UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

GRAFICKÁ VIZUALIZÁCIA SEIZMICKÝCH VLNOVÝCH POLÍ V 3D MODELOCH

Diplomová práca

Bc. Miroslav Medveď 2013

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

GRAFICKÁ VIZUALIZÁCIA SEIZMICKÝCH VLNOVÝCH POLÍ V 3D MODELOCH

Diplomová práca

Študijný program: Aplikovaná informatika
Študijný odbor: 2511 Aplikovaná informatika
Školiteľ: Doc. Mgr. Jozef Kristek, PhD.

Bc. Miroslav Medveď 2013

Čestné prehlásenie

Čestne prehlasujem, že som túto diplomovú prácu vypracoval len s použitím uvedenej literatúry a návrhov a pripomienok vedúceho diplomovej práce.

V Bratislave 3. mája 2013 .....

Miroslav Medveď

Poďakovanie

Touto cestou by som chcel poďakovať vedúcemu diplomovej práce Doc. Mgr. Jozefovi Kristekovi, PhD. za všestrannú pomoc, cenné rady a pripomienky pri vypracovaní tejto práce.

Zároveň sa chcem poďakovať Doc. RNDr. Romanovi Ďurikovičovi, PhD. za cenné rady a ochotnú spoluprácu pri grafických výpočtoch.

Abstrakt

Autor:	Bc. Miroslav Medveď
Názov:	Grafická vizualizácia seizmických vlnových polí v 3D prostredí
Univerzita:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katedra:	Katedra aplikovanej informatiky
Vedúci diplomovej práce:	Doc. Mgr. Jozef Kristek, PhD.
Počet strán:	55
Rok:	2013

Cieľom práce bolo vizualizovať výsledky numerických simulácií seizmických vlnových polí v 3D nehomogénnych štruktúrach, pričom numerické simulácie sú vykonávané externým programom. Táto práca vznikala vďaka spolupráci Katedry aplikovanej informatiky (KAI) a Katedry astronómie, fyziky Zeme a meteorológie (KAFZM). Práca bola inšpirovaná vizualizačným nástrojom používaným v Southern California Earthquake Center (SCEC), ktorý vytvorili v spolupráci so San Diego Supercomputer Center (SDSC).

Hlavnou myšlienkou práce je však ukázať ako sa programuje softvér pre technické a komerčné účely. Poukazujeme tu nie len na vývoj softvéru pomocou grafickej karty, ale hlavne na algoritmy, ktoré riešia základné problémy pri renderovacích technikách a užívateľsky prístupných rozhraniach pre grafické nástroje. Pre zobrazenie sa využíva prevažne výkon grafickej karty s podporou knižnice OpenGL

Výsledky sme testovali s rôznymi nastaveniami a numerickými simuláciami a zobrazenie sme porovnávali s tým, čo používajú v SCEC.

Kľúčové slová: seizmické vlny, vektor posunutia, scéna, model, frame, API, OpenGL a C++.

Abstract

Author:	Bc. Miroslav Medveď
Title:	Graphical visualization of seismic wave fields at 3D
University:	Comenius University in Bratislava
Faculty:	Faculty of Mathematics, Physics and Informatics
Department:	Department of Applied Informatics
Supervisor:	Doc. Mgr. Jozef Kristek, PhD.
Number of Pages:	55
Year:	2013

This diploma thesis had principal target in visualization of numerically computed seismic wave fields in 3D. The numerical computing of wave fields is made by external program, which may be running at same computer too. The thesis was creates by cooperation of two university departments: Department of Applied Informatics (DAI) and Department of Astronomy, Physics of the Earth and Meteorology (DAPEM). Visual inspiration for this thesis came from San Diego Supercomputer Center (SDSC) which creates application for Southern California Earthquake Center (SCEC).

The idea of this thesis is also illustration of technical programming and commercial utilization. We show how a programmer may development software under GPU (graphical processing unit), which is now available in every average computer. Over and above we have sample of same useful and simple graphical and user-friendly GUI techniques. We used free library OpenGL without any special shader for programming at GPU.

At the final we test the result with commercial product for visualization of numerical computing seismic wave fields which use in SCEC.

Keywords: seismic wave, displacement vector, scene, model, frame, API, OpenGL and C++.

Predhovor

Aplikovanú informatiku som študoval 6 rokov na Fakulte matematiky, fyziky a informatiky na Univerzite Komenského v Bratislave. Posledné 3 roky som sa zameral na odbor umelej inteligencie, na ktorej som sa naučil, ako môžu inteligentne pracovať počítače. Prvú tému diplomovej práce som si zapísal z odboru umelej inteligencie z dôvodu, že som to študoval. Po roku som sa však rozhodol pre terajšiu tému, pretože som ju videl ako pomoc fakulte a hlavne Katedre astronómie, fyziky Zeme a meteorológie (KAFZM). Dozvedel som sa o nej od Prof. RNDr. Petra Mocza, DrSc. na predmete Úvod do fyziky Zeme.

Práca by sa v jednoduchosti dala považovať za načítanie a následne vykreslenie súborov. Za touto prácou sa však skrývajú zložité algoritmy vykresľovania, mnoho užívateľských nastavení a užívateľský príjemné a veríme, že aj jednoduché grafické rozhranie. Seizmológom z KAFZM chýbal moderný program pre vizualizáciu seizmických vĺn a práve preto vznikla táto diplomová práca. Práca sa opiera hlavne o vizualizáciu viacrozmerných dát pomocou grafickej knižnice OpenGL. Knižnica je voľne šíriteľná a podporujú ju bežné počítačové grafické karty. Má skvelú podporu v najpoužívanejších operačných systémoch ako Windows, Linux a MacOS. Zároveň sa táto knižnica ako štandard vyučuje aj na Fakulte matematiky, fyziky a informatiky na Univerzite Komenského. Kód našej práce je dokumentovaný a voľne šíriteľný.

V tejto diplomovej práci sa čitateľ dozvie, ako sa pracuje s grafickou knižnicou OpenGL pod C++ platformou a s frameworkom QtCreator. Prídavkom tejto práce pre čitateľa je aj práca s matematickými komplexnými číslami, ktoré využívajú quaternióny a praktické využitie teórie s virtuálnou rotovacou guľou (ArcBall). V neposlednom rade, sa čitateľ dozvie aj niečo zo seizmológie a problematiky numerických simulácií seizmických vlnových polí. Táto téma je však popísaná len stručne, pretože samotné modelovanie seizmických vĺn nebolo cieľom práce.

Obsah

Zoznam obrázkov

Úvod

Jednou z hlavných úloh súčasnej seizmológie je predikcia účinkov zemetrasenia na záujmovej lokalite. Aj keby sme vedeli predpovedať blížiace sa zemetrasenie (čo zatiaľ nevieme a je veľmi otázne, či to niekedy aj bude možné), aj tak by bolo veľmi dôležité poznať účinky daného zemetrasenia. Na základe takejto predikcie je možné prispôsobiť výstavbu a technológie na zníženie nepriaznivých následkov zemetrasenia. Hlavným nástrojom na predikciu účinkov zemetrasenia, alebo inak povedané na predikciu seizmického pohybu na danej lokalite sú numerické simulácie seizmických vlnových polí. Výsledkom takýchto simulácií je najčastejšie časopriestorové rozloženie posunutia alebo rýchlosti posunutia, čo v reálnych modeloch znamená obrovské množstvo dát. Vhodná vizualizácia týchto výsledkov je preto nevyhnutná pre správnu interpretáciu alebo overenie výsledkov.

Vizualizácia je nezanedbateľná aj preto, lebo zrak je jeden z najviac používaných zmyslov novodobého človeka. V informatike sa hlavne po príchode UML[1] diagramov ujalo, že jeden obrázok niekedy ukáže viac ako tisíce slov. Vhodná grafická virtualizácia prostredia pre to dôležitá pre vnímanie pochopenie zložitých procesov.

Cieľom tejto práce bolo zobraziť numericky simulované seizmická vlnové polia v 3D prostredí. Vizualizácia sa mala zobrazovať prehľadne hlavne pre seizmológov, avšak aj pre ľudí, ktorý o seizmológii a zemetraseniach toho veľa nevedia. Pre správnu vizualizáciu sme čerpali poznatky z predmetov zameraných na grafiku, ktoré sa vyučujú na Katedre aplikovanej informatiky. Druhou dôležitou úlohou je sprístupniť možnosť ľahko a jednoducho pracovať s virtuálnym 3D prostredím ako napríklad využitie virtuálneho ArcBall pre rotovanie scény a objektov a možnosť spravenia krokov vzad a vpred (undo-redo).

[1] Unified Model Language – modelovací jazyk určený pre navrhovanie a špecifikáciu programových systémov

1. Seizmológia

Seizmológia je časťou geofyziky, ktorá sa zaoberá fyzikou seizmického zdroja, vznikom a šírením seizmických vĺn a seizmickým ohrozením záujmových lokalít.

Objektom záujmu seizmológov je buď celé planetárne teleso alebo len pripovrchová časť Zeme. Vďaka analýze seizmických vĺn zo zemetrasení je seizmický model Zeme (model rozloženia rýchlosti šírenia seizmických vĺn) najpresnejším modelom štruktúry Zeme. Seizmológia je nezastupiteľná pri hľadaní nálezísk nerastných surovín (ropa, plyn), pri ktorých sa analyzujú seizmické vlny vygenerované umelo.

1.1. Zemetrasenia

Zemetrasenia väčšinou vznikajú na aktívnych zlomoch (dislokáciách). Dĺžka týchto zlomov môže dosahovať desiatky až stovky kilometrov, pričom ich šírka je maximálne niekoľko metrov. Zlom je oslabená zóna, ktorá oddeľuje dva horninové bloky. Pohybujú sa v rôznych smeroch vzájomnej rýchlosti až niekoľko centimetrov za rok (napríklad pohyb na najznámejšom zlome San Andreas je až rýchlosťou 5cm/rok).

Tektonická platňa je časť Zemskej kôry (povrchu Zeme), ktorá sa inak nazýva aj litosferická doska. Zemský povrch sa skladá z 12 veľkých tektonických platní a desiatok rádovo menších mikroplatní. Litosferické dosky sa posúvajú prakticky neustále, avšak na zlome sa v dôsledku trenia môže hromadiť deformačná energia. Táto energia sa uvoľní po rokoch (niekedy stovky až tisícky rokov) hromadenia v okamihu (10^{-1} – 10^{1} sek.). Rýchlosť odskoku dvoch susediacich bodov na zlome sa pohybuje okolo 10^{2} cm/s. Diskontinuita posunutia[2] sa v hypocentre môže pohybovať v metroch (najväčšie zemetrasenia presahujú 10m). Trhliny, ktoré takto vznikajú sa šíria prakticky všetkými smermi a vďaka nehomogénnosti materiálu je priestor aj čas šírenia trhliny nerovnomerný a zložitý. Rýchlosť šírenia trhliny je spravidla v priemere okolo 0.8 násobok rýchlosti šírenia priečnych vĺn (viď. Sekcia 1.2), no môžu nastať aj prípady, kedy sa trhlina šíri rýchlosťou väčšou ako rýchlosť šírenia priečnych vĺn (supershear).

[2] odskok dvoch na začiatku susediacich bodov

1.2. Šírenie seizmických vĺn

Seizmické vlny sa v Zemskom telese šíria ako elastické vlny. Tieto vlny sú generované nielen zemetraseniami, ale aj pomocou rôznych fázových zmien vo vnútri Zeme, hydrologickou cirkuláciou, zosuvom pôdy, morskými vlnami prípadne obyčajným štartom prúdového lietadla.

Z fyzikálneho hľadiska je šírenie seizmických vĺn popísané pohybovou rovnicou kontinua (tiež elastodynamickou rovnicou):

$$\rho \frac{\partial^2}{\partial t^2} u_i = \frac{\partial}{\partial x_j} \sigma_{ij} + f_i \ , \tag{0.1}$$

kde $\vec{u}(\vec{x},t)$ je vektor posunutia, $\rho(\vec{x})$ je hustota, $\sigma_{ij}(\vec{x},t);\ i,j = 1,2,3$ je tenzor napätia a $\vec{f}(\vec{x},t)$ je vonkajšia sila pôsobiaca na jednotkový objem.

Pri šírení seizmických vĺn sa teleso deformuje, či už v tvare, alebo v objeme. Teleso sa však po ukončení tohto deformačného tlaku môže vrátiť do svojej pôvodnej formy (tvaru a objemu). V tomto prípade hovoríme o elastickom telese. Vzťah medzi napätím a deformáciou elastického telesa popisuje Hookeov zákon.

Hookeov zákon:

Zákon platí len do medze pružnosti, t.j. len pri menších deformáciách. Je charakterizovaný normálovým napätím $\sigma_n = F/S$, kde **F** je veľkosť pružnej sily, ktorá pôsobí na plochu s obsahom **S**. Deformácia rozmerov v ťahu sa objekt s dĺžkou l_0 predĺži na dĺžku l. Tento rozdiel je zapísaný ako $\Delta l = l - l_0$. V praxi sa však častejšie používa pojem relatívne predĺženie: $\varepsilon = \Delta l / l_0$. Deformácia pružných telies je priamoúmerná pôsobiacim silám, čo sa dá zapísať ako $\sigma_n = E\varepsilon$ pričom E – je konštanta modulu pružnosti. Tento modul pružnosti sa dá vypočítať z prierezu objektu (a, b), jeho sily (F) a dĺžky (l): $E = (l^3 F) / (4ab^3)$.

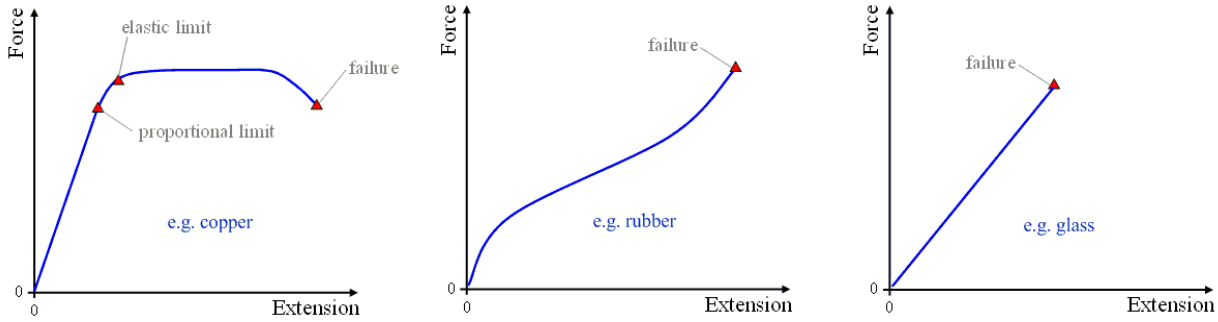
Vo všeobecnosti môžeme zapísať Hookeov zákon pre tenzor napätia v tvare

$$\sigma_{ij} = \lambda \frac{\partial}{\partial x_k} u_k \delta_{ij} + \mu \left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j} \right) , \tag{0.2}$$

kde $\lambda(\vec{x})$ a $\mu(\vec{x})$ sú Laméove elastické koeficienty.

Pre homogénne kontinuum možno pohybovú rovnicu separovať na dve nezávislé vlnové rovnice. Fyzikálne to znamená, že v takom prostredí sa môžu šíriť dva druhy

nezávislých vĺn: pozdĺžna (tiež kompresná, dilatačná) rýchlosťou $\alpha = \sqrt{(\lambda + 2\mu) / \rho}$ a priečna (tiež strižná, rotačná) rýchlosťou $\beta = \sqrt{\mu / \rho}$

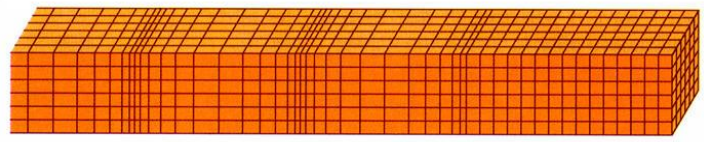


Obrázok 1: príklady pružnosti pomocou hookeových zákonov (kov, guma a sklo) [2]

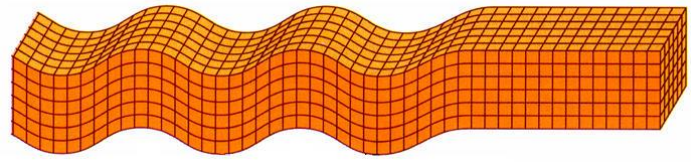
V zložitejších heterogénnych prostrediach sa seizmické vlny sa dajú rozdeliť do dvoch základných typov:

1. Objemové
    a. Pozdĺžne (Obrázok 2: P-vlny)
    b. Priečne (Obrázok 3: S-vlny)
2. Povrchové
    a. Loveove (Obrázok 4: L-vlny)
    b. Rayleighove (Obrázok 5: R-vlny)

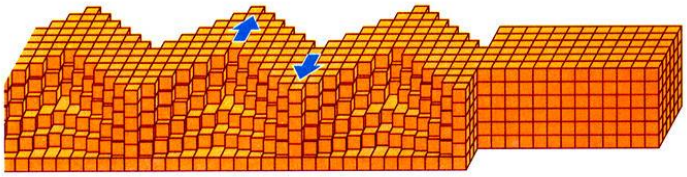
Pre simuláciu šírenia seizmických vĺn je potrebné vyriešiť parciálne diferenciálne rovnice (0.1) a (0.2), čo sa dá analyticky, alebo numericky.



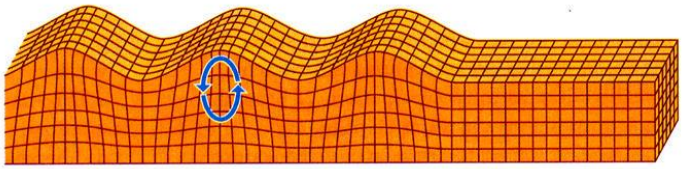
Obrázok 2: Pozdĺžne vlny (P-vlny) [7]



Obrázok 3: Priestorové priečne vlny (S-vlny) [7]



Obrázok 4: povrchové Loveove vlny (L-vlny) [7]



Obrázok 5: povrchové Rayleighove vlny (R-vlny) [7]

Analytické riešenia sú síce presné, ale majú obmedzujúce hraničné a počiatočné podmienky. Používajú sa hlavne pri jednoduchých kanonických modeloch, a preto je pre zložité realistické modely prostredia nutné riešiť rovnice numerickými metódami. Numerické riešenie nie je presné, ale vieme ho ohraničiť chybou menšou ako určené ε. Čím je ε menšie, tým je presnosť vyššia, ale zároveň je vyššia aj výpočtová náročnosť. V podstate sa stále snažíme dosiahnuť čim vyššiu presnosť zachovaním čo najmenšej zložitosti výpočtov. Numerické metódy pracujú s diskrétnymi modelmi.

