledovne:

$$\begin{aligned} x&=(n+H)*\cos(B*\frac{\mathrm{Pi}}{180})*\sin(L*\frac{\mathrm{Pi}}{180}) \\ y&=(n+H)*\sin(B*\frac{\mathrm{Pi}}{180}) \\ z&=(n+H)*\cos(B*\frac{\mathrm{Pi}}{180})*\cos(L*\frac{\mathrm{Pi}}{180}) \end{aligned} \tag{4.1}$$

Keď máme zapnuté zobrazenie na guli, zmení sa nám ovládanie panela. S ľavým tlačidlom myši sa nebude posúvať trojrozmerný model, ale bude sa otáčať okolo stredu gule.

4.4 Ďalšie nastavenie

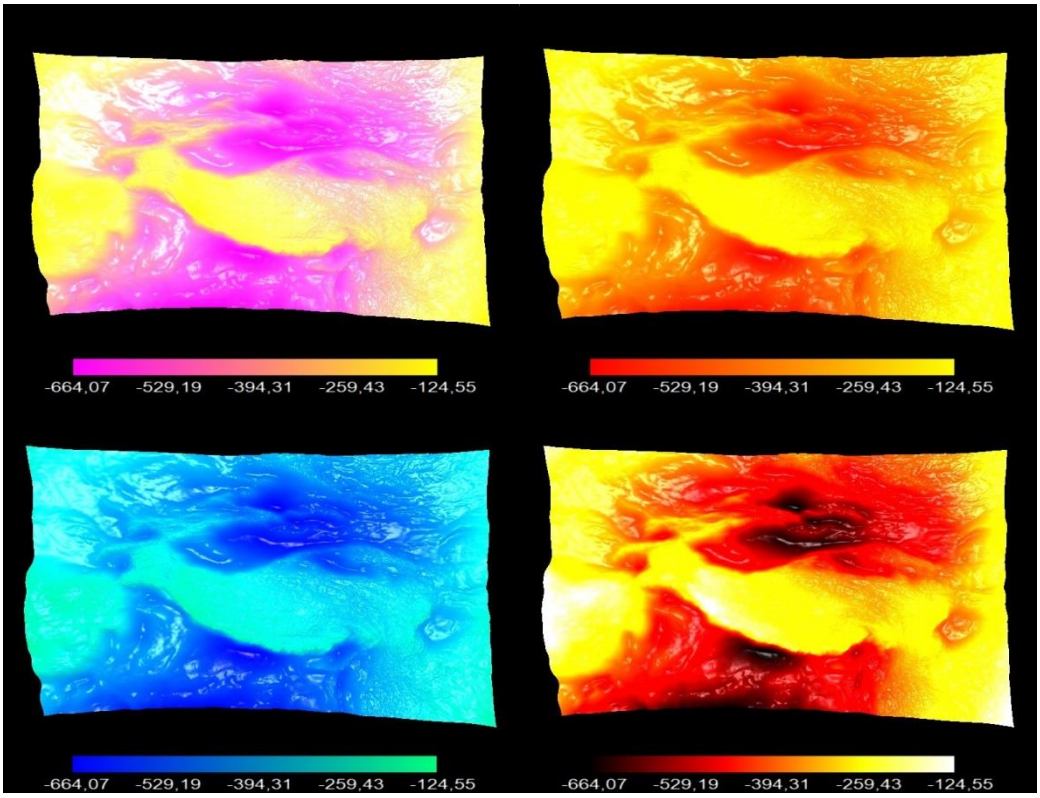
Okrem nastavenia spôsobu zobrazovania dát môžeme nastaviť aj ďalšie vlastnosti našich trojrozmerných modelov. Tými sú napríklad použitá farebná škála alebo spôsob osvetlenia. Tieto možnosti sa nachádzajú pod položkou “Nástroje“, ktorý vidíme na obrázku 4.7. Podrobne popíšeme aké sú tieto nastavenia.



Obr. 4.7 : Menu bar “Nástroje“

Keď zapneme “Biele pozadie“, pozadie panelu sa mení z čiernej na bielu. To môže byť užitočné, keď na zobrazenie dát používame farebnú škálu s tmavšími farbami alebo chceme robiť obrázky vhodné na tlač. Po stlačení voľby “Resetovať ViewPort“ sa vráti natočenie a posunutie modelu na štandardné nastavenie.

V menu “Nástroje“ máme veľa možností na manipuláciu s farebnou škálou, ktorá je použitá na zobrazenie našich dát. Máme 8 štandardných farebných škál: Jet, Spring, Summer, Autumn, Winter, Gray, Hot, Cool [7].