Výpočtové metódy:

V praxi sa dajú rozdeliť do troch základných skupín:

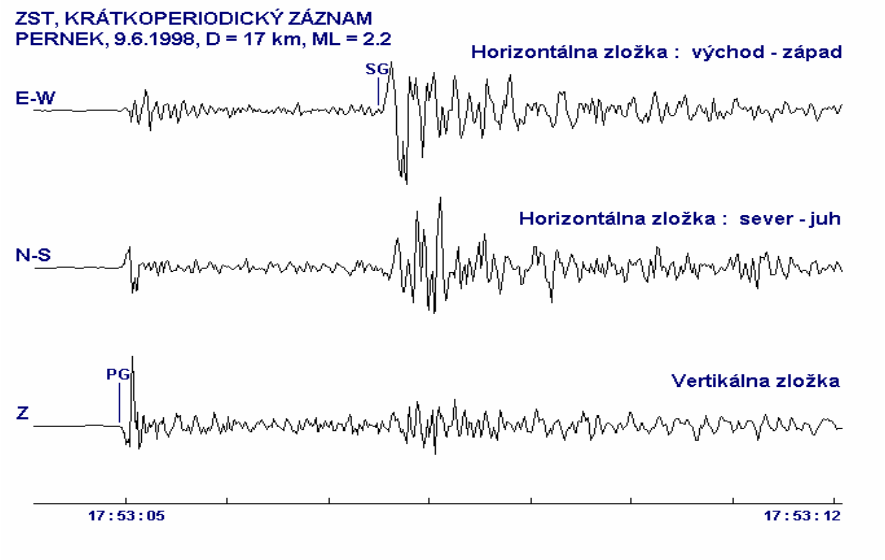
1. Hraničné metódy (metóda hraničných rovníc, hraničných elementov a diskrétnych vlnových čísel)
2. Doménové metódy (metóda konečných alebo spektrálnych prvkov, metóda konečných diferencií a pseudospektrálna metóda)
3. Hybridné metódy

Hraničné metódy sú podstatne presnejšie ako doménové, avšak sú obmedzené len na relatívne jednoduché modely z dôvodu výpočtových nárokov. Metóda konečných diferencií (MKD sa používa hlavne pre jej relatívnu jednoduchosť implementácie. Každá numerická metóda má svoje nedostatky, ktorá je možné potlačiť použitím hybridných metód (napr. spojením metódy konečných diferencií a metódy konečných prvkov).

1.3. Monitorovanie zemetrasení

Seizmický pohyb sa zaznamenáva pomocou špeciálnych zariadení nazývaných seizmometre. Ak sú rozšírené aj o záznamové zariadenie, tak sa nazývajú seizmografy. Tieto prístroje zaznamenávajú pohyby častíc. Ak seizmometer zaznamenáva zrýchlenie, nazýva sa akcelerometer, ak zaznamenáva rýchlosť pohybu nazýva sa velocimeter.

Spravidla seizmometer zaznamenáva všetky tri zložky pohybu: vertikálnu a dve horizontálne orientované v smere zemepisných súradníc: východo-západnú (EW) a severo-južnú (NS).



Obrázok 6: Záznam zemetrasenia v Perneckej ohniskovej zóne seizmografom v Bratislave (Železná Studnička) so silou 2.2 magnitúdo [1]

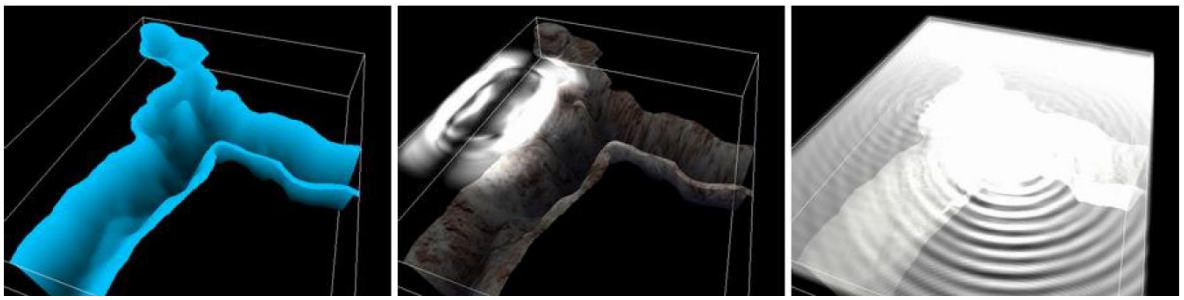
1.4. Vizualizácia numerických simulácií seizmických vlnových polí

Numerické simulácie seizmických vlnových polí produkujú veľké množstvo údajov, ktoré pri monitorovaní reálnych zemetrasení z praktického a ekonomického hľadiska nemôžeme mať. Preto je nevyhnutné výsledky simulácií rozumne vizualizovať.

V súčasnosti existuje niekoľko nástrojov, ktoré napomáhajú vizualizovať numerické simulácie seizmických vĺn. Jedny z najprepracovanejších vizualizačných nástrojov boli vyvinuté v rámci iniciatívy SCEC (The South-ern California Earthquake Center). Tieto boli aplikované najmä na vizualizáciu tzv. ShakeOut scenára možného zemetrasenia na zlome

San Andreas. Popis a príklad ich práce je pribalený v elektronickej forme (viď. prílohy scec1,2).

Prvým pokusom o trojrozmernú vizualizáciu na Slovensku bola diplomová práca V Rácka v roku 2005. Vo svojej práci použil hmlovinu ako nástroj pre zviditeľnenie šírenia vĺn. Tento spôsob sa ukázal nepostačujúci, pretože sa jednalo len o vizualizáciu pomocou bielej farby s rôznou intenzitou. Postupným šírením vĺn vzniklo komplikované usporiadanie hmloviny, z ktorej nebolo jasné v akej hĺbke je ešte seizmický pohyb významný (Obrázok 7). Ďalším nedostatkom tohto softvéru bola nedostatočná možnosť manipulácie s priestorom a nemožnosť vytvárať výseky z priestoru.



Obrázok 7: Ukážka doterajšieho programu [8]

2. Cieľ práce

Cieľom tejto práce je:

- vizualizovať seizmické vlnové pole, ktoré je výsledkom numerických simulácií (z externého programu),
- využiť výkon grafickej karty (GPU) na proces vizualizácie,
- vytvoriť stabilný softvér nezávislý na operačnom systéme,
- vytvoriť užívateľsky príjemné rozhranie na zobrazenie a manipuláciu so scénou, ktoré bude umožňovať:
    - celkový pohľad na vnútornú štruktúru modelu prostredia (vrstvy rôzneho materiálu),
    - vyberanie len určitej časti (výseku) z celkového modelu a následnú vizualizáciu seizmických vĺn len vo vybranom výseku.
- vytvoriť dokumentovaný kód, ktorý bude mať aj edukačnú funkciu základov programovania GPU s využitím OpenGL knižnice.

3. Grafická vizualizácia

Grafická vizualizácia je stručne povedané zobrazenie stavu v grafickej podobe. Projekt pre vizualizáciu seizmického pohybu sa skladá z dvoch rôznych programov. Prvý slúži pre výpočet stavov (modelu) a druhý na jeho vizualizáciu. V spolupráci s doc. Kristekom sme sa zaoberali prevažne grafickou implementáciou. Predpokladom pre správne fungovanie programov bola ich nezávislosť prípadne znovu použitie (vykreslenie pod inými nastaveniami) vypočítaných dát (seizmického vlnového poľa, t.j. vektora posunutí).

3.1. Virtuálne prostredie

Virtuálna realita (virtuálne prostredie) je pojem, ktorý prišiel s príchodom éry počítačov a počítačových simulácií. Je to zobrazenie časti priestoru v pohodlí domova. Pomocou virtuálneho prostredia sa môžu simulovať fyzické procesy, operácie ľudí alebo aj zložité mestské infraštruktúry.

V bežných domácich počítačoch sa pre vizualizáciu prostredia používajú grafické karty s podporou OpenGL a Microsoft ® DirectX ®. Grafická vizualizácia sa oddelila od výpočtovej časti, pretože mnohokrát je na nej nezávislá. Nezávislosť grafického výpočtového zariadenia chce v tejto dobe dosiahnuť spoločnosť NVIDIA v projekte CUDA (viď 5.4 *Cuda*). Druhým vážnym dôvodom oddelenia je kompilovateľnosť vizualizovaných výpočtov. Vykreslenie scény znamená vypočítať scénu a jej farby v 3D priestore, následne usporiadať objekty pre správne poradie. Ďalšou úlohou je zistiť čo sa vôbec zo scény má vykresliť a čo už nie a všetko to preniesť do 2D projekcie (premietanie scény). Všetky tieto časti sa musia zobrazovať v reálnom čase s čo najvyššou obnovovacou frekvenciu (min 30 FPS[3]). Ak by tento vykresľovaní proces bol celý počítaný len pomocou hlavného procesora (CPU), tak ostatné výpočty v programoch by museli byť zjavne potlačené. V skutočnosti to znamená, že niektoré simulácie, ktoré majú zložitejšie výpočty by nemohli bežať v reálnom čase, resp. v počítačových hrách by to mnohokrát znamenalo zhoršenie kvality výpočtov pre kolízie na úkor lepšej vizualizácie a naopak.

Pre premietanie scény[4] vo virtuálnom prostredí sa používa prechod z 3D prostredia na 2D obrazovku (nazývanou priemetňu). Transformácia z trojrozmerného priestoru na dvojrozmerný sa používa z dôvodu, že sa pozeráme na monitor, ktorý zobrazuje scénu

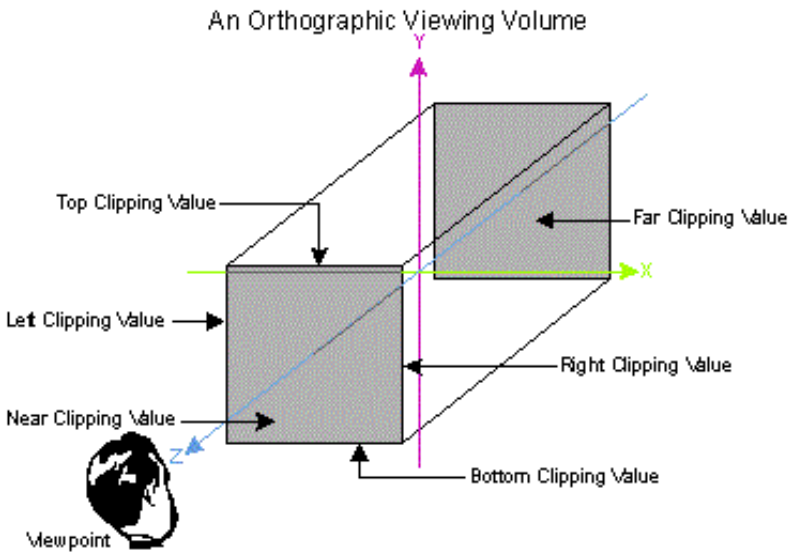
[3] Frame Per Second – počet vykreslených obrázkov za sekundu
[4] Transformácia z n rozmerného priestoru do m rozmerného priestoru, pričom m < n

v dvoch dimenziách, ale scéna je uchovaná a rotovaná, resp. transformovaná v 3D. Poznáme dva základné prístupy premietania:

1. Rovnobežné premietanie
2. Perspektívne premietanie

Rovnobežné premietanie:

Všetky premietacie lúče sú rovnobežné, to znamená, že objekty, ktoré sa nachádzajú ďalej od kamery sú rovnako veľké ako tie, ktoré sa nachádzajú pri kamere. Pri zväčšení a zmenšení objektov v priemete sa tak nepoužíva vzdialenosť od pozorovateľa, ale veľkosť priemetne. Zachováva sa tu rovnobežnosť strán a objektov. Postup rovnobežného premietania sa uplatňuje hlavne pri technických aplikáciách.



Obrázok 8: Rovnobežné premietanie [4]

Ak teda chceme premietnuť objekt z 3D priestoru do roviny XY, tak jednoducho zanedbáme Z-tovu informáciu (x' = x, y' = y). Keďže väčšinou potrebujeme premietnuť obraz na ľubovoľnú rovinu, ktorá nie je rovnobežná so žiadnou z troch rovín (XY, YZ, XZ), tak sa používajú tieto prepočtové vzorce:

$$x' = x - a\frac{ax + by + cz + d}{a^2 + b^2 + c^2}$$

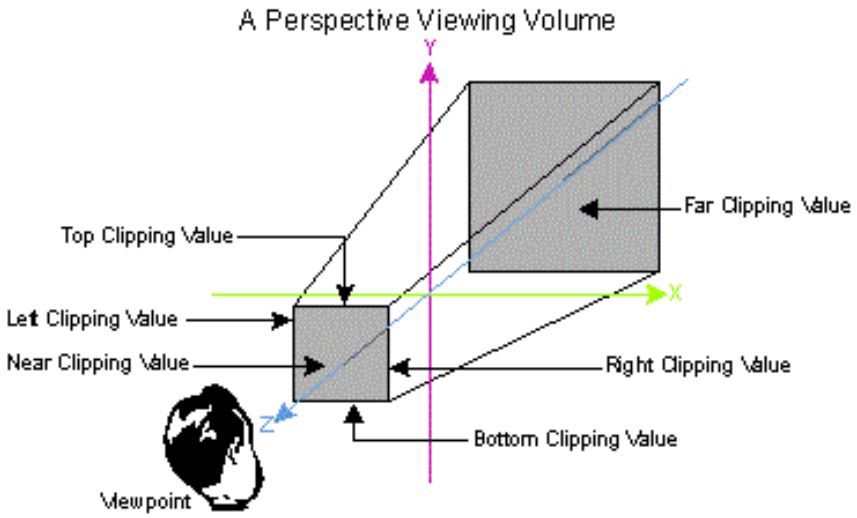
$$y' = y - b\frac{ax + by + cz + d}{a^2 + b^2 + c^2}$$

$$z' = z - c\frac{ax + by + cz + d}{a^2 + b^2 + c^2}$$

a, **b**, **c** a **d** sú predvypočítané konštantné čísla zo všeobecnej rovnice roviny.

Perspektívne premietanie:

Pri perspektívnom (stredovom) premietaní lúče vychádzajú z jedného bodu a postupne sa rozširujú. Perspektívne premietanie sa nazýva aj stredovým, pretože bod z ktorého lúče vychádzajú sa nazýva stred premietania. Keďže obraz vychádza z jedného bodu, tak sa nezachováva rovnobežnosť strán a súbežné roviny sa tak v diaľke stretávajú. Objekty, ktoré sa nachádzajú ďalej od stredu premietania (pozorovateľa) sa nám týmto javia menšie. Perspektívne premietanie funguje aj v normálnom živote, pretože čo je od nás vzdialenejšie, tak sa nám zdá byť menšie, resp. vzdialených objektov sa nám do obzoru zmestí viac. Postup sa používa prakticky vo väčšine scén, kde potrebujeme vidieť reálne zachytenú scénu (Hry, Architektúra, ...).



Obrázok 9: Perspektívne premietanie [4]

Existujú tri druhy perspektívneho premietania:

1. Jednobodové (vzniká ak priemetňa pretína len jedinú os)
2. Dvojbodové (vznikne ak priemetňa pretne dve osi súčasne)
3. Trojbodové (v tomto prípade priemetňa pretína všetky tri osi zároveň)

Rovnica premietnutia sa samozrejme líši od pravouhlej a to aj tým, že keď premietame na rovinu XY, tak napriek tomu, že vynecháme výpočet Z musíme zarátať jeho hĺbku, keďže sa obraz zbieha: $x' = S_z x/(S_z - z)$, $y' = S_z y/(S_z - z)$, $z' = 0$

Pre všeobecný vzorec potrebujeme premietnuť bod **x, y, z** na rovinu zadanú pomocou všeobecnej rovnice **aX+bY+cZ+d = 0** a nech stred premietania (kamera) má súradnice **Sx, Sy a Sz**. Premietací lúč **L** má týmto parametrické vyjadrenie **L = (x+(x-Sx)t, y+(y-Sy)t,**

z+(z-Sz)t). Dosadením do všeobecnej rovnice a vyjadrením parametra **t** dostávame:

$$t = \frac{ax + by + cz + d}{a(S_x - x) + b(S_y - y) + c(S_z - z)}$$

Jednoduchým dosadením parametra tak dostávame nasledujúce vzorce pre perspektívnu projekciu:

$$x' = x + (x - S_x)\frac{ax + by + cz + d}{a(S_x - x) + b(S_y - y) + c(S_z - z)}$$

$$y' = y + (y - S_y)\frac{ax + by + cz + d}{a(S_x - x) + b(S_y - y) + c(S_z - z)}$$

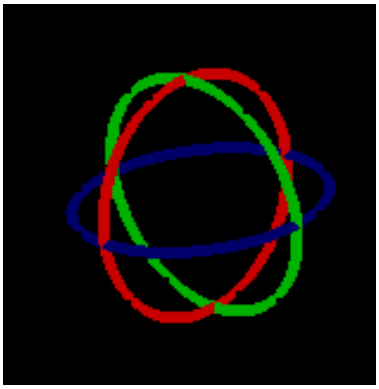
$$z' = z + (z - S_z)\frac{ax + by + cz + d}{a(S_x - x) + b(S_y - y) + c(S_z - z)}$$

3.2. ArcBall

TrackBall je hardvérové zariadenie s guľou, pomocou ktorej užívateľ otáča/rotuje jednotlivé objekty v prostredí, čím vytvára črty scény a scénu samotnú. Guľa zariadenia je uložená na podstavci, pričom užívateľ keď prechádza po povrchu gule, tak sa otáča okolo svojho stredu a podľa toho sa otáča aj scéna, resp. jednotlivé objekty v scéne.

Grafický interface pre manipuláciu s 3D prostredím, ako je jeho rotácia sa nazýva ArcBall. Dalo by sa povedať, že je to virtuálne naprogramovaný TrackBall, ktorý sa používa vo väčšine grafických programoch (napr. 3DsMax, Cinema4D) pre plynule otáčanie objektov. Prvý krát bol implementovaný Kenom Shoemakeom v roku 1985 pomocou quaterniónov (viď 3.2.1 *Quaternióny*). Funkcionalita pre ArcBall funguje štýlom „drag and drop“, t.j. užívateľ klikne a hýbe myšou po obrazovke, pričom objekty ktoré označí, prípadne celá scéna sa rotujú plynulo v priestore. Problém pre takúto implementáciu bol, že obrazovka má len dve osi (X a Y) avšak priestor sa skladá z troch nezávislých osí (X, Y a Z), ktoré sú navzájom kolmé. Prechod v rámci osi X a Y je jednoduchý, no keďže tam vznikla aj hĺbka, tak bolo potrebné implementovať ako pomocou dvoch osí otáčať objekty okolo troch aby to vyzeralo realisticky a bolo jednoduché pre užívateľa naučiť sa s tým pracovať. ArcBall sa skladá z troch kružníc, ktoré sa nachádzajú na povrchu virtuálnej gule. Tieto kružnice značia os okolo ktorej sa objekt aktuálne otáča. Väčšinou sa dajú vybrať jednotlivé osi kliknutím na jednu z kružníc a tak sme to naimplementovali aj do nášho programu. Samozrejme ak užívateľ klikne na 2 osi zároveň, tak sa objekt otáča len okolo týchto dvoch osí.

Nástroj ArcBall je implementovaný pre pohodlie užívateľov a to zväčša grafikov. Ak by mali užívatelia otáčať objekty v scéne len pomocou čísel niekde v panely, tak by to asi nebolo príliš user-friendly. Pre náš ArcBall sme použili quaternióny z dôvodu, že umožňujú rýchle výpočty pre rotovanie objektov, ako je napríklad násobenie. V porovnaní násobenia quaterniónov pri rotácií s násobením matíc, kde je zložitosť približne $O(n^3)$. Ďalšou veľkou výhodou je, že nevzniká GimBall Lock (prekrytie dvoch rotačných osí).



Obrázok 10: Ukážka ArcBall z nášho programu

3.2.1. Quaternióny

Quaternióny sa na začiatku považovali za nevhodný a zbytočný objekt, ktorý dokonca porušoval komutatívny zákon. Ich uplatnenie sa však našlo neskôr v aplikovanej matematike a teoretickej fyzike. V informatike nám slúžia, pre rýchle výpočty pri rotovaní a posúvaní objektov v scéne. Pri rotovaní len okolo troch osí bežne vzniká aj jeden závažný problém nazývaný GimBall Lock, čiže stratenie jednej z rotačných osí pri otáčaní objektu. Pri rotácií sa stáva, že sa 2 osi prekryjú a začnú sa tak pohybovať súčasne (viď Obrázok 11). Pri riešení GimBall Lock problému sa pridala štvrtá informácia a tou je vektor celkového natočenia. Takto sa informácie o natočení objektu uložili do vektora posunutia a troch uhlov (vlastných osí) známych ako Yaw, Pitch a Roll (niekedy sa nazývajú aj Heading, Elevation a Bank). Quaternióny sa prakticky aplikovali v grafike pomocou tohto spôsobu, no však sú jeho vylepšením vďaka používaniu komplexných čísel a násobení. Ich reprezentácia sa dá zapísať ako: qw + **i** qx + **j** qy + **k** qz, pričom **i, j, k** sú komplexné čísla pre ktoré platia určité pravidlá násobenia definované Williamom Rowanom Hamiltonom v roku 1843. Podmienky pre násobenie komplexných čísel v quaternióne sú po svojom objaviteľovi známe taktiež ako Hamiltonovské podmienky.

Tabuľka 1: násobenie komplexných čísel v quaterniónoch

×	**1**	***i***	***j***	***k***
1	1	*i*	*j*	*k*
i	*i*	−1	*k*	−*j*
j	*j*	−*k*	−1	*i*
k	*k*	*j*	−*i*	−1

Každý quaternión **q** má svoj tzv. odraz nazývaný conjugate **K(q)**, pre ktorý platí **q*K(q) = K(q)*q**, pričom tento súčin je nulový, len ako quaternión **q = 0**. Inverzný prvok ku quaterniónu **q** sa hľadá pomocou tohto conjugate $\mathbf{q^{-1} = K(q) / (q*K(q))}$. Element, ktorý bol prenásobený quaterniónom dostaneme do pôvodnej hodnoty pomocou conjugate toho istého quaterniónu. Pre rotáciu objektov pomocou dvoch quaterniónov je potrebné ich prenásobiť. Pri násobení dvoch quaterniónov si musíme dať pozor nielen na ich poradie (pre komplexné čísla viď Tabuľka 1), ale aj na to, že sa nám nový quaternión predĺži, keďže uchováva v sebe aj vektor posunutia. Z toho vyplýva, že po otočení objektu pomocou dvoch quaterniónov je potrebné normovať dĺžku vzniknutého quaterniónu. Normalizácia sa prevádza pomocou dĺžky (normy) quaterniónu **|h| = q*K(q)**. Quaternión je normovaný ak jeho dĺžka je jedna. Normalizácia je nutná, ak objekt nechceme prenášať, len ho rotovať, čo je aj náš prípad.

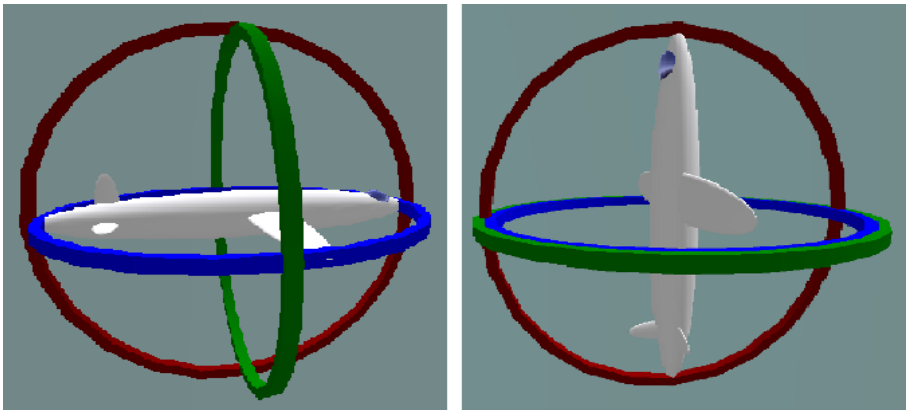
Keďže jednotlivé body v scéne sa v OpenGL prepočítavajú pomocou afinných transformačných matíc, tak bolo potrebné vytvoriť maticu z týchto quaterniónov. Pre transformáciu existujú vzorce na prepočet z quaterniónov, ktoré sú zobrazené v nasledujúcej tabuľke (Tabuľka 2).

Tabuľka 2: prevod quaterniónov na maticu rotácii

$1 - 2*qy^2 - 2*qz^2$	$2*qx*qy - 2*qz*qw$	$2*qx*qz + 2*qy*qw$
$2*qx*qy + 2*qz*qw$	$1 - 2*qx^2 - 2*qz^2$	$2*qy*qz - 2*qx*qw$
$2*qx*qz - 2*qy*qw$	$2*qy*qz + 2*qx*qw$	$1 - 2*qx^2 - 2*qy^2$

Prefix **q** znamená, že sa jedná o veličinu quaterniónu. Ako vidno v tabuľke je každé násobenie použité dvakrát. Pre jedno číslo v matici sa takto v podstate využije len jedno násobenie, čo nám umožní prechod z quaterniónu do matice rotácie len pomocou 9 násobení. Pri bežnom násobení dvoch matíc je zložitosť $O(n^3)$, čo v našom prípade znamená $4^3 = 64$ násobení pri rotácií. Pri rotovaní objektu pomocou násobení dvoch

quaterniónov je zložitosť len 16 obyčajných násobení čísel. Pomocou quaterniónov získavame nielen vyriešenie problému GimBall Lock, ale aj niekoľko násobne menej násobení pri rotácií objektu pomocou afinných transformačných matíc. Mnohokrát sa pri rotácii objektov viac oplatí takto premietnuť dve transformačné matice na quaternióny, ktoré sa pretransformujú na tretí quaternión. Tento výsledný quaternión sa následne premietne na výslednú transformačnú maticu. Tento postup nie je však úplne bežný, ale je efektívnejší ako násobenie dvoch matíc.



Obrázok 11: Ukážka GymBall Lock (v pravo vzniklo prekrytie dvoch osi) [2]

3.3. Vizualizácia viacrozmerných dát

Trojrozmerné dáta sú našou každodennou súčasťou. Málokto si uvedomuje vnímanie viac ako trojrozmerných dát len očami. S vizualizáciou viac ako trojrozmerných dát sa však stretávame každodenne, len si to neuvedomujeme. Dobrý príklad je vizualizácia predpovede počasia, keď rosnička ukazuje na mapu Európy, kde sú jednotlivé obrázky ako slniečka, alebo mraky s bleskom. Na týchto mapách vidíme ďalší rozmer vnímania a to či bude jasno, alebo zamračené, prípadne, či nás čakajú na nasledujúci deň búrky. V seizmológii sa pri vizualizácií seizmických vĺn používajú prevažne tri základné prístupy:

1. Farebná škála (Obrázok 12)
2. Vysunutie povrchu
3. Hybridné zobrazenie

Farebná škála:

Čim sa Zemská kôra viac posunula oproti svojej pôvodnej polohe, tým je intenzita farby jasnejšia. Škála je priložená ako legenda, pričom farbám sú priradené numerické

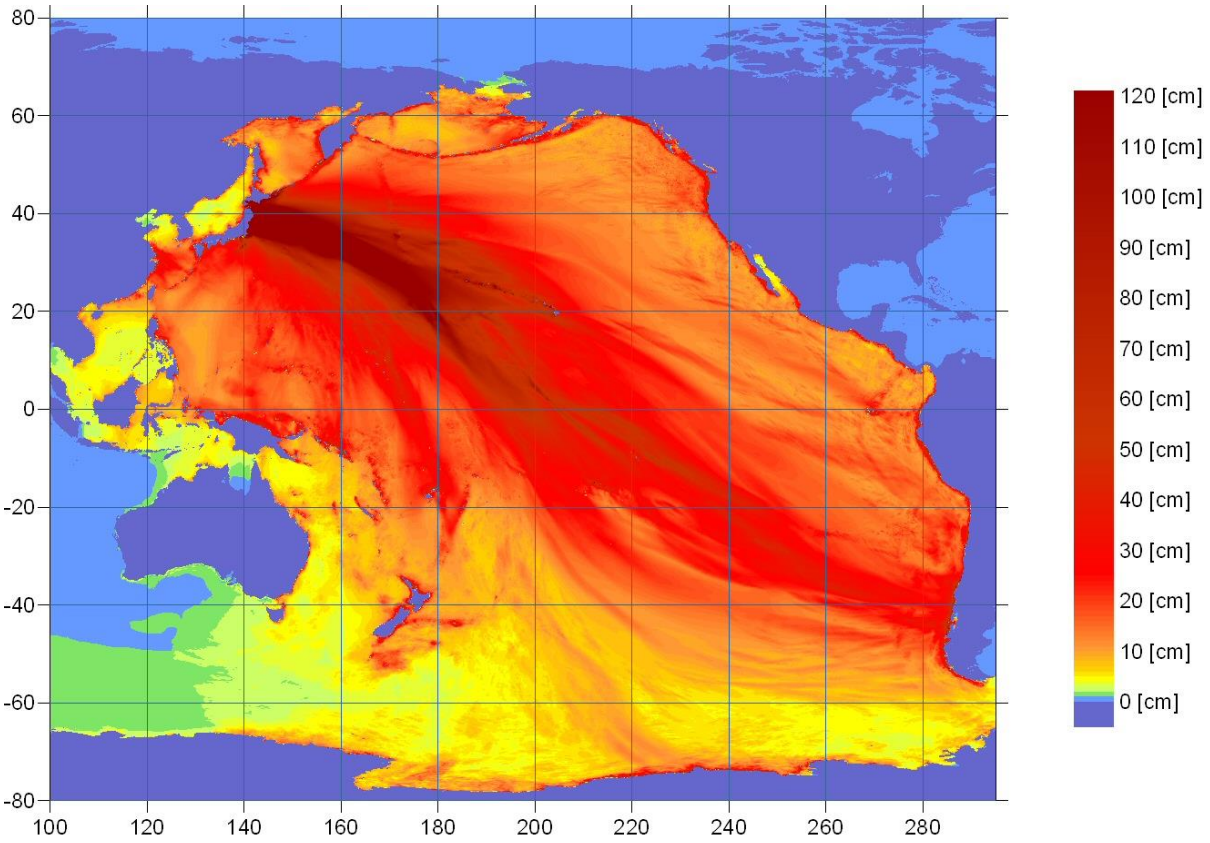
hodnoty väčšinou v cm pre metrické systémy. Jej použitie je univerzálne a nedeformuje model povrchu. Pre jej jednoduchosť vnímania sa prevažne používa pre 3D zobrazenia modelov šírenia seizmických vĺn.

Vysunutie povrchu:

Používa sa hlavné pre 2D zobrazenia povrchov, alebo určitých vrstiev. Postup je zrejmý, ak sa litosferické dosky od seba posunuli, tak z tohto bodu bude vyčnievať štít. Čím je vektor posunutia väčší, tým bude aj vysunutý štít vyššie. Následná mapa ktorá týmto vznikne, vyzerá ako fotografia pohoria. Výhodou je, že intenzita posunutia nie je obmedzená farebnou škálou, ale je prakticky nekonečná. Druhou výhodou je, že tento obraz si môžu pozrieť aj ľudia so zníženým vnímaním farieb (farboslepí ľudia).

Hybridné zobrazenie:

Spojením predošlých dvoch zobrazení vznikne farebná mapka z ktorej ako keby vychádzali pohoria. Používa sa často hlavne pre vizualizáciu posunu seizmických vĺn na povrchu.



Obrázok 12: Príklad znázornenia posunov v dôsledku zemetrasenia [9]

3.4. Farebná škála

Farebná škála je jedna z možností ako zobrazovať finálne viacrozmerné dáta z určitých meraní. Pri výslednom (renderovanom) zobrazení je zvykom napísať vysvetlivku/legendu k jednotlivým farebným prvkom. Základné farebné škály sa zmiešavajú pomocou dvoch modelov: RGB[5] a CMYK[6]. Ak potrebujeme odtiene istej farby, tak sa používa HSL, resp. HSV farebný model, ktorý sa však prevádza do RGB, pretože pomocou týchto troch základných farieb sa vykresľujú objekty na monitor.

RGB:

Obsahuje tri farebné svetelné zložky pomocou ktorých sa miešajú všetky pre človeka viditeľné farby. Základné zložky sú červená (Red), zelená (Green) a modrá (Blue). Farby sa miešajú aditívnym spôsobom, t.j. sčítaním základných farebných zložiek. Pomocou RGB modelu sa spájajú aj svetelné lúče. Vďaka tomuto spôsobu vykresľovania nám fungujú staršie CRT[7], alebo aj nové LCD[8] monitory. Zmiešaním všetkých troch zložiek nám vznikne biela farba, takže pozadie monitorov je čierne a tým pádom pri vykreslení čiernej farby sa všetky tri farby modelu nastavia na nulu.

CMYK:

Skladá farby pomocou troch základných farieb určené pre miešanie viditeľných farieb subtraktívnou metódou, t.j. odčítaním. Základné farby, ktoré sa tu používajú sú azúrová (Cyan), purpurová (Magenta) a žltá (Yellow). Štvrtou zložkou je čierna farba (Black), pretože je fyzicky náročné dosiahnuť efekt čistej čiernej farby za použitia len týchto troch zložiek. Ďalším dôvodom prečo sa používa čierna je ekonomický, pretože náklady na čistú čiernu farbu na papieri sú niekoľko násobné nižšie keď sa použije osobitný cartridge ako keby sa mala miešať z ostatných troch farebných zložiek. Touto technológiou sa tlačia obrazy biely povrch akým je napríklad papier. Medzi CMYK a RGB modelom je prepojenie viazané jednoduchým vzorcom: C = (255 - R); M = (255 - G); Y = (255 - B). Ak pomocou RGB modelu potrebujem vytvoriť napríklad azúrovú farbu, stačí na to použiť zelenú a modrú zložku. Pomocou predošlého vzorca vlastne vidíme aj to, že tieto dva základné modely sú svojím opakom.

[5] **R**ed **G**reen **B**lue
[6] **C**yan **M**agenta **Y**ellow Blac**k**
[7] **C**athode **R**ay **T**ube
[8] **L**iquid **C**rystal **D**isplay

HSV (HSB):

Je to farebný model založený na troch zložkách. Prvou zložkou je farebný tón (Hue), ktorý určuje o akú farbu z farebného spektra (Obrázok 13 resp. Obrázok 14) ide a to pomocou stupňov z kruhu (0° – 360°). Druhou hodnotou je sýtosť farieb (Saturation) taktiež nazývaná silou, alebo čistotou farby. Jej hodnota sa mení pridaním ostatných farieb v forme sivej. Hodnota sýtosti farby sa meria v percentách (1 – 100%). Poslednou hodnotou farebného modelu je hodnota jasu (Value/Bringhtness) farby, t.j. množstvo bieleho svetla. Taktiež by sa dalo povedať, že jas znamená pridávanie čiernej farby do základu.

Prevod z RGB:

Najprv si musíme vypočítať maximálnu (**max**) a minimálnu (**min**) hodnotu z RGB farieb. Každá zložka z RGB modelu bude mať priradenú farbu pomocou reálneho čísla z intervalu 0 až 1. Hodnota max sa zoberie ako maximálna hodnota z jednotlivých farebných zložiek červenej, zelenej a modrej (max(r, g, b)). Minimálna sa berie obdobne (min(r, g, b)). Hodnota Hue sa počíta na základe výsledku min a max hodnoty nasledovne:

If (min == max) { H je nedefinované }

If (red == max) {

If (green >= blue){ $H = 60° * \frac{green - blue}{\max - \min} + 0°$ }

else { $H = 60° * \frac{green - blue}{\max - \min} + 360°$ }

}

If (green == max) { $H = 60° * \frac{blue - red}{\max - \min} + 360°$ }

If (blue == max) { $H = 60° * \frac{red - green}{\max - \min} + 360°$ }

Sature sa počíta trošku jednoduchšie:

If (max == 0) { S = 0 }

else { $S = \frac{\max - \min}{\max} = 1 - \frac{\min}{\max}$ }`

Do Value resp. Bringhtness sa priradí max:

V = max

HSL:

Ako napovedá názov, tak prvé dve zložky sú podobné, avšak mení sa len tretia zložka a tou je svetlosť (Lightness). Vďaka svetlosti tak na nastavenie čiernej, alebo bielej farby nestačí len jas ako to bolo v modely HSV, ale je potrebné kombinovať svetlosť so sýtosťou farby.

Prevod z RGB:

Hodnota Hue sa počíta rovnako ako v modely HSV, avšak Sature sa prepočítava trošku ináč so závislosťou na svetelnosti:

$$L = \frac{max + min}{2}$$

If (L == 0 alebo max == min){ S = 0 }

$$\text{If } (0 < L <= \frac{1}{2})\{ S = \frac{max - min}{max + min} \}$$

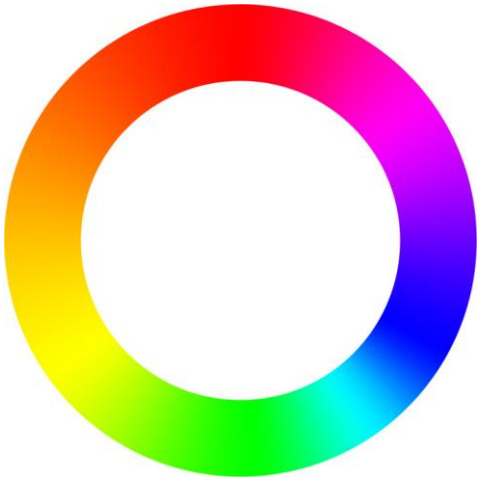
$$\text{If } (\frac{1}{2} < L)\{ S = \frac{max - min}{2 - (max + min)} \}$$

Základná a najpoužívanejšia farebná škála sa skladá z jednoduchých prevodov modrej do zelenej a zo zelenej do červenej. Táto škála vznikla z lámavosti svetla, keďže svetlo sa zlomí na šesť základných človekom viditeľných farieb. Človek je schopný začať vnímať farby od fialovej, ktorá má vlnovú dĺžku okolo 390 nm (približne 750 THz). Pod touto hodnotou vnímania vlnovej dĺžky sa svetlo nazýva ultrafialové. V niektorých odborných literatúrach sa rozdeľuje aj ultrafialové svetlo na vzdialené a blízke. Blízke ultrafialové svetlo sa pohybuje okolo 300 nm (1000 THz) a vzdialené sa udáva, ak je nižšie ako 200 nm (čiže viac ako 1500 THz). Koniec ľudského vnímania svetla je červené farebná vlnová dĺžka, ktorá sa pohybuje okolo 780 nm (približne 400 THz). Nad touto hranicou ľudského vnímania farieb sa nachádza infračervené svetlo, ktoré sa však definitívne udáva až za hranicou 1000 nm (menej ako 300 THz).