Obr. 4.8 : Ukážky farebných škál : Spring, Autumn, Winter, Hot

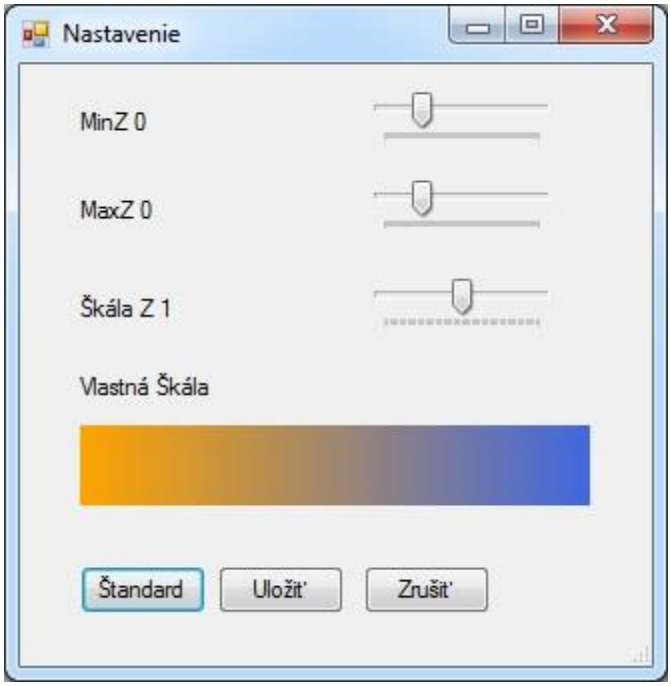
Okrem týchto 8 si používateľ môže zvoliť farebnú škálu z množiny škál získaných zo softvéru Mathematica [8], vybratím voľby “Nástroje“>>“Farebné škály“>>” Mathematica” alebo si môže vytvoriť vlastnú škálu z viacerých farieb.

Pri zvolení jednej farebnej škály program vytvorí maticu s rozmermi 255x3. Každý riadok reprezentuje hodnoty farebného modelu RGB. V stĺpcoch sú ukladané intenzity jednotlivých farebných kanálov, z ktorých pozostává výsledná farba. V našom prípade farba daná hodnotou z-ových súradníc vrcholov trojuholníkovej siete. Pri ofarbení vrcholov transformujeme naše hodnoty na indexy matice nasledujúcim spôsobom:

$$\text{Index} = \begin{cases} 0 & z < Zmin \\ (int)\left(\frac{(z-Zmin)254}{Zmax-Zmin}\right) & Zmin \leq z < Zmax \\ 254 & z \geq Zmax \end{cases} \qquad (4.2)$$

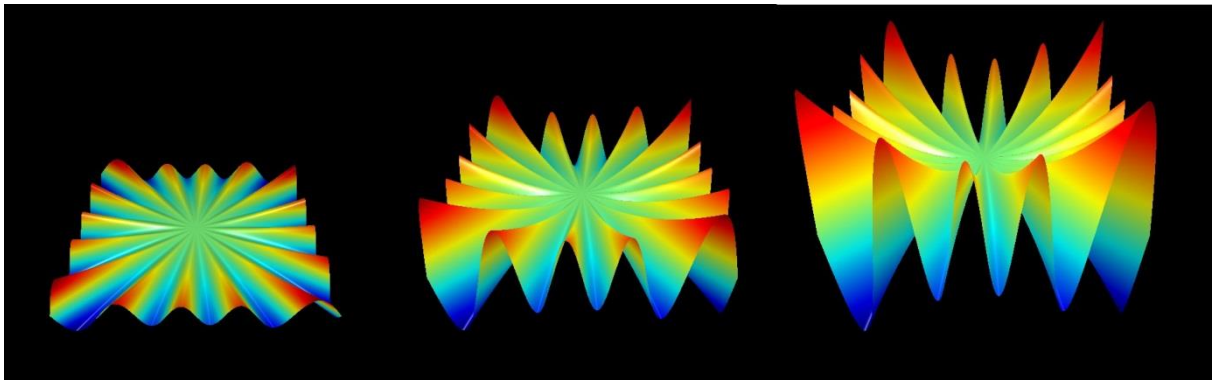
Týmto spôsobom môžeme používať celú farebnú škálu na reprezentovanie našich dát.

Hodnoty Zmin a Zmax sú štandardne nastavené na najmenšiu a najväčšiu z-ovú súradnicu získanú z našich dát. Tieto hodnoty sa za behu program dajú nastaviť, aby sme vedeli lepšie znázorniť tie dáta, ktoré nás viac zaujímajú. Nastavenie maxima, minima ako aj nastavenie škálovania z-ovej súradnice modelu a vytvorenie vlastnej farebnej škály, môžeme urobiť pod položkou “Nástroje >> Farebná Škála >> Nastavenie”.



Obr. 4.9: Okno pre nastavenie maxima, minima a vlastnej farebnej škály

Škálovanie z-ovej súradnice je štandardne nastavené na interval (0,1). Pod položkou “Nastavenie” máme možnosť toto škálovanie prispôsobiť pre vlastnú potrebu. Môžeme tento interval zmenšiť alebo zväčšiť v rozsahu (0,2). Na obrázku 4.10 vidíme ukážku ako sa mení model podľa zvoleného intervalu.