Algoritmy pre farebné škály sa líšia hlavne plynulosťou prechodov ako môžeme vidieť aj na nasledujúcom obrázku (Obrázok 13: Farebné spektrum) alebo aj začiatkom, prípadne koncom farebného spektra.

Obrázok 13: Farebné spektrum

Keďže na začiatku lomu svetla sa nachádza červená farba (časť fialovej) a aj na konci spektra je červená, tak sa týmto uzatvára do kruhu (viď Obrázok 14). Práve pre tento uzavretý plynulý farebný kruh sa časť z jeho začiatku/konca zvykne vyseknúť, aby farebný prechod nebol natoľko plynulý medzi prvou a poslednou hodnotou. V našom programe sme sa preto rozhodli useknúť fialovú farebnú dĺžku. Takže najmenší (nulový) časticový posun je označený modrou s jemným nádychom fialovej nazývanej indigo. Koniec spektra, t.j. farba s najvyšším posunom má sýto červený odtieň.



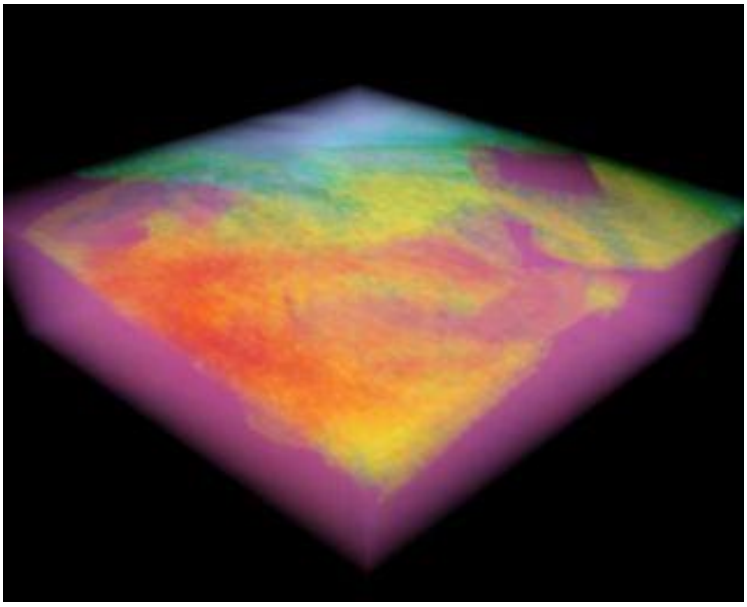
Obrázok 14: Uzavreté farebné spektrum [10]

3.5. Hmla

Efekty hmly sa v grafike využívajú pre realistickejšie zobrazenie niektorých scén. OpenGL, ktoré sme sa rozhodli využívať pre náš projekt vizualizácie seizmických vĺn má vo svojom základe taktiež efekt hmly. Tento základný efekt sa však nedá nastaviť na naše požiadavky. Hmla sa v OpenGL dá nastaviť len v celkovom pohľade na objekty. Objekty ktoré sú ďalej od kamery sú viac a viac zastreté hmlovinou, ktorá ich prekrýva podľa hustoty, až nakoniec nebudú viditeľné. OpenGL API dokonca nepodporuje ani prízemnú hmlovinu. Druhým nedostatkom hmly v OpenGL je, že môže obsahovať len jedinú farbu, takže sa nedá spraviť farebný prechod hmly.

V našom programe sme potrebovali efekt, ktorý by zviditeľnil aj vnútorné štruktúry modelu seizmických vĺn. Spoločnosť Southern California Earthquake Center (SCEC) s tým pomáhala spoločnosť San Diego Supercomputer Center (SDSC). Spoločnosť SDSC pod vedením Amit Chourasia a Steve Cutchin vytvorila volumetrickú vizualizáciu znázornenú

na nasledujúcom obrázku (Obrázok 15). My sme sa snažili dosiahnuť tento efekt najprv jednoduchým spôsobom rozmazaných bodov.



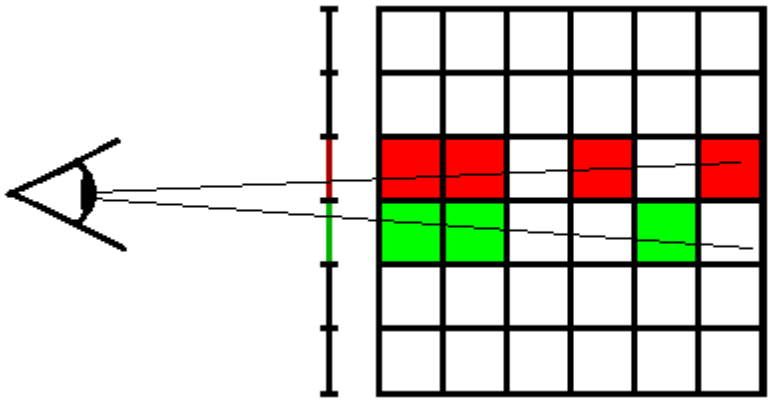
Obrázok 15: Vizualizácia SCEC dát spoločnosťou SDSC [6]

OpenGL ponúka možnosť rozmazania bodu pomocou funkcie SPRITE. Funguje to na princípe, že každému bodu sa priradí špeciálna textúra, ktorej stred je jasný, ale prechádza do tmavých okrajov. Textúra je následne zafarbená pomocou farby bodu. Postup nám však nevyvolal správny efekt, pretože pri hlbších vizualizáciách vznikali nežiaduce efekty sčítania farieb pri priehľadnosti. Mnoho farieb sa sčítalo až na čistú bielu, prípadne ak sa nastavila nízka priehľadnosť jednotlivých bodov, tak vnútorná štruktúra nebola dostatočne viditeľná. Nakoniec sme zistili, že sa dá dosiahnuť tento efekt pomocou Volume Ray-Casting algoritmu. Bohužiaľ pre zložitosť tohto algoritmu sme nestihli implementovať na náš program túto teóriu. Napriek tomu sme sa rozhodli, že túto metódu popíšeme aspoň v skratke, pretože v budúcnosti by sme určite chceli pokračovať v tomto projekte a to hlavne rozšírením vizualizácie o práve algoritmus Volume Ray-Casting.

3.5.1. Volume Ray-Casting

Ray-Casting je najpoužívanejšia technika pre vizualizáciu objemových dát. Funguje na princípe určitých lúčov, ktoré vychádzajú od pozorovateľa až k objektom. Každý objekt je zložený z voxlov, ktorým keď pretne jeden lúč, tak všetky farby sa prepočítajú pomocou zadaných vzorcov. Výsledná farba sa zapíše do bufferu. Príklad si môžete pozrieť na

nasledujúcom ilustračnom obrázku (Obrázok 16) pre 2D zobrazenie avšak v priestore je to prakticky identické.



Obrázok 16: Ray-Casting

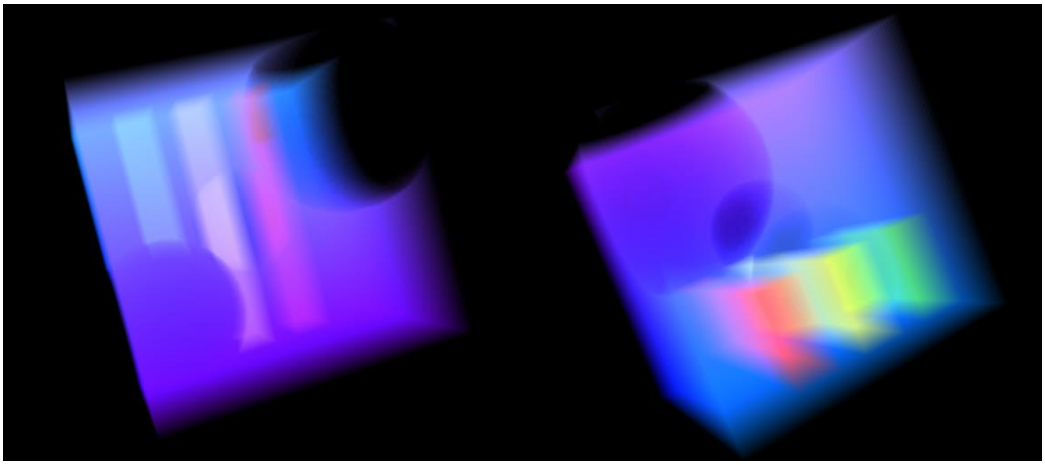
Základný vzorec sa dá zapísať ako $\mathbf{P'_i = \alpha_i * P_i + (1-\alpha_i) * P'_{i-1}}$, kde $\mathbf{\alpha_i}$ určuje priehľadnosť aktuálneho pixlu, $\mathbf{P_i}$ posledný pixel v buffery, $\mathbf{P'_{i-1}}$ pixle zapísane v predošlom buffery a $\mathbf{P'_i}$ aktuálny pixel, ktorý sa zapíše do bufferu.

Existujú dva základné prístupy k vypočítaniu pixla v buffery:

1. Zozadu dopredu
   - Používa sa len základný vzorec, keďže to čo je vzadu sa prekresľuje stále tým čo je v popredí
   - $\mathbf{P'_i = \alpha_i * P_i + (1-\alpha_i) * P'_{i-1}}$
2. Spredu dozadu
   - Vzorec sa rozširuje o zohľadnenie predošlých informácií, ktoré sú vlastne v popredí.
   - $\mathbf{P'_i = \alpha_{i+1} * P_{i+1} + (1-\alpha_{i+1}) * P'_i}$
   - $\mathbf{\alpha'_i = \alpha_{i+1} + (1-\alpha_{i+1}) * \alpha'_i}$

Technológia sa využíva napríklad pre zobrazenie RTG snímkou, kde je podstatné zobrazenie výplne objektu. Pri RTG snímkach sa vezme priemerná farba (intenzita farby) spomedzi všetkých voxlov v lúči.

V našej implementácií seizmických vĺn sme chceli využiť tento spôsob vizualizácie. Vizualizácia mala prebiehať pre viditeľnosť podzemných vĺn cez povrchovú štruktúru aj s vizualizáciou podloží. Príklad podobnej vizualizácie je znázornený na nasledujúcom obrázku (Obrázok 17).



Obrázok 17: Ukážka zahmlenia pomocou Ray-Casting algoritmu

V našej implementácii vizualizácie seizmických vĺn by sme potrebovali nastaviť priehľadnosť farby podľa dĺžky vektora posunutia. Ak sa vektor posunie len o minimum, tak jeho priehľadnosť bude podstatne vyššia než priehľadnosť vektorov, ktoré sa posunuli o väčšiu vzdialenosť oproti pôvodnej pozícii. Týmto postupom sa zvýraznia vysoké vektory posunutia seizmických vĺn. Samozrejme, že len takýto výpočet by nestačil a pre väčšinu zobrazení by sme potrebovali vypočítať aj priemery farieb ako je to napríklad s RTG snímkami.

4. Implementácia

4.1. Hlavný postup

Scéna sa vykresľuje po jednotlivých plochách do hĺbky. Jedná plocha je tvorená podľa vykresľovacieho módu bodmi alebo trojuholníkmi. Každý bod je prepočítaný pomocou aktuálne vybraného farebného spektra. (viac v 3.4 *Farebná škála*) Plochy môžu byť delené pomocou rovinných rezov, t.j. niektoré body sa nevykreslia, ak nespĺňajú podmienky pre vykreslenie pomocou deliacich rovín.

Pri spustení sa v programe nakonfigurujú dve základné triedy Config a Settings s nastaveniami scény. Config je všeobecná trieda, ktorá je počas behu programu z časti nemenná. Sú tu uložené cesty k súborom ako podložie, mapa povrchu a samotná simulácia. Menná časť triedy Config je informácia o rotácií, posunutí a priehľadnosti. V triede Settings sú uložené informácie, ktoré sa týkajú samotnej animácie, ako napríklad rozlíšenie súborov so seizmickou aktivitou, alebo podložia. Obe triedy sú navrhnuté patternom Singleton (viď 5.2 *Singleton*), pretože sú používané všetkými ostatnými triedami ako nastavenia.

4.2. Načítanie scény

Keďže model prostredia a seizmické vlnové pole sú generované pomocou externého programu, tak komunikácia medzi týmito programami je sprostredkovaná pomocou súborového systému. Výpočet seizmických vĺn v jednom frame trvá desiatky minút, prípadne až niekoľko hodín. Práve preto sme sa rozhodli, že grafická vizualizácia bude úplne nezávislá od programu pre fyzické výpočty šírenia seizmických vĺn. Takto sa dá scéna s menšími obmedzeniami prehrávať aj v reálnom čase, čo podstatne zjednodušilo užívateľom prácu s nastaveniami modelu a celkovej scény.

Súbory seizmického vlnového poľa sú tvorené binárnou reprezentáciou[9] štvorbajtových reálnych čísel. Každý bod pre vykreslenie je zložený ako vektor posunutia. Vektory sú reprezentáciou trojrozmerného posunutia v priestore, takže potrebujú pre svoju reprezentáciu tri reálne zložky. Podľa týchto poznatkov sme sa rozhodli, aby každý bod bol v súbore reprezentovaný tromi reálnymi číslami. Jeden súbor však reprezentuje len jedinú rovinu v jednom čase a nie celý frame. Dôvod prečo sme sa rozhodli pre každú platňu

[9] Každý bajt je reprezentovaný ôsmimi bitmi a tento súbor ma väčšinou určený typ, ktorý tieto bajty reprezentujú

vytvoriť osobitný súbor bol veľkosť a následná manipulácia so súbormi. Týmto spôsobom tak môže byť 2D frame reprezentovaný len jedným súborom. Pre reprezentáciu 3D vlnového poľa je potrebný taký počet súborov akú hĺbku výseku seizmických vĺn potrebujeme, resp. máme k dispozícii.

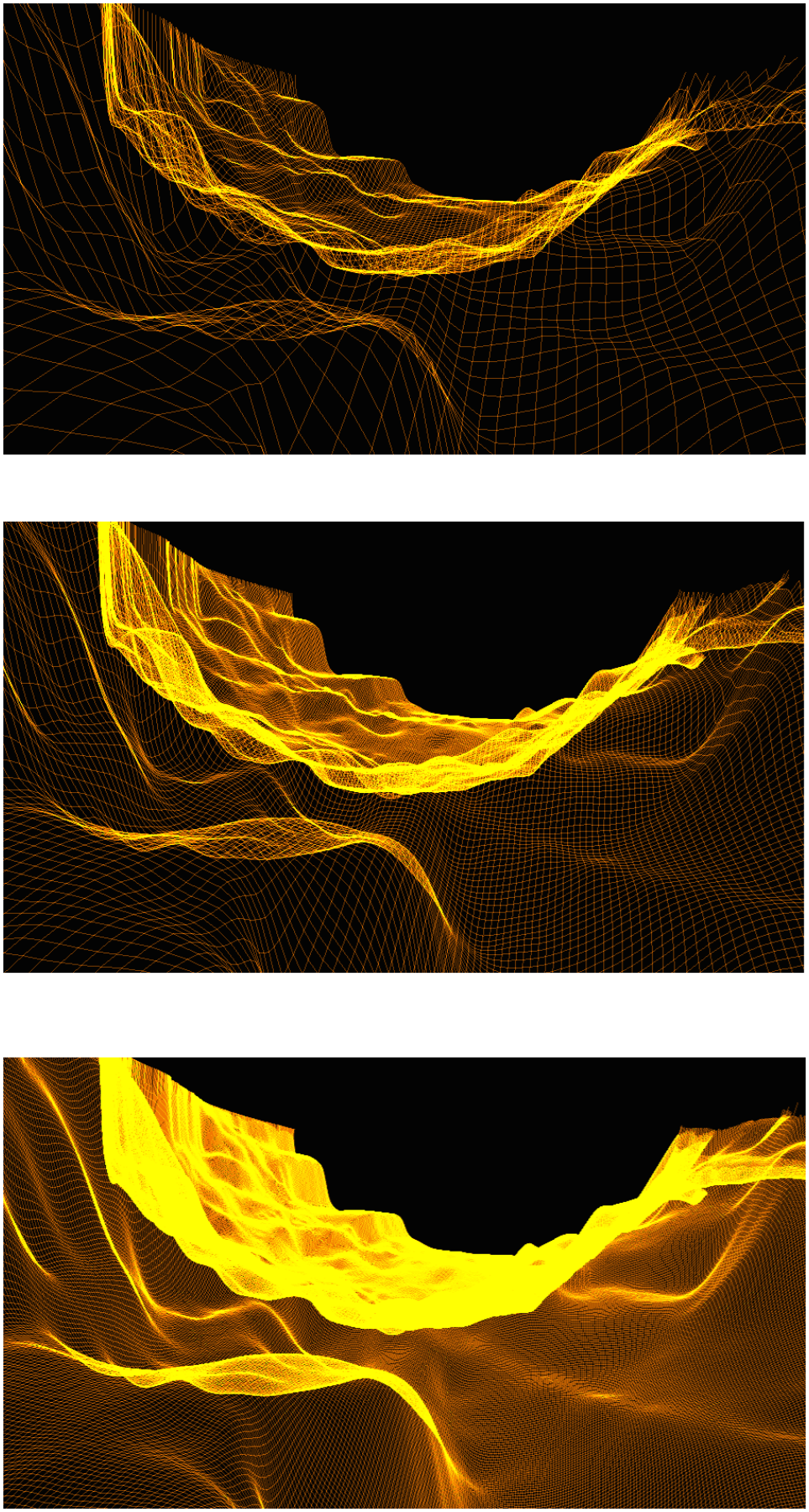
Pre modelovanie seizmického vlnového poľa teda používame súbory jednotlivých hĺbkových rovín, pričom každý bod má svoj vektor posunutia. Pre načítanie každého bodu vektora posunutia potrebujeme zistiť vektorovú dĺžku, aby sme mu vedeli priradiť farbu z určeného farebného spektra. Dĺžku vektora počítame pomocou euklidovských rovníc v priestore: $\mathbf{d_v = sqrt(x^2 + y^2 + z^2)}$. Pred načítaním modelu seizmického vlnového poľa je potrebné, aby užívateľ zadal rozsah dĺžky vektora posunutia. Zadá minimálnu dĺžku vektora, ktorý sa zaráta do vykreslenia určitej farebnej hĺbky. Maximálna dĺžka vektora je takisto určená užívateľom, ale na rozdiel od minimálnej, ak nejaký vektor presiahne túto dĺžku, tak mu bude pridelená maximálna hodnota farebného spektra. Každý vektor načítaný zo súborov seizmického vlnového poľa po načítaní orežeme minimálnou dĺžkou: $\mathbf{d_v = d_v - min}$. Vektor, ktorý je medzi týmito dĺžkami (min. a max) je nakoniec prenásobený hodnotou $\boldsymbol{\alpha}$ nasledovne: $\mathbf{v = d_v * \alpha}$. Pomocou premennej $\boldsymbol{\alpha}$ tak dostaneme rovnomerné rozdelenie dĺžky vektorov počas celého farebného spektra.

Výpočet $\boldsymbol{\alpha}$:

Najprv zistíme dĺžku rozsahu vektora posunutia z minimálnej a maximálnej dĺžky vektora: $\mathbf{d = max - min}$. Rozdiel **d** následne prepočítame na škálu dĺžky 255: $\boldsymbol{\alpha} = \mathbf{255 / d}$.

Ako môžeme vidieť, hodnota $\boldsymbol{\alpha}$ nám určí, ako sa musí každá dĺžka prenásobiť, aby sme dostali z rozsahu **0** až **max-min** rozsah **0** až **255**. Práve pre túto skutočnosť musíme stále od dĺžky vektora odčítať **min**.

Druhým typom vstupných súborov je model podložia. Podložie oddeľujem jednotlivé vrstvy zemských materiálov. Dalo by sa povedať, že v jednotlivých podložiach sú homogénne materiály, avšak na týchto rozhraniach sa hustota materiálu mení. Podľa podloží môžeme sledovať, ktoré rozhranie najviac vplývalo na šírenie seizmických vĺn. Keďže rozhrania sú vykresľované v mriežke, tak nám stačí vedieť hĺbku každého bodu. Načítanie podložia nie je natoľko zložité ako seizmických vĺn. Vizualizácia podložia je jednofarebná, aby sme vedeli jednoducho rozlíšiť jednotlivé vrstvy materiálov. Každé podložie si pamätá aj svoju vygenerovanú farbu, ktorá sa však dá zmeniť užívateľskými nastaveniami. Pre vykreslenie sa každý bod umiestni do správnej hĺbky. Najčastejšie sa vrstvy zobrazujú pomocou mriežky (4.4 *Vykresľovacie módy*), ktorej hustotu si určuje samotný užívateľ (Obrázok 18).



Obrázok 18: Nastavenie hustoty podložia

4.3. Deliace objekty

Deliace roviny sa dajú pridávať pomocou GUI programu. Trieda rovín vie povedať, či sa bod nachádza za, alebo pred rovinou, t.j., či sa má tento bod vykresliť. Ďalšou možnosťou je aj zgrupovanie do kocky, ktorá má 2 základné módy. Prvým módom je vykreslenie všetkého vo vnútri kocky a druhým je vykreslenie všetkého, čo sa nachádza mimo tejto kocky. Deliace roviny však môžu nadobúdať aj tzv. hrúbku, čo znamená, že sa vykreslia body (trojuholníky), ktoré sú od roviny vzdialené len určitú vzdialenosť. Pomocou kocky, ktorá nadobúda túto hrúbku tak dostaneme pekný výsek a viditeľnosť len potrebných stien plného objektu. V tomto móde vysekávania sa však kontroluje každý bod scény či sa nenachádza za nejakou deliacou rovinou, alebo mimo kocky. Každá rovina si pamätá svoje natočenie, pozíciu a hlavne všeobecnú rovnicu, vďaka ktorej vieme jednoducho zistiť, či zadaný bod leží za, alebo pred rovinou pomocou znamienka po dosadení do rovnice. Táto rovnica tak nemusí byť stále prepočítavaná z bodov roviny.

Užívateľ si taktiež môže zvoliť jednoduchší mód, ktorý však podporuje delenie len po X a Y osiach. Jeho činnosť je jednoduchá a špecifická. Užívateľ si vie posúvať začiatok a koniec vykresľovania, ale len po osi X a Y. Pomocou tohto program neprechádza celé platne, ale len výsek ktorý si užívateľ zadá. Zároveň sa nevykresľuje obsah celého výseku, ale len okraje, čo podstatne zrýchľuje renderovanie scény.

Samozrejme aj tieto deliace objekty sa dajú uložiť a použiť opätovne pri iných scénach. Ukladajú sa len základné parametre, ako body a natočenie, prípadne v móde XY delenia sú to len začiatočné a koncové body. V našom programe je na deliace objekty použitý pattern Singleton (5.2 *Singleton*), pretože k týmto objektom musia mať prístup podložia, ale aj samotné seizmické vlny, ktoré sú orezávané.

Deliace objekty sa samozrejme dajú uložiť do textového súboru. Formát súboru je pre človeka čitateľný a ako prvú informáciu zaznamenáva mód, ktorý sa používa pre vyrezávanie scény. Ak sa vyrezáva len pomocou osí X, Y a Z, tak sa zapíšu do súboru ich začiatočné a koncové hodnoty. Ďalšie informácie sa zapisujú už jednotlivé deliace roviny, pričom sa zapamätáva ich natočenie a pozícia. Tieto roviny však môžu súčasne tvoriť aj vyrezávaciu kocku a vtedy sa na jeden riadok musí zapísať 6 týchto rovín. (PRIKLAD)

4.4. Vykresľovacie módy

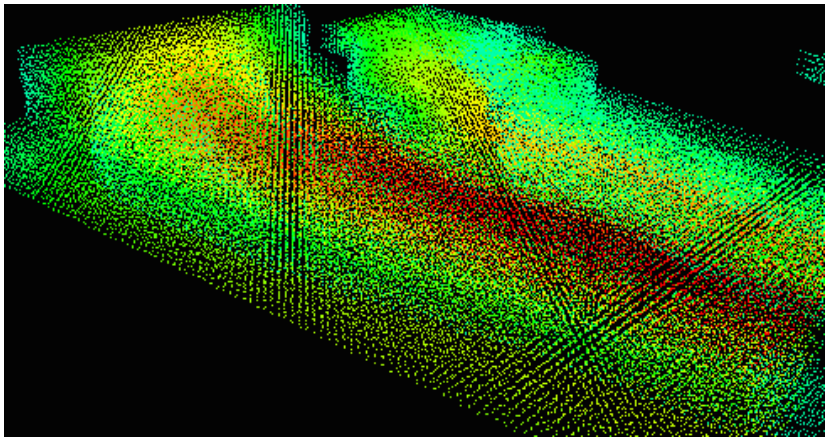
Dáta posunutia sú poskytované vo formáte vektorov posunutia, t.j. troch súradníc (X, Y a Z) v eulerovskom priestore. Každý súbor je reprezentáciou jednej roviny v priestore,

keďže sa bavíme o diskrétnom priestore, tak toto riešenie sa nám zdalo byť najjednoduchšie na implementáciu. Pri načítaní súboru sa vyjadrí dĺžka vektora posunutia, pri ktorej sa zistí jej farebná hĺbka. Najskôr sa zistí, či dĺžka vektora dosahuje užívateľom zadané minimum, ak nie, tak farebná hĺbka bude nastavená na 0. Užívateľ zadáva aj hornú hranicu dĺžky vektora posunutia. Akonáhle je táto hranica prekročená, tak farba bodu bude nastavená na maximálnu možnú hodnotu (255). Farebné spektrum má týmto presnosť 256 rôznych farieb. Ak sú dĺžky vektorov v rozsahu, tak sa farebná veličina prepočíta pomerovo k zadanému rozsahu.

Dáta pre podložie sú načítavané ako súradnice v trojrozmernom priestore. V skutočnosti je to však pravidelne usporiadanie bodov v rámci X a Y, ale hĺbka jednotlivých bodov je kľúčová, vďaka čomu sa formujú samotné krivky podložia. Tieto krivky následne ukazujú kde sa mení homogenita materiálu, čo je pre šírenie seizmických vĺn podstatné. Každé podložie je uložené v osobitnej triede, ktorá uchováva Z-tovú súradnicu (hĺbku) každého bodu a farebnú hĺbku podložia.

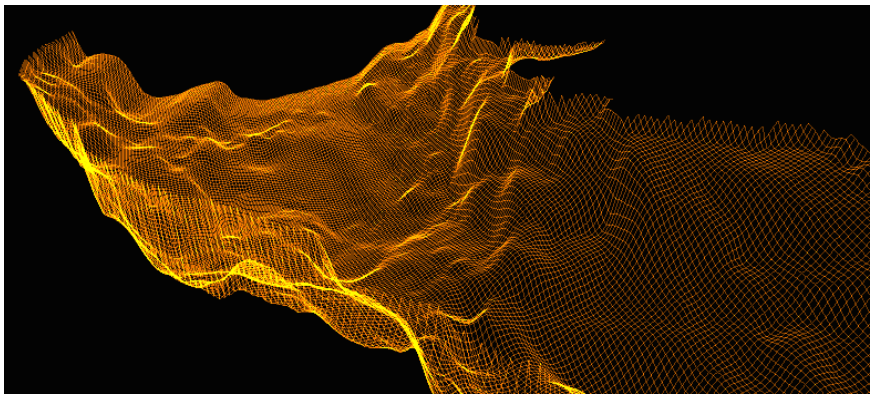
Pre vykreslenie vlnových polí a podložia používame viaceré možnosti zobrazenia. Líšia sa rýchlosťou a plnosťou vykreslenia. Niekedy však potrebujeme, aby sa vykreslili len obrysy, ale inokedy je potrebné plné objekty a tvary.

Prvé a zároveň najjednoduchšie je vykreslenie jedného farebného bodu pre každý vopred vypočítaný bod. Týmto vykreslením scény sme začínali, pretože netreba pozerať na žiadne susedné body. Každý bod sa vykreslí samostatne a farebne sa nemusí dodatočne dopočítavať (vypĺňať) priestor medzi bodmi. Používa sa ak potrebujeme vidieť celé vlnové pole (vrátane vnútorných štruktúr) v jednom frame.



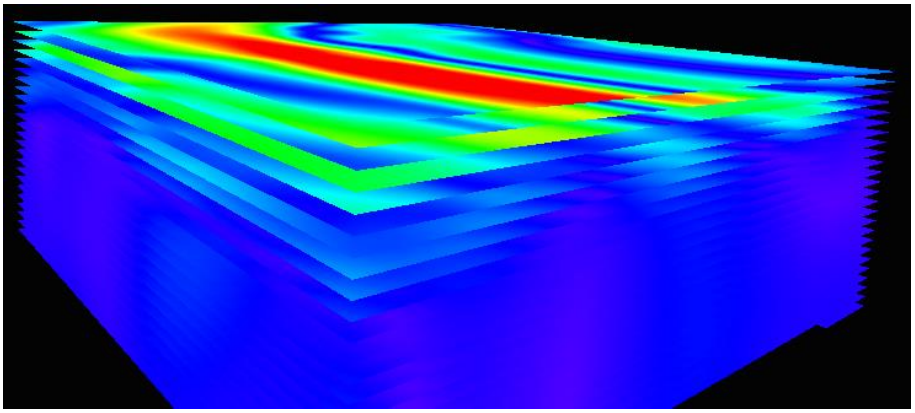
Obrázok 19: Ukážka bodového vykreslenia

Druhým typom vykreslenia je sieťovina. Používa sa len pre podložie a pomáha vidieť krivky modelu podložia, čiže prechodu sedimentov v pomerne dobrých detailoch, ale nie príliš jednofarebne.



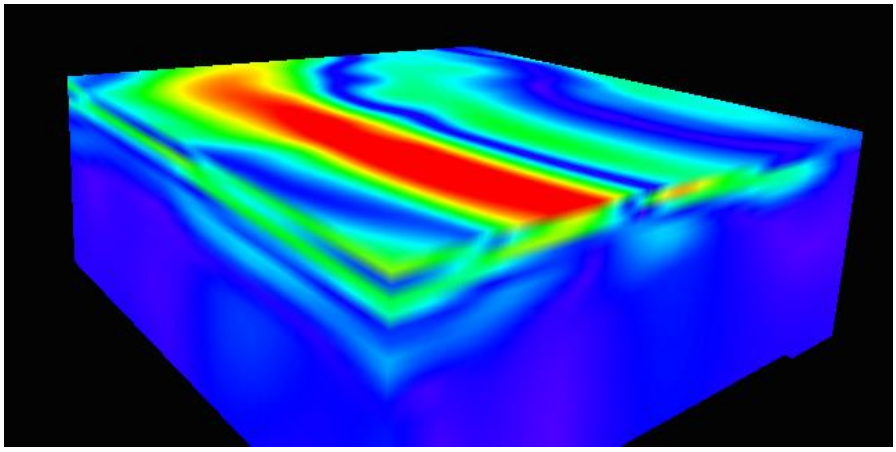
Obrázok 20: Ukážka sieťového vykreslenia

Tretí vykresľovací mód pomáha vypĺňať jednotlivé roviny podložia seizmickými vlnami. Tento spôsob vykreslenia sa používa hlavne pre pohľady na scénu z vrchu, čiže do hĺbky zeme na jej povrch. Často sa k tomuto vykresleniu pridáva aj mapa povrchu, čím vznikne verejnosti známi 2D pohľad pre šírenie vĺn. Jednotlivé body sú spájané pomocou trojuholníkov a plynulé prechody farieb sú tak prepočítavané pomocou základnej knižnice OpenGL, čiže procesorom grafickej karty.



Obrázok 21: Ukážka platňového vykreslenia

Štvrtý spôsobom vykreslenia je plné, čím sa zatiaľ vykresľuje len seizmické vlnové pole. Objekt sa javí ako plný, avšak je taktiež skladaný len pomocou trojuholníkov. Vnútro tohto objektu je niekedy vyplnené kvádrami medzi jednotlivými bodmi, alebo to môže byť len prázdny objekt. Príklad vykreslenia plného módu je znázornený na nasledujúcom obrázku (Obrázok 22).



Obrázok 22: Ukážka objemového vykreslenia

Posledným módom, ktorý sme chceli pridať do práce bol efekt hmly. Vykreslenie by sa malo prepočítavať pomocou Ray-Casting algoritmu (3.5.1 *Volume Ray-Casting*). Vzniknutý efekt mal pripomínať obrázky SCEC. Nakoniec sa nám nepodarilo implementovať tento algoritmus. Jeho jednoduchšie náhrady ako rozpíjanie jednotlivých bodov pomocou OpenGL SPRITE nám nepomohli docieliť tento efekt.

4.5. Výstup

Pre výstup z programu sme sa rozhodli použiť nekomprimovaný rastrový formát BMP. Pre tento formát sme sa rozhodli z dôvodu, že nám bol známy a jednoducho sa s ním pracuje. Formát BMP pre obrázky je zároveň jeden z najpoužívanejších vo svete. Keďže každý pixel je uložený bez kompresie, tak nevzniká žiadna strata dát, resp. kvality výsledného obrazu. BMP formát sa používa od jedno bitového pixlu až po 24 bitový. Jednobitový formát znamená, že si pamätá pre každý pixel len informáciu, či je alebo nie je vyplnený. Takto vznikajú jednoduché čierno-biele obrazy. My sme sa rozhodli využívať 24 bitový formát, ktorý si pre každý pixel zaznamená tri farby (červenú, zelenú a modrú) o hĺbke 8 bitov. 24 bitovým uchovaním sa pre každý pixel môže zachovať až 16,7 mil. farebných odtieňov. BMP si na rozdiel od PNG nepamätá priehľadnosť jednotlivých pixlov, ale to nám v podstate pre náš projekt nebolo potrebné. Nevýhodou BMP formátu je jeho veľkosť. Tým, že sa obrázky nekomprimujú, tak každý pixel zaberá plných 24 bitov, t.j. 3 bajty. Nezdá sa to veľa, ale pri priemerných renderovaných obrazoch rozlíšenie dosahuje hodnotu 10 megapixlov, čo znamená 30 MB na jeden frame. Avšak každý BMP obrázok sa dá bez väčších problémov prerobiť stratovou kompresiou na používaný JPG formát.

4.6. Priehľadné objekty

Priehľadnosť vlnových polí je dôležitá vo viacerých vykresľovaných technikách. Najviac asi pri pohľade z vrchu na Zemský povrch, aby sme videli aj orientačnú mapu podkladu. Druhá taktiež dôležitá vykresľovacia technika s priehľadnosťou je vykreslenie jednotlivých bodov a ukázanie vnútorných vektorov posunu seizmických vĺn.

Prehľadné objekty sa musia usporiadať v rámci scény, aby sa vykreslili v poradí od najvzdialenejšieho po najbližší ku kamere. Dôvod usporiadania je, že ten, ktorý sa aktuálne vykresľuje musí byť farebne počítaný z objektov, ktoré sa nachádzajú za ním. V reálnom živote to funguje podobne, pretože vidíme všetko čo je za oknom, ale nie tak čisto, ako keby tam to okno nebolo. Veci ktoré sa nachádzajú za farebným oknom majú pre pozorovateľa skreslenú farbu. V grafike preto musíme rátať so všetkým čo sa nachádza za prehľadným objektom, aby sme vedeli vypočítať aké farby má mať konkrétne vykresľovaný objekt.

V minulosti sa objekty usporadúvali pomocou maliarovho algoritmu. Tento algoritmus však bol zdĺhavý a bolo nutné stále premiestňovať objekty v stacku. Ďalším problémov bol vznik slučky z troch objektov. Slučka vznikla ak nebol jasné, aké je poradie objektov, pretože každý z troch objektov bol pred nadchádzajúcim, ale zároveň bol za predchádzajúcim. V skutočnosti sa musel jeden objekt rozdeliť na dva, aby bolo jasné, ktorý je v pozadí a ktorý v popredí. Pre dnešné usporiadanie objektov v scéne sa najčastejšie používajú stromy ako Octree (Oktánový strom) a BSP-Tree (Binary Space Partition – binárne rozdeľovanie scény). Octree rozdeľuje priestor na rovnaké diely – 8 kociek (rekurzívne). Priestor sa rozdeľuje, pokiaľ je kocka prázdna, alebo má maximálne určený limit objektov vo svojom priestore, resp. voxle. Octree sa používajú aj pri rozdelení scény pre implementáciu volume Ray-Casting algoritmu, pretože pripravia scénu rozdelenú po voxloch, ktoré sú pre tento algoritmus potrebné. BSP-Tree fungujú na trošku voľnejšom princípe. Z názvu je jasné, že priestor budú rozdeľovať len na 2 časti a nie 8 ako to bolo pri Octree. Scéna je delená pomocou tzv. deliacich rovín, ktoré si do svojho podstromu ukladajú objekty, ktoré sa nachádzajú na tejto rovine (všetko v rovine sa vloží do vrcholu binárneho stromu). Polpriestor, ktorý sa nachádza pred rovinou je rekurzívne uložený ako ľavý syn. Naopak polpriestor, ktorý sa nachádza za touto deliacou rovinou je uložený do pravého podstromu. Postup pre vytvorenie BSP-Tree je rekurzívny, pričom priestor sa nemusí deliť celý, ale len čo má k dispozícii. Vznikne nám takto oveľa menej vrcholov ako pri octree, avšak tento postup je mnohokrát náročnejší na implementáciu. Ak sa stane, že

trojuholník má bod, ktorý leží za deliacou rovinou vrcholu, ale iný bod toho istého trojuholníka leží pre rovinou, je potrebné rozdeliť trojuholník na dva rôzne polygóny. Tieto polygóny môžu byť oba trojuholníky a to ak vrchol deleného trojuholníka ležal v rovine, alebo trojuholník a štvoruholník, ak na jednej strane deliacej roviny boli dva body z trojuholníka.