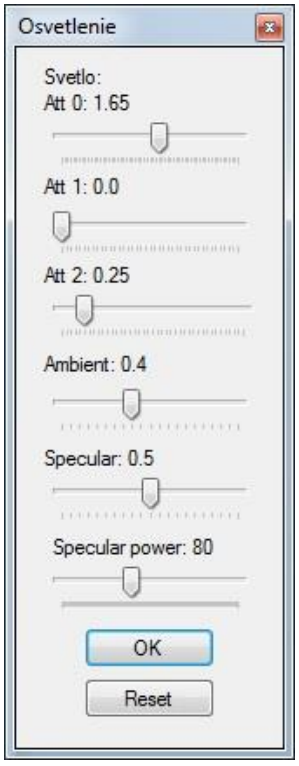
Obr. 4.10: Model s intervalmi (0,0.3),(0,1),(0,1.7) pre z-ovú súradnicu

Farebná škála je štandardne vytvorená z dvoch farieb. Kliknutím na obrázok škály v okne “Nastavenie“ môžeme pridať alebo zmeniť farby na jednotlivých pozíciách. Medzi jednotlivými dvojicami farieb sa vytvorí gradientný prechod. Ten spravíme pomocou lineárnej interpolácie:

$$(R, G, B) = (\mathrm{R1}, \mathrm{G1}, \mathrm{B1}) + ((\mathrm{R2}, \mathrm{G2}, \mathrm{B2}) - (\mathrm{R1}, \mathrm{R1}, \mathrm{R1})) * \frac{\mathrm{i-Index1}}{\mathrm{Index2-Index1}} \quad (4.3)$$

Vo vzorci (R1,G1,B1) reprezentuje prvú farbu, (R2,G2,B2) druhú farbu, i je index pre pole, do ktorého uložíme vypočítanú farbu (R,G,B). Pozícia v škále prvej farby z dvojice je uložená v premennej Index1 a pozícia druhej farby v premennej Index2. Vždy, keď pridáme farbu sa znova rátajú prechody medzi farbami. Program v cykle prejde každú dvojicu farieb, a tak vytvorí celú škálu.

Keď zvolíme “Upraviť nasvietenie“ objaví sa nám dialógové okno, ktoré vidíme na obrázku 4.10. Tu sa dá nastaviť intenzita a ďalšie parametre svetla, pričom zmena parametrov sa priamo aplikuje na zobrazený model v reálnom čase počas posúvania trackbarov.



Obr. 4.11: Okno pre nastavenie osvetlenia

5 Záver

Podarilo sa nám vytvoriť vizualizačný softvér, vďaka ktorému sa dajú pohodlne zobrazovať rozsiahle detailné modely vytvorené z matematických dát. Práca s programom je rýchla a intuitívna, pričom program nemá problém plynule zobrazovať modely zložené z viac ako 16 miliónov trojuholníkov aj na dva roky starom notebooku, takže súčasné generácie grafických kariet budú mať svoje limity postavené ešte vyššie. Zároveň sa dá program využiť aj na zobrazovanie ľubovoľných iných dát, ktoré sú dané na obdĺžnikovej sieti bodov, nemusíme sa obmedzovať iba na geodetické dáta. Program máme dostupný aj so zdrojovým kódom, takže je možné ho v budúcnosti rozšíriť o ľubovoľnú funkcionalitu.

Literatúra

[1] Macák, M., Minarechová, Z., Čunderlík, R., Mikula, K.: Latest improvements in solution of the geodetic boundary value problem by the finite volume method, In MAGIA 2011: Mathematics,geometry and their applications.Proceedings. Kočovce,SR,28.-30.10.2011. 1. vyd. Bratislava: Nakladateľstvo STU, 2012, s. 14--21. ISBN 978-80-227-3780-7

[2] SlimDX, highly polished, fully featured wrapper around almost all Microsoft multimedia and gaming APIs, http://www.slimdx.org/

[3] Špir, R.: Riešenie geodetických okrajových úloh metódou okrajových prvkov. Diplomová práca. Bratislava: SvF STU, 2011

[4] Archer, T: Myslíme v jazyku C#, Grada, 2002, ISBN 8024703017

[5] Svitič, J.: Čo je C#, http://ics.upjs.sk/~rkb/web/s.ics.upjs.sk/_jsvitic/sps/csharp.html

[6] DirectX, In *Wikipedia, The Free Encyclopedia*. Získané 09:05, April 11, 2013, z http://en.wikipedia.org/w/index.php?title=DirectX&oldid=549425185

[7] Dr. Jack J. H. Xu : Practical C# Charts and Graphics http://www.codeproject.com/Articles/18150/Create-Custom-Color-Maps-in-C

[8] Wolfram Mathematica 9.0, Wolfram Research Inc., http://www.wolfram.com/mathematica/