V našej práci sme použili jednoduchšiu implementáciu BSP stromov. Pomocou tejto teórie určujeme len poradie jednotlivých platní (hĺbka do povrchu Zeme). Postup je jednoduchý a bez delenia polygónov. Vyberie sa stredná platňa znázorňujúca vektor posunutia seizmických vĺn a ak je kamera (pozorovateľ) vyššie, tak sa najprv vykreslí všetko čo sa nachádza hlbšie. Pochopiteľne to čo sa nachádza hlbšie pod Zemským povrchom, je pre nás vzdialenejšie len ak sa nachádzame nad Zemou, resp. stojíme na jej povrchu. Tento postup sa samozrejme opakuje rekurzívne pomocou BSP-Tree teórie, a to tak, že ako prvý vykreslíme vzdialenejší polpriestor.

4.7. Undo-Redo

Undo-Redo je technika používaná vo väčšine programov, povoľuje nám robiť kroky späť, alebo vpred. Požíva sa to hlavne, ak chceme spätne zmazať kroky, ktoré sme nechceli spraviť napr. v grafike pri nesprávnom otočení objektu. V praxi sa technika pre kroky späť uplatňuje nie len v grafike, ale aj nastaveniach prevádzkových prístrojov alebo pri obyčajnom písaní textov.

V princípe sa používajú viaceré postupy (techniky) pre aplikáciu spätných krokov. Používanejší je v princípe program pre uchovávanie inverzných krokov, t.j. trieda si pamätá akciu, ktorá bola vykonaná a pri spätnom kroku spraví presný opak konkrétnej akcie.

Príklad:

- máme objekt1, ktorý užívateľ otočí o 45° po osi X
- program si zapamätá akciu a informáciu o tom ako presne bol objekt rotovaný (objekt1: rot[45, 0, 0])
- užívateľ zadá krok spať a program objekt vráti do pôvodného stavu pomocou inverzného príkazu: objekt1: rot[-45, 0, 0]

Druhým spôsobom undo-redo je zachovanie si celkového stavu, to je, že po každom kroku si program zapamätá všetky objekty a pri kroku späť vráti celú scénu.

Príklad:

- máme objekt1, ktorý užívateľ natočí o 45° po osi X

- program si zapamätá informáciu o všetkých objektoch a ich vlastnostiach (objekt1: rot[0, 0, 0]pos[10, 0, 0]; objekt2:rot[0, 0, 0]pos[0, 10, 10])
- užívateľ zadá krok spať a program objekt vráti do predošlého stavu pomocou uchovaného nastavenia všetkých objektov (objekt1: rot[0, 0, 0]pos[10, 0, 0]; objekt2:rot[0, 0, 0]pos[0, 10, 10]).

Pri inverzných krokoch sa mnohokrát pamätá typ akcie, ktorá sa následne zoskupí do jednej. V jednoduchosti to znamená, že ak píšeme text a určíme krok späť, tak sa nám nevymaže len posledný znak, ale celé slovo, alebo veta.

Tretím spôsobom je hybridné uchovanie informácií, pretože uchovávanie si krokov je pamäťovo menej náročné a zároveň je to mnohokrát rýchlejšie spraviť jeden inverzný krok ako nastaviť všetky objekty na predošlé hodnoty. Avšak nie každý krok môže mať inverzný, napr. pri pridaní prvku do AVL-Stromu sa preskupia vrcholy, no však pri jeho vybratí nebude nutnosť vyvažovať tento strom. Práve pre tieto prípady sa používa uchovanie si stavu v akom sa nachádzala scéna.

V našom programe používame uchovávanie si celého stavu, pretože je to jednoduchšie pre implementovanie. Druhým dôvodom bolo, že si netreba uchovávať až toľko informácií, aby to rázom pohltilo celú pamäť. Uchovávame si len rotáciu a posunutie scény, pretože to sú jediné kroky, ktoré sa stávali nepozornosťou užívateľa. Samozrejme pri ďalšom používaní programu sa uvidí pre aké kroky ešte bude potrebné si uchovávať spätné informácie.

4.8. Osvetlenie scény

Svetlá vo virtuálnej scéne sú dôležité ako samotné objekty. V reálnom živote nám stále cez deň svieti slnko, prípadne miestnosti máme osvetlené umelým svetlom. Ak by sme nemali svetlo vo virtuálnom prostredí, takisto by sme nevideli objekty, resp. by sme ich videli len skreslene.

V našom programe sme umožnili užívateľom prácu s osvetlením. Na výber má užívateľ základne osvetlenie, ktoré ponúka OpenGL, kde sa používa len okolitý (ambient) svetelný zdroj. Pri takom osvetlení sa v objekte neukážu rôzne preliačiny a výbežky. Pracuje na princípe, že svetlo sa nachádza po celej scéne v rovnakom množstve a neráta sa ani odrazený svetelný lúč. Pridaním svetelného zdroja, ktorý má v sebe okrem tejto všeobecnej svetelnej zložky aj priamu zložku. Podrobnejšie sú svetlá a svetelné zdroje prebraté v kapitole 5.3.1 *Svetlá v OpenGL*. Užívateľ si môže okrem druhu a pozície svetla nastaviť aj jeho intenzitu, prípadne farbu. Pre nastaviteľnú intenzitu svetelného zdroja sme sa

rozhodli podľa potrieb užívateľov, pretože v rôznych podmienkach scény (bez, resp. s podložím, alebo zemského povrchu) je nutnosť osvetlenia a jeho intenzita rôzna pre najlepší výsledný efekt.

4.9. Triedy programu

Štruktúra tried v programe sa triedi na dve základné skupiny:

1. skupina s prefixom DP_
   - označuje triedy na výpočtovom pozadí programu (Back-End)
2. skupina bez prefixu
   - označuje triedy, ktoré sa starajú o GUI[10] (Front-End)

DP_Settings a DP_Config:

- Ukladajú nastavenia programu ako je natočenie, hustota bodov
- Tieto triedy sa na začiatku konfigurujú pomocou súboru priloženom pri programe **config.cfg**, alebo sa dajú užívateľom načítať pod iným názvom
- Sú tvorené pomocou Singleton Patternu (5.2 *Singleton*)

DP_DivObjects, DP_DivObject a DP_DivPlane:

- Do týchto tried si program ukladá deliace objekty (4.3 *Deliace objekty*)
- Trieda DP_DivPlane si pamätá jednu deliacu rovinu, ktoré môžu byť spojené do deliacej kocky v triede DP_DivObject. Trieda DP_DivObjects si ukladá všetky deliace objekty a informáciu o deliacom móde.
- Sú tvorené pomocou Singleton Patternu (5.2 *Singleton*)

DP_SeismicSpace, DP_Space a DP_Plane:

- V triedach sú uložené informácie o seizmických vlnách (4.2 *Načítanie scény*)
- Trieda DP_Plane si pamätá jednu platňu seizmických vĺn. Tieto platne sa skladajú do trojrozmerných frame pomocou triedy DP_Space. Frame sú vykresľované v správnom poradí pomocou triedy DP_SeismicSpace

DP_SeismicStations:

- Trieda je určené na načítanie a vykreslenie seizmických staníc a epicentra zemetrasenia
- V programe môže byť vykreslené len jedno epicentrum zemetrasenia, pričom seizmických staníc je viacero.
- Načítanie zo súboru:

[10] Graphical User Interface (grafické užívateľské rozhranie) – umožňuje užívateľovi ovládanie programu

- Súbor obsahuje informácie, ktoré sú čitateľné pre užívateľa len za pomoci textového editora
- Epicentrum je zadané s úvodným znakom **E** a nasledujú súradnice v priestore pomocou osi X, Y a Z a rotácia okolo týchto troch osí (Napr. E 625 530 -15 45 0 0 – vieme, že sa jedná o epicentrum v hĺbke 15 jednotiek s natočením 45 stupňov okolo osi X)
- Ďalšou informáciou v súbore sú seizmické stanice, ktoré začínajú znakom **S** po ktorom nasledujú informácie o umiestnení stanice v súradniciach X, Y a Z podobne ako pri epicentre. Ďalšou informáciou je farba seizmickej stanice, pretože niektoré sú virtuálne a najlepším znakom ako rozlíšiť virtuálne od skutočných bola farba.

- Vykreslenie:
  - Epicentrum sa vykresľuje ako guľa s textúrou, ktorá má 4 čierno-biele pásiky čím dosiahneme efekt beach ball, ktorý sa používa v seizmológii pre zobrazenie epicentra zemetrasenia.
  - Seizmické stanice sa vykresľujú pomocou kvadratického ihlanu[11] s určenou farbou.

DP_OpenGLLight:

- Trieda je určené pre nastavenie svetiel v programe (4.8 *Osvetlenie scény*). GUI pre túto triedu je určené v LightSettings.

DP_ModelSurface a DP_Model:

- Tieto triedy sa starajú o podložia (4.2 *Načítanie scény*)
- Trieda DP_Model obsahuje jedno podložie, jeho hĺbku a farbu. Trieda DP_ModelSurface uchováva jednotlivé podložia pomocou vektora.
- DP_ModelSurface je tvorená pomocou Singleton Patternu (5.2 *Singleton*)

DP_ArcBall a DP_Quaternion:

- Tieto triedy sa starajú o správne otáčanie scény a jednotlivých objektov ako aj deliacich rovín a kociek (3.2 *ArcBall*)

DP_UndoRedo:

- Trieda je určená pre uchovávanie transformácií (4.7 *Undo-Redo*)

Zvýšné triedy sú len pomocné, napr. pre prácu s maticami alebo počítanie FPS.

[11] Ihlan so štvoruholníkovou podstavou

5. Technológie

V tejto kapitole predstavíme bližšie vlastnú implementáciu, resp. technológie programu pre vykreslenie seizmických vlnových polí.

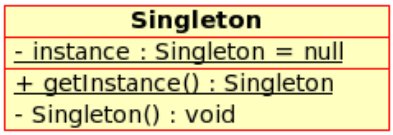
5.1. C++ a Qt

Prvá otázka asi bude, prečo sme si vybrali práve C++ za programovací jazyk. Hlavnou výhodou tohto jazyka je, že sa všetko kompiluje do strojového kódu a preto je program väčšinou rýchlejší. Java je síce jednoduchší programovací jazyk, ktorý používa komunikáciu pomocou smerníkov, avšak beží len vo virtual machine prostredí, čím je mnohokrát pomalšia od C++. Java má však výhodu v multiplatformovom prístupe, avšak to sa vyriešilo otvoreným zdrojovým kódom, ktorý sa týmto dá skompilovať na rôznych platformách. Zo štatistiky vyplýva, že asi 95% engine-ov počítačových hier využíva C++. Java sa prevažne používa pre internetové, alebo mobilné aplikácie. Náš projekt sa skôr dá porovnať s engine-om počítačových simulácií a hier, tak sme sa práve aj pre túto štatistiku väčšiny rozhodli vyžiť programovací jazyk C++.

Kvôli tomu, aby program bol spustiteľný na rôznych operačných systémoch (prevažne Linux), tak sme sa rozhodli použiť QtCreator, ktorý má podporu v základných OS, ako Linux, Windows a Macintosh. Na začiatku sme však používali Microsoft Visual Studio 2010, pretože sme nepoznali lepší vývojový nástroj pre programovací jazyk C++. Predošlá práca pre vizualizáciu seizmických vĺn na Slovensku, ktorú robil pán Racko, bola takisto vyrábaná na Microsoft Visual Studio 2010 a samozrejme sme boli trošku skeptický voči QtCreator. Nakoniec sme však prešli k tomuto frameworku, keďže to muselo byť spustiteľné na serveri s operačným systémom Linux.

5.2. Singleton

Je to návrhový vzor (design pattern), ktorý používa statickú inštanciu vlastnej triedy. Použitie patternu sa oplatí hlavne ak potrebujeme mať v programe jednu inštanciu triedy pre všetky ostatné objekty. Často sa využíva pre triedy, ktoré v sebe nesú globálne nastavenia programu. Trieda má pre iné objekty skrytý konštruktor a tak ju nevie nikto vytvoriť okrem nej samotnej. Ak si ju chceme zavolať resp. vytvoriť, alebo potrebujeme nastaviť jej premenné, tak je nutné zavolať jej vlastnú funkciu, ktorá nám vráti inštanciu tejto triedy. Keďže trieda má vlastnú inštanciu statickú, tak sa správa podobne ako globálna trieda s globálnymi premennými.



Obrázok 23: Singleton pomocou UML diagramu

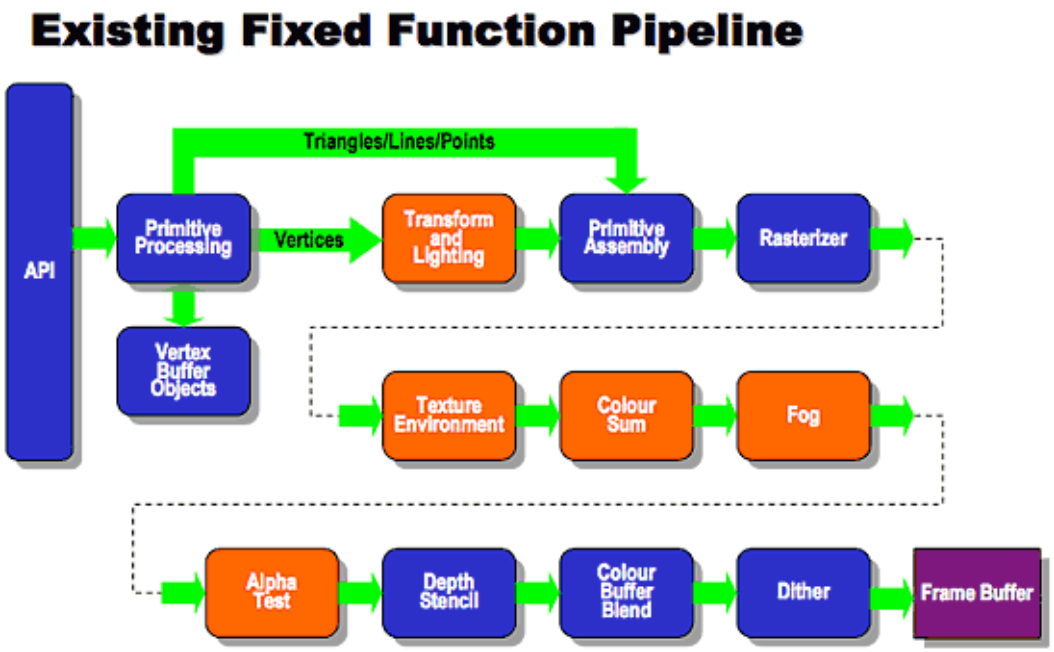
Ako môžeme vidieť na obrázku (Obrázok 23) trieda Singleton začína statickou inštanciou, ktorá je prázdna. Pri prvom volaní funkcie getInstance() sa inicializuje premenná instance pomocou konštruktora (ktorý je skrytý pre ostatné triedy). Funkcia getInstance() zistí, či už bola, alebo ešte nebola inicializovaná daná trieda pomocou porovnania premennej instance s hodnotou null. Ak už bola inicializovaná, tak len vráti už vytvorenú statickú inštanciu triedy Singleton. Tento postup sa používa hlavne v silno objektových programovacích jazykoch ako je Java, keďže globálne premenné by sa tam robili zložitejšie, tak sa používa statická (globálna) trieda. My sme sa rozhodli využívať tento pattern, pretože nám zjednodušil prácu s globálnymi premennými a ich načítaním z konfiguračných súborov.

5.3. OpenGL

OpenGL je grafický štandard, ktorý sa inštaluje na pevno k základným grafickým kartám od roku 1992. Za týmto projektom sa podieľali Mark Segal a Kurt Akeley, avšak bez podpory veľkých a známych firiem ako Apple, SGI, Dell, IBM, či Intel, nVidia, ATI a mnoho ďalších, ktoré stáli za presadením grafického štandardu by tento projekt upadol. Na začiatku projektu OpenGL stála aj spoločnosť Microsoft, ktorá však po roku odstúpila od zmluvy, aby si vyvinula svoj vlastný štandard pridávaný do grafických kariet Microsoft ® DirectX ®. Tento štandard od firmy Microsoft sa však presadil až v roku 1995, čiže až 3 roky po knižnici OpenGL.

Platformu OpenGL sme si vybrali z viacerých dôvodov. Prvý a základný dôvod bol, aby to bolo spustiteľné pod OS Linux. Platforma Microsoft ® DirectX ® nie je podporovaná OS Linux, tak určite nebola vhodná pre náš projekt. Druhým dôvodom bola cena, keďže knižnica OpenGL je bezplatná, tak prevládala aj v tomto nad knižnicou Microsoft ® DirectX ®.

Zobrazovací kanál OpenGL sa skladá z postupného vytvárania scény ako je znázornené na obrázku (Obrázok 24).



Obrázok 24: Zobrazovací kanál v OpenGL [3]

Postup zobrazovacieho kanálu:

1. Načítanie vrcholov v špecifickom poradí
2. Spracovanie vrcholov pomocou vrcholového shadera
3. Vyváranie primitívnych tvarov pomocou geometrického shadera
4. Orezávanie primitívnych tvarov a ich prípadne odstraňovanie
5. Rasterizácia primitív na fragmenty
6. Spracovanie fragmentov pomocou fragmentového shadera
7. Zmiešanie priehľadnosti, hĺbky a ostatných šablónových operácií

Spracovanie vrcholov:

Vrcholový shader vyťahuje vrcholy zo zdroja dát a následne ich spracováva. Funguje tak, že prerába vytiahnuté vrcholy na vrcholy jednoduchšieho charakteru. Tieto vrcholy nazývame výsledné a budú založené na ľubovoľnom programe. Každý vstupný vrchol nesie so sebou vstupné informácie. Výstupné vrcholové informácie sú na rozdiel od tých vstupných obohatené o ďalšie požiadavky. Jednou z dôležitých informácií, ktorú nesie výstupný vrchol, je pozícia. Je vypočítaná pomocou vrcholového shadera. Pozícia je dôležitá pretože z nej získavame informácie o ďalších krokoch.

V procese spracovania vrcholov pomocou vrcholového shadera existuje základné obmedzenie. Toto obmedzenie hovorí, že každý výsledný vrchol musí zmapovať

predchádzajúci vrchol. Vstupné a výstupné hodnoty sa mapujú v pomere 1:1, pretože vrcholový shader nemôže zdielať stav medzi týmito hodnotami. Môže nastať situácia, že dostaneme rovnaké výsledné dáta. V tom prípade do vrcholové shadera boli zadané rovnaké hodnoty v rovnakých primitívnych tvaroch. Takto sa optimalizuje proces spracovania vrcholov. Dáta sa uskladňujú v takzvanej post-transformálnej pamäti, a preto nie je nutné opakovať proces znovu spracovania vrcholov.

Sústredenie primitív:

Do primitívnych tvarov boli zozbierané a uložené výsledné dáta, ktoré boli vypočítané z vrcholového shadera. Postupy v tomto kroku sa líšia. Pomerne dobrý príklad je, ak by na vstupe bolo 12 vrcholov a primitívny tvar bol nadstavený na trojuholník. Potom výstup by mohol byť aj 10 rôznych trojuholníkov.

Geometrický shader (GS):

Je to program, ktorý spracúva každý vstupný tvar a je definovaný používateľom. Výsledkom tohto shadera môžu byť viac východiskové alebo až nulové primitívne tvary. Z predošlého kroku sú definované vstupné tvary. Geometrický shader pomáha vrcholovému shaderu. Dokáže konvertovať rôzne primitíva. Vie napríklad dostať zo základných primitívnych tvarov, trojuholník, alebo z priamky iba body a podobne.

Transformácia spätnej väzby:

OpenGL má špeciálny mód nazývaný „transform feedback“. Umožní nám skončenie celého zobrazovacieho cyklu aj v tomto bode. Je to možné lebo geometrický shader už dokáže vytvárať plnohodnotné tvary, preto už nemá zmysel pokračovať v procese pipeline.

Vystrihovanie a odstraňovanie:

Tento krok sa využíva iba v tom prípade. ak k vykresleniu všetkých tvarov nestačil predchádzajúci proces. Strihanie je rozdelenie na viacero primitív. Robí sa v tom prípade, ak sú medzi vonkajší a vnútorným objemom primitívne tvary. Vystrihujú sa iba tie primitíva, ktoré nie sú všetky vo vnútri prevádzkovej oblasti. Primitíva z vnútra prevádzkovej oblasti sa môžu vykresliť celé.

Rasterizácia:

Po dosiahnutí štádia rasterizácie sú primitívne tvary rozdelené do menších fragmentov. Na výpočet výsledného pixlu potrebujeme jeden fragment čo je množina vrstiev. Informácie o umiestnení na obrazovke a príklade pokrytia nesie každá jedna vrstva. Pomocou geometrického shadera je počítaný výsledný súbor dát. Vypočíta sa interpoláciou medzi hodnotami dát.

Spracovanie fragmentov:

Po rozbití primitív na fragmenty sa pridajú farby, farebné hĺbky a farebné odtiene pre každý jeden segment.

Per-sample operácie:

Konečnú hodnotu obrazu vytvorí postupnosť viacerých krokov za sebou. Táto operácia sa nazýva per-sample. Do výsledného framebufferu nebudú pridané fragmenty, ktoré sa odstránia po otestovaní pomocou šablónového testu. Skorý test hĺbky dokáže nahradiť fragmetový shader v prípade, ak fragmentový shader nezaznamená hĺbku fragmentov.

Nasleduje sprehľadnenie jednotlivých objektov, je to možné iba ak už prebehli odstránenia. Z hodnôt framebuffera a aktuálneho vykresľovaného fragmentu sa vypočíta hodnota farieb. Následne sú do framebuffera zapíšu nové fragmentové dáta. Pre tento krok je veľmi dôležité správne poradie priehľadných trojuholníkov pretože vo framebufferi musí byť všetko čo je za aktuálnym vykresľovaním priehľadným polygónom.

5.3.1. Svetlá v OpenGL

Svetlá sú pre vizualizáciu objektov veľmi podstatné. V reálnom živote taktiež bez osvetlenia nevieme rozoznať objekty a ich farby. Aby sme mohli vidieť objekt, musí byť osvetlený. Od kvality a intenzity osvetlenia závisí aj kvalita viditeľnosti objektu. Avšak každý objekt sa môže správať ináč aj pri rovnakom svetle. Objektom sa v OpenGL navrhujú rôzne materiály, prípadne či vôbec bude reagovať na určitú svetelnú zložku.

OpenGL nám ponúka tri abstraktné typy osvetlenia:

1. Ambient light (okolité osvetlenie)
2. Diffuse light (rozptýlené osvetlenie)
3. Specular light (zrkadlové osvetlenie)

Ambient light:

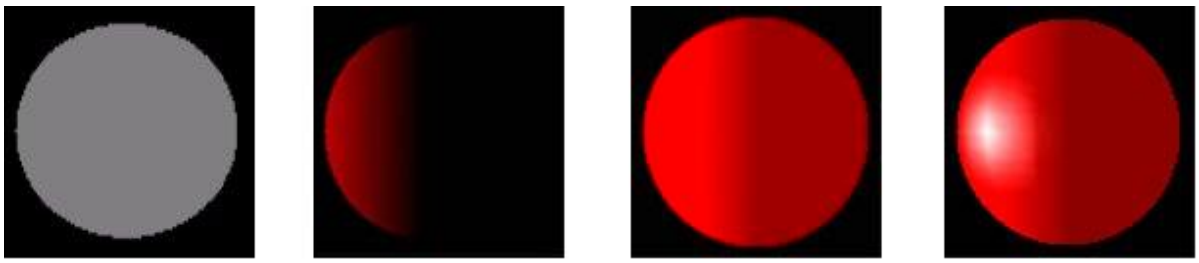
Okolité osvetlenie sa správa ako všeobecné svetlo, ktoré raz z vyšlo z konkrétneho bodu, ale mnohokrát sa už poodrážalo a tak sa zdá, ako keby prichádzalo z viacerých svetelných zdrojov zároveň. Všetky časti objektu sú takto osvetlené s rovnakou intenzitou. Ako môžeme vidieť na nasledujúcom obrázku (Obrázok 25 v ľavo), tak 3D objekt ako je guľa je viditeľná len ako kruh, čiže sa nám javí len ako 2D objekt. Pri jednotvárnom objekte to veľmi nie je vidno, ale pri objekte, ktorý má výbežky má takéto svetlo nežiaduci účinok. Mnoho detailov ako sú aj tiene alebo doliny v podloží takto zanikajú.

Diffuse light:

Rozptýlené svetlo vyráža z jedného bodu a tým osvetľuje privrátenú časť objektu viac ako jej odvrátenú stranu. Funguje to tak, že objekt je schopný časť tohto svetla pohltiť a tým sa stáva, že sa na časti v scéne ani nedostane. Ako môžeme vidieť na ilustračnom obrázku (Obrázok 25 druhý z ľava), objekt vyzerá, ako keby bol osvetlený len z ľavej strany. Spojením okolitého a rozptýleného (ambient - diffuse) svetla dostávame realistickú scénu a viditeľnosť objektov (Obrázok 25 druhý z prava).

Specular light:

Posledným, ale nezanedbateľným typom svetelného zdroju v OpenGL je zrkadlové svetlo. Ak má objekt lesklý materiál, tak tento druh svetelného zdroja sa bude odrážať od jeho privrátených stien viac. Pre lepšie pochopenie je znázornený príklad na nasledujúcom obrázku (Obrázok 25 v pravo).



Obrázok 25: Ambient light - Diffuse light – AmbDiff light – Specular light [5]

Samotné nastavenie svetiel v OpenGL však nestačí, pretože pre správne vykreslenie objektov je potrebné nastaviť aj materiály samotných objektov. Materiál môže nadobúdať rôzne hodnoty, ako je lesklosť resp. matnosť, farbu prípadne textúru. OpenGL API ponúka rôzne nastavenie pre prednú aj zadnú stenu. Vrcholy polygónov sa zadávajú väčšinou po

smere hodinových ručičiek, pretože vtedy dostaneme prednú stranu na určené miesto bez ďalších nastavení. Normála polygónov sa však dá prestaviť, ale vykreslenie vrcholov v správnom poradí je určite menej náročnejšie.

Knižnica OpenGL ponúka jednoduchú funkciu pre nastavenie typu materiálu, ktorý sa bude používať pre polygóny s názvom glMaterialX. Sufix X je určený pre vstupné dáta, pretože materiál môže byť nastavený pomocou reálnych, alebo celých čísel, prípadne len pomocou nastaveného poľa čísel. Prvý parameter špecifikuje ktorej strany materiálu sa nastavenia týkajú (predná: GL_FRONT, zadná: GL_BACK, predná aj zadná zároveň: GL_FRONT_AND_BACK). Druhý parameter hovorí na ktoré svetlo bude daný materiál reagovať (GL_AMBIENT, GL_DIFFUSE, a pod.). Posledným parametrom je pole čísel, ktoré udáva farbu materiálu v štyroch zložkách (RGBA), alebo len jedno číslo, ktoré sa však používa hlavne pri nastavovaní lesklosti materiálu (GL_SPECULAR).

5.3.2. Priehľadnosť v OpenGL

V počítačovej grafike sa pre priehľadnosť objektov využíva tzv. alfa kanál. Alfa kanál je braný ako štvrtá farebná hĺbka pixlu, alebo voxlu. Základná hodnota kanálu je dvojbitová, t.j. len oznam o tom, či je pixel priehľadný, alebo nie. V dnešných grafických kartách sa však používa už 32 bitový farebný systém, takže plných 8 bitov (2^8 kombinácií) je rezervovaných pre alfa kanál.

V knižnici OpenGL si môžeme vybrať medzi viacerými možnosťami počítania farieb v priehľadnosti. Základných funkciou počítania priehľadnosti je **glBlendFunc(src, dst)**. Parameter **src** určuje ako sa počíta farba, ktorá sa ide vykresliť, napr. **GL_SRC_ALPHA** znamená, že farba, ktorá sa ide vykresliť sa vykreslí po prenásobení alfa kanálom. Druhým parametrom **dst** sa nastavuje ako sa bude zohľadňovať predchádzajúca farba v buffery. Ak sa vypne Z-Buffer, tak nie je nutnosť usporiadania scény, ale celá scéna bude musieť byť priehľadná. Väčšinou je vypnutie Z-bufferu však len na ukážky priehľadnosti a v skutočnosti sa používajú algoritmy ktoré usporiadajú objekty do správneho poradia (viď kapitolu 4.6 *Priehľadné objekty*). V našom programe používame priehľadnosť, kde sa aktuálna farba násobí alfa kanálom a predošlé farby v buffery majú alfa kanál odčítaný od jednej: glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA).

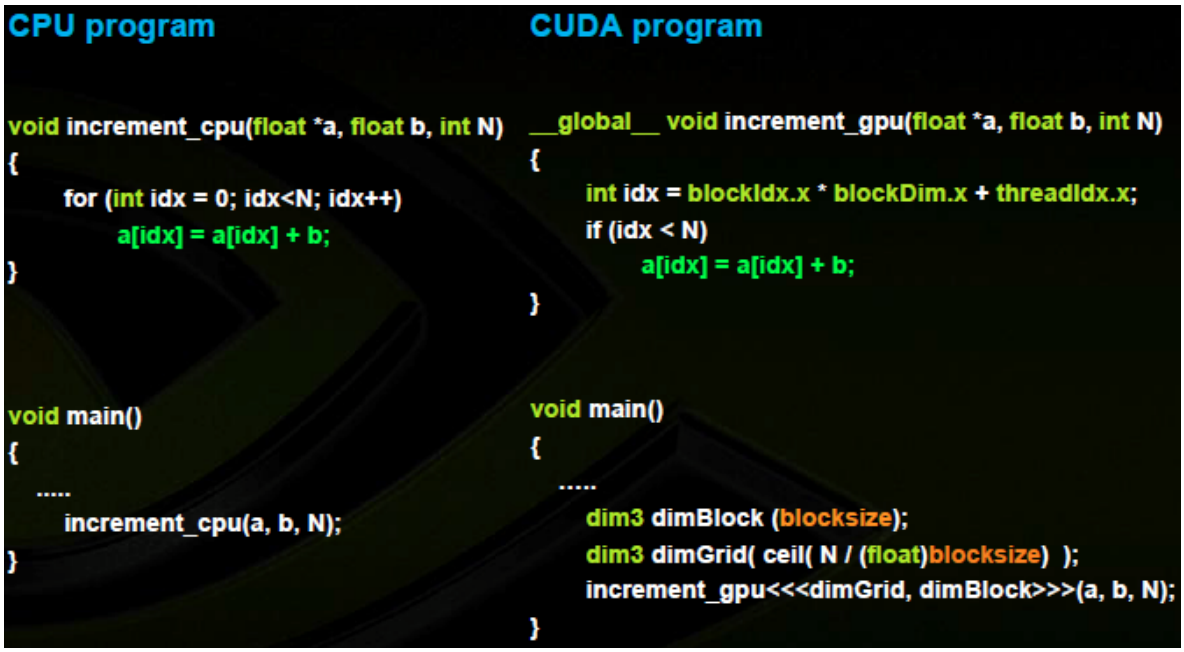
5.4. Cuda

Keďže náš program je prevažne grafický a vizuálny, tak je určite najlepšie čím viac nechať výpočty na GPU (Graphics processing unit). Práve pre túto skutočnosť by sme

časom chceli prejsť aj na túto novú technológiu, ktorú však podporuje len NVIDIA. Systém Cuda veľmi dobre napodobňuje viacjadrové procesory, t.j. dokáže pracovať paralelne. Tieto procesory obsahujú fyzicky niekoľko vlákien, no aj virtuálne vlákna, pre ľahšiu spoluprácu výpočtov. Tieto thready sú spájané do blokov (mriežok). Bloky majú výhodu, že sa nemusia synchronizovať, pretože v podstate už sú synchrónne. Grafické karty vybavené Cuda sú schopné počítať rýchlosťou 500 GFlops.

Dôvod, prečo sme v našej práci nepoužili túto novinku je, že je to príliš neuniverzálny nástroj, keďže server na ktorom sa pracuje využíva grafickú kartu od spoločnosti ATI Radeon. Avšak ani staršie NVIDIA grafické karty nepodporujú túto novinku. V budúcnosti by sme sa však chceli venovať aj prechodu systému pre výkonnejšie spracovanie, čo by znamenalo aj prepis určitých operácii do Cuda systému.

Programovací jazyk pre Cuda je veľmi podobný C/C++, keďže spoločnosť sa snažila, aby si programátori nemuseli zvykať na nič nové, ale na niečo čo je staré a osvedčené. Ako príklad som vybral ukážku od spoločnosti NVIDIA pre porovnanie s C++ (Obrázok 26).



Obrázok 26: Ukážka Cuda systému [11]

Ako môžete vidieť, tak cykly, v ktorých nezáleží na poradí sa dajú počítať paralelne, čo z lineárnej zložitosti O(n) spraví konštantnú zložitosť O(1).

Vývojári mysleli dokonca aj základné matematické funkcie ako mocnina (powf), odmocnina (sqrt) alebo sínus (sin). Spoločnosti NVIDIA išlo hlavne o minimalizovanie

prechodov medzi CPU a GPU a tak z toho spravili plnohodnotný paralelný programovací jazyk na grafickej karte.

6. Výsledky

Finálny program splnil očakávania seizmológov. Užívateľ vie paralelne spustiť program pre simuláciu seizmických vĺn a náš program pre vizualizáciu týchto vĺn. Náš program pracuje na princípe, že ak nemá potrebné súbory, tak sa na chvíľu uspí, čím nezaťažuje výpočtovú kapacitu počítača. Po určenom čase sa tento program zobudí pomocou trigera a zistí, či sú súbory pripravené na načítanie. Ak sa tieto súbory stále nenachádzajú na určenom mieste, tak sa spánok behu programu opakuje až do chvíle, keď sa zobrazia vypočítané dáta v určenom priečinku. Vďaka vykresľovaniu objektov pomocou knižnice OpenGL nie je potreba príliš zaťažovať procesor a tak ho náš program prevažne prenecháva programu pre výpočet šírenia seizmických vĺn.

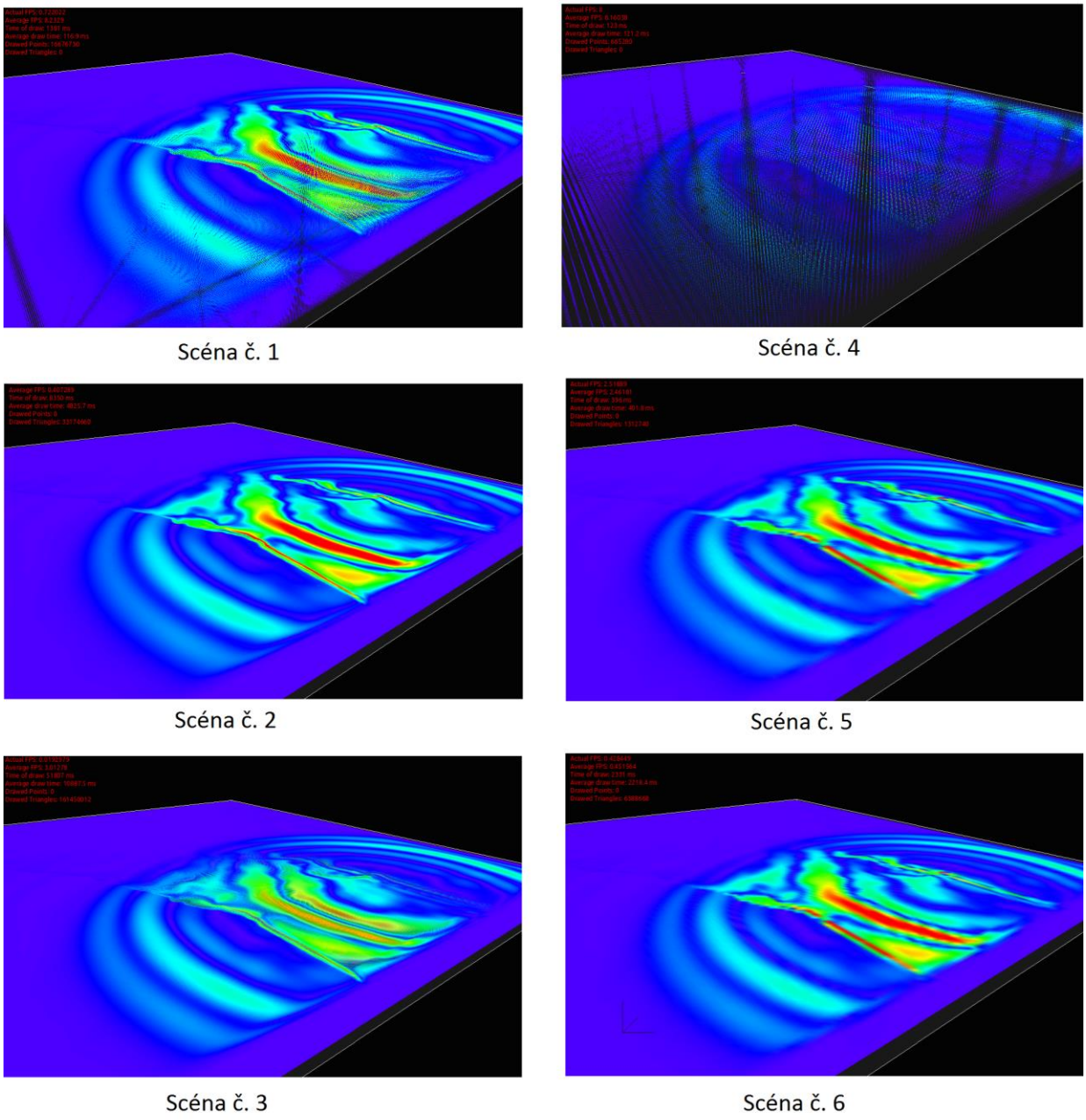
Množstvo dát pre náš program sa pre jeden obraz pohybuje okolo 500 mil. bodov. Každý bod je v súbore zapísaný ako trojrozmerný vektor posunutia uložený pomocou štvorbajtového desatinného čísla (float). Týmto spôsobom je jediný bod zapísaný pomocou 12 bajtov. V našom programe si namiesto celej polohy a smeru vektoru pamätáme len podstatnú dĺžku, ktorú premietneme na 8 bitové číslo, čo je jeden bajt. Týmto zmenšením sme dosiahli možnosť zapísať 12x viac informácií do RAM. Avšak na takéto zmenšenie musíme pri každom vykreslení vypočítať farbu bodu (pixlu), čo nám zároveň umožňuje meniť farebné škály aj počas behu programu.

Vďaka nastaveniam intenzity jednotlivých bodov a módu vykreslenia si môžeme určiť kvalitu, ale aj rýchlosť renderovania.

Tabuľka 3: Výsledne dáta – plné vykreslenie

Scéna	Mód vykreslenia	Hustota bodov	Počet vykreslených		Čas vykreslenia v ms	FPS
			Bodov	Trojuholníkov		
1	Body	1	16 676 730	0	1 381	0,72
2	Platne	1	0	33 174 660	8 350	0,11
3	Objem	1	0	161 450 012	51 807	0,02
4	Body	5	665 280	0	121	8,16
5	Platne	5	0	1 312 740	401	2,46
6	Objem	5	0	6 388 668	2218	0,42

V tabuľke č. 3 vidíme závislosť času vykreslenia na základe zvoleného módu vykreslenia a hustoty bodov (scény). Na nasledujúcom obrázku (Obrázok 27) sú znázornené scény podľa tabuľky výsledkov (Tabuľka 3). Prvý stĺpec obrázkov znázorňuje hustotu vykreslenia, či sa jedná o vykreslenie každého bodu, alebo každého piateho. Prvý riadok obrázkov znázorňuje vykreslenie scény pomocou bodov. Druhý riadok znázorňuje vykreslenie pomocou platní a posledný tretí riadok znázorňuje objemové vykreslenie.

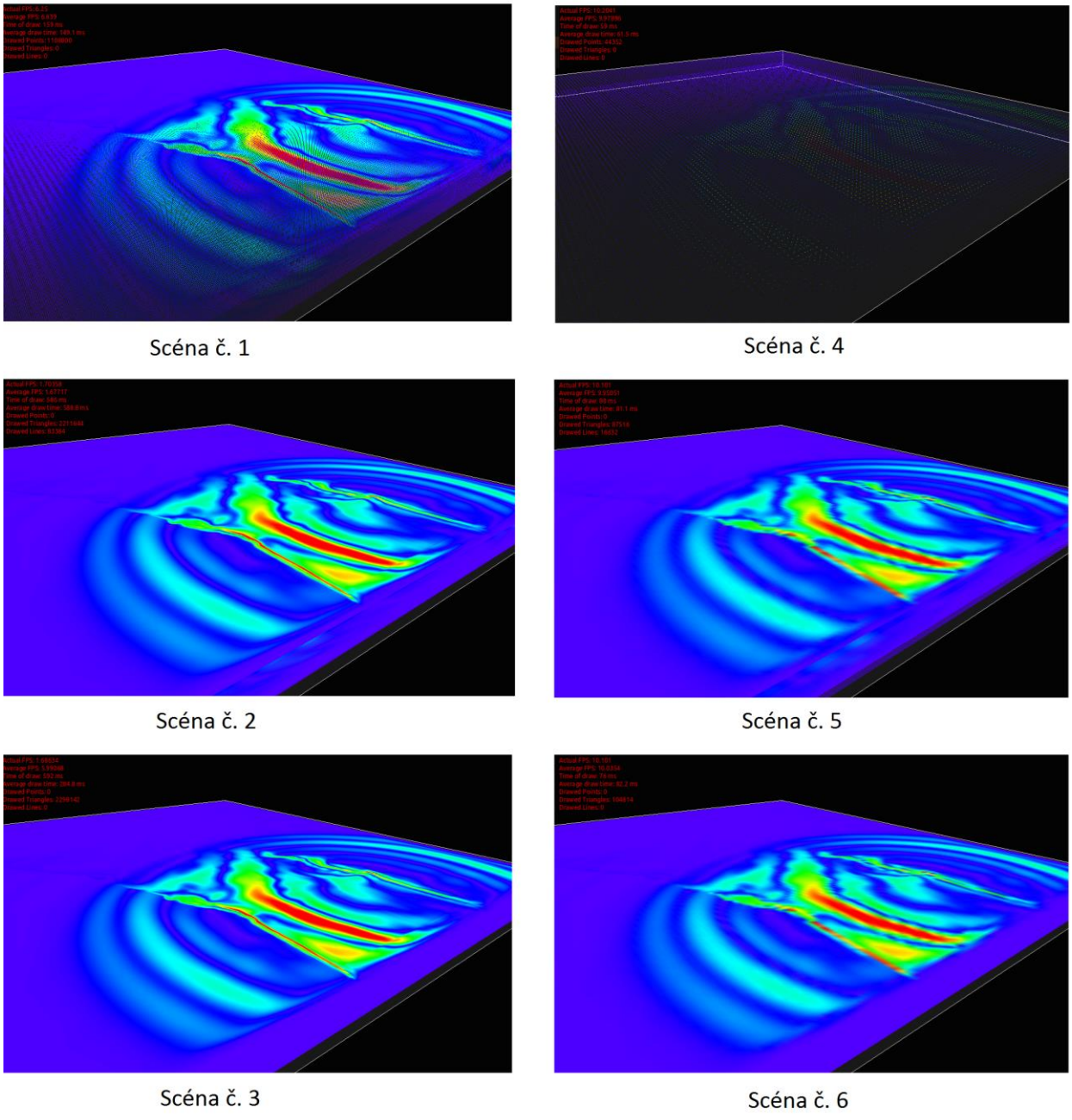


Obrázok 27: Výsledne frame podľa tabuľky 3

Tabuľka 4: Výsledne dáta - výrez

Scéna	Mód vykreslenia	Hustota bodov	Počet vykreslených			Čas vykreslenia v ms	FPS
			Bodov	Čiar	Trojuholníkov		
1	Body	1	1 108 800	0	0	149,1	6,64
2	Platne	1	0	83 384	2 211 644	588,8	1,68
3	Objem	1	0	0	2 298 142	592	1,69
4	Body	5	44 352	0	0	61,5	10
5	Platne	5	0	16 632	87 516	81,1	10
6	Objem	5	0	0	104 814	82,2	10

Posledná tabuľka výsledkov (Tabuľka 4) znázorňuje ako sa dá pomocou iného druhu vykreslenia, resp. vyrezania pracovať s programom v reálnom čase. Ako aj v predchádzajúcej tabuľke tak aj tu sme poukázali na hustotu a jednotlivé druhy vykreslenia. Na nasledujúcom obrázku (Obrázok 28) tak môžeme vidieť aj to, že pri zmene druhu orezania sa kvalita v podstate neznížila, avšak čas vykreslenia sa podstatne zlepšil.



Obrázok 28: Výsledne frame podľa tabuľky 4 - výrez

Ako vidieť na výsledkoch, pri zmenšení kvality sa dá s týmto programom pracovať v reálnom čase. Pri renderovaní sa rozlíšenie obrazu dá nastaviť ľubovoľné, čím sa predĺži aj čas vykreslenia, avšak tento čas pre nás mnohokrát nie je podstatný, keďže fyzicky výpočet šírenia seizmických vĺn je niekoľko násobné dlhší.

Parametre testovacieho PC:

OS:	Linux Ubuntu
RAM:	8GB
HDD:	5400 otáčok/min
CPU:	Intel i7 (2,0 GHz – 4 jadrá, 8 thread)
GPU:	zdieľaná Intel 950

V budúcnosti by sme chceli pokračovať vo vylepšovaní vykreslenia framu pomocou FBO (frame buffer object). Vykresľovanie pomocou FBO by malo zlepšiť rýchlosť vykresľovania objektov (viac v kapitole 7 *Budúcnosť projektu*).

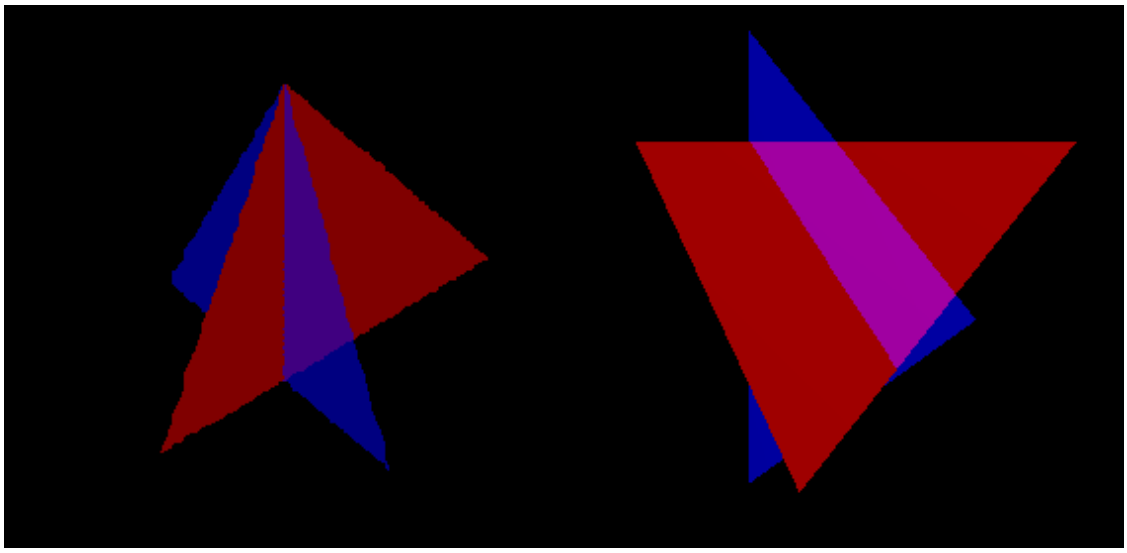
7. Budúcnosť projektu

V budúcnosti chceme určite pokračovať v tomto projekte, na jeho lepších vizualizáciách a optimalizácií algoritmov. Keďže program je vytvorený pomocou nástroja Qt, tak môže byť vyvíjaný v troch najpoužívanejších operačných systémoch (Microsoft Windows, Linux, Mac OS).

Pre vylepšenie vizualizácie by sme chceli vytvoriť nástroj pre stmavenie podložia podľa hĺbky. Správanie by bolo jednoduché, čím bude podložie hlbšie, tým bude farba tmavšia. Pre aplikáciu by sme použili HSL, alebo HSV technológiu pre zníženie intenzity farieb. Keďže pracujeme s farebným modelom RGB, tak prechod z HSL, resp. HSV nebude zložitý. Druhým a zároveň zaujímavejším bude vytvorenie hmlovitých farieb pomocou Ray-Casting algoritmu. Výsledok by mal priniesť vizuálny výsledok podobný ako má SCEC. Tento algoritmus je viac popísaný v kapitole 3.5.1 *Volume Ray-Casting*.

Pre optimalizáciu a tým aj zrýchlenie vykresľovania v reálnom čase by sme chceli prerobiť vykresľovanie pomocou FBO (Frame Buffer Object). Postup vykresľovania pomocou bufferových objektov spočíva v renderovaní načítaných dát a následným uložením do textúr. Namiesto vykreslenia mnoho malých bodov (trojuholníkov) by stačilo vykresľovať tieto textúry ako platne. Rendrovanie do FBO sa už používa v tomto projekte a to pri móde renderovania do súborov. Ďalšou výhodou vykreslenia do bufferových objektov je, že si nemusíme zapamätať všetky body platne. Ak navyše použijeme komprimáciu obrázkov (napr. JPG) tak tým znížime aj nároky na RAM. Vytvorenie prierezov pomocou deliacich rovín by potrebovalo vytvorenie menších mashov. Vznikali by tam menšie objekty (trojuholníky), ktoré by mali pridelené svoje časti textúry. Orezanie mashov by znamenalo implementovanie orezania úsečky a následnej triangulácie. Postup pri orezaní úsečky nie je až taký náročný. Stačí zistiť priesečník priamky s deliacou rovinou, čo je vlastne vyriešenie troch rovníc o troch neznámych ak máme priamku zadanú všeobecne. Pri následnej triangulácií je to trošku ťažšie, pretože musíme brať do úvahy dve možnosti rozdelenia trojuholníka. Prvá možnosť je, ak vzniknú po rozdelení 2 nové trojuholníky. Prípady sa stávajú len výnimočne a to v prípade, že delenie vznikne v jednom z troch vrcholov trojuholníka (Obrázok 29 v ľavo). V tomto prípade vykreslíme stále len jeden trojuholník. Druhým a zároveň častejším spôsobom je ak vzniknú 3 trojuholníky z jedného. Prípad nastane ak deliaca rovina pretne trojuholník na dvoch rôznych stranách (Obrázok 29 v pravo). Môže nastať, že budeme vykresľovať stranu, kde vzniknú dva trojuholníky, alebo tú druhú, kde vznikne len jeden. Napriek týmto výpočtom by

vykreslenie malo byť oveľa rýchlejšie a zaberalo by menej RAM. Pri objemovom móde však musíme brať do úvahy aj krajné body týchto nových trojuholníkov a z nich vytiahnuť farby zo správnych bodov. Orezanie textúry však funguje v 2D a nie v 3D priestore. Budeme si musieť premietnuť tento obraz do 2D pomocou rovnobežného premietania (viac v kapitole 3.1 *Virtuálne prostredie*) a následne useknúť textúru v správnom pomere a nájsť ten správny bod (Liang-Barsky) a jeho farbu.



Obrázok 29: Rozdelenie trojuholníkov

Poslednou optimalizáciou, ktorá je však závislá od hardvéru je prerobenie výpočtov pre CUDA systémy. Pre tento účel je však potrebné zakúpiť grafické karty s podporou CUDA systému. Viac o tomto systéme je v kapitole 5.4 *Cuda*.

Záver

V práci sme prezentovali výsledky, ktoré môžeme zhrnúť nasledovne:

- vizualizovali sme seizmické vlnové pole z výsledkov získaných z externého programu,
- pre vizualizáciu sme využívali štandardné funkcie knižnice OpenGL pre GPU,
- softvér, ktorý sme vytvorili je stabilný pod OS Linux a OS Windows,
- užívateľsky príjemné prostredie sme docielili:
    - využitím ArcBall pre jednoduché natočenie objektov ako aj samotnej scény,
    - využitím quaterniónov na vyriešenie GimBall Lock problému,
    - zahrnutím niekoľkých rôznych možností zobrazenia modelu štruktúry podložia,
    - možnosťou definovania výsekov scény pomocou dvoch rôznych prístupov,
- vytvorili sme možnosť generovania výstupy s lepším rozlíšením a kvalitou detailov ako je schopná zobraziť výpočtová jednotka v reálnom čase,
- vytvorený program môže slúžiť ako ukážka použitia knižnice OpenGL,
- vytvorený program je možné použiť nie len na vizualizáciu seizmických vlnových polí ale aj na vizualizáciu iných časopriestorovo rozložených vektorov,
- program efektívne narába zo systémovými prostriedkami, čím dovoľuje súčasný beh spolu s programom na simuláciu seizmických vlnových polí.

Zdroje

[1] P. MOCZO and P. LABÁK, Zemetrasenia a seizmické ohrozenia, Geofyzikálny ústav SAV, 2000.

[2] "Wikipedia," 2012. [Online]. Available: www.wikipedia.org.

[3] K. Group, "http://www.khronos.org," Gold Standard Group, 2013. [Online].

[4] M. S. Journal, "http://www.microsoft.com," Microsoft Corporation, November 1996. [Online]. Available: http://www.microsoft.com/msj/archive/S2085.aspx.

[5] G. Sidelnikov, "Fallout Software," 2003-2013. [Online]. Available: http://www.falloutsoftware.com/.

[6] A. Chourasia, "Visualization Service," San Diego Supercomputer Center, 2004. [Online]. Available: http://visservices.sdsc.edu.

[7] P. Moczo, "Úvod fyziky Zeme," 2011. [Online]. Available: http://www.fyzikazeme.sk/mainpage/stud_mat/index.htm.

[8] V. Rácko, "3D Vizualizácia Seizmických Vlnových Polí," p. 80, 2005.

[9] "Japan's Quake Ripple Through the Pacific," Visualy inc., 2013. [Online]. Available: http://visual.ly/japan%E2%80%99s-quake-ripple-through-pacific.

[10] N. Pettit, "Art and the Web: Color," Treehouse, 24 10 2011. [Online]. Available: http://blog.teamtreehouse.com/art-and-the-web-color.

[11] C. Zeller, "Cuda Tutotial," NVIDIA Corporation, 2008. [Online]. Available: http://people.maths.ox.ac.uk/~gilesm.

Prílohy

Používateľské rozhranie programu

Náš program ponúka rozhranie pomocou ktorého si používateľ môže nastaviť pohľad na zemské podložie, prípadne Zemský povrch. Scénu si samozrejme môže nielen natočiť, ale aj rozrezať, prezerať z vnútra.

Načítanie a vykreslenie seizmických vĺn

Po spustení programu sa nakonfiguruje automaticky pomocou súboru „config.cfg“, ktorý sa nachádza v tom istom adresári ako aplikácia. Ak sa tam nenachádza tento súbor, tak bude nakonfigurovaný automaticky. Užívateľ si samozrejme môže zadať, kde sa nachádza config súbor, v ktorom sú uložené nastavenia scény.

Pre deliace objekty si užívateľ musí načítať súbor s konfiguráciou týchto objektov, alebo tieto objekty jednoducho vytvorí.

Používateľské nástroje

Požívateľské nástroje nášho programu sú rozsiahle, keďže ide o vizuálny nástroj pre seizmológiu.

Výstupné súbory

Výstupný súbor nášho programu sú frame-ove obrázky v nekomprimovanom formáte BMP. Z týchto súborov je následne jednoduché vytvoriť film, alebo prezentáciu pomocou iných jednoduchších, alebo zložitejších nástrojov. Práve preto sme sa rozhodli, že film nebude dobrým výstupným súborom. Najlepšie je nekomprimované a nekódovane BMP, prípadne pre uloženie alfa kanálu v priehľadnosti PNG, alebo TIFF. Keďže alfa kanál je potrebný len pre vykreslenie seizmických vlnových polí a nie pre celú scénu, tak sme z týchto formátov taktiež upustili.

DVD nosič:

- Elektronická verzia textu (PDF, DOC)
- Zdrojový kód programu
- Exportované pracovné prostredie Linux pre VirtualBox s programom
- Príklad vstupných dát
- Readme pre pokyny