Katedra počítačovej grafiky a spracovania obrazu
Fakulta matematiky, fyziky a informatiky
Univerzita Komenského

Diplomová práca

Vladimír Rácko

BRATISLAVA 2005

Katedra počítačovej grafiky a spracovania obrazu
Fakulta matematiky, fyziky a informatiky
Univerzita Komenského

3D Vizualizácia Seizmických Vlnových Polí

Autor: Vladimír Rácko

Konzultant: Prof. RNDr. Peter Moczo, DrSc.

BRATISLAVA, APRÍL 2005

Čestne vyhlasujem, že som túto diplomovú prácu vypracoval samostatne, len s použitím uvedenej literatúry.

.....

V Bratislave 29.apríla 2005 Vladimír Rácko

Chcel by som poďakovať FMFI UK za možnosť používať vývojové prostredie Borland C++ Builder™ verzie 6.0. Obzvlášť ďakujem svojmu diplomovému konzultantovi, Prof. RNDr. Petrovi Moczovi, DrSc., za všestrannú pomoc, cenné rady a pripomienky pri vypracovaní tejto diplomovej práce.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam používaných skratiek a symbolov

1D, 2D, 3D – Vyjadruje rozmer problému

Cluster – súbor navzájom prepojených počítačov tvoriacich paralelný počítač

Div – celočíselné delenie

Fps – Frames per Second, počet snímkov(obrázkov) za sekundu

Gigabitový Ethernet – počítačová sieť schopná výmeny údajov rýchlosťou 1GB/s

Grid – štruktúrovaná mriežka

GUI – Graphical User Interface, grafické užívateľské rozhranie

Interpolácia – Počítanie hodnoty bodu pomocou hodnôt okolitých bodov

Kodek – Program zabezpečujúci kódovanie a dekódovanie videa

Mesh – Mriežka bodov používaná v počítačovej grafike

MFLOPS – Millions of Floating Point Operations Per Second, počet operácií v plávajúcej radovej čiarke(reálne čísla) vyjadrený v miliónoch

Mod – zvyšok po delení

Normála – kolmica

Pixel – Picture element, hodnota (farba) je konštantná

Rendering – proces projekcie

Superpočítač – Sálový počítač obsahujúci desiatky procesorov, značnú pamäťovú a úložnú kapacitu, predurčený na zložité výpočty a simulácie

Triangulácia – Hranovo maximálny planárny graf obsahujúci všetky vrcholy

Tetrahedron – Štvorsten

Textúra – Udáva výzor materiálu

Voxel – Volume element, objemová bunka, ktorú tvorí najčastejšie kocka s 8 vrcholmi, hodnota je konštantná iba v bodoch gridu, teda vo vrcholoch, inde sa počíta napríklad trilineárnou interpoláciou hodnôt vrcholov

1 Úvod

1.1 Počítač ako prostriedok pre vedecké simulácie

V dnešnej modernej dobe nastal veľký boom informačných technológií, čo poznamenalo nielen informatiku ako takú, ale aj mnoho iných vedných oborov. Výkonné počítače začínajú pomaly prenikať do všetkých sfér nášho života. Výnimkou nie sú ani výskumné pracoviská, kde s nimi majú možnosť pracovať tisícky matematikov, fyzikov, chemikov a mnohých ďalších. Vráťme sa ale o pár rokov späť. Každý z nás si ešte určite pamätá časy, keď počítače neboli schopné poskytnúť dostatočne veľký výkon ani na plynulé prehrávanie videa a tobôž nie na zložité vedecké simulácie. Tieto totiž spravidla pracujú na viacrozmerných vstupných dátach (rádovo až 6-rozmerných), trojrozmerných výstupných dátach a vyžadujú veľkú presnosť. To samozrejme spôsobuje značné „predraženie“ výpočtu, z hľadiska pamäťovej náročnosti a tiež z hľadiska počtu vykonaných operácií.

Problém	**Výpočtová náročnosť(MFLOPS)**	**Pamäťová náročnosť(MB)**
Utriediť 5000 reálnych čísel	0,02	0,02
Prehrať video 640x480x25fps	30,7	1,3
Vygenerovanie obrázku 640x640 bodov ray–castingom	48000	192

Tab.1: Približné výpočtové a pamäťové náročnosti pre niektoré problémy

Procesor	**Výpočtový výkon(MFLOPS)**
Intel Pentium MMX 200 Mhz	50
AMD Athlon 850 Mhz	400
AMD Opteron 2,6 Ghz	3100
Intel Xeon 4 3,06 GHz	4300
Intel Pentium 4 3,6 GHz	4900

Tab.2: Prehľad procesorov s výkonom v MFLOPS

V tab.1 sú znázornené niektoré druhy výpočtov, ich pamäťové a výpočtové zložitosti. V tab.2 pre porovnanie uvádzame výkon viac aj menej výkonných procesorov, ktoré sa používali alebo používajú v osobných a serverových počítačoch.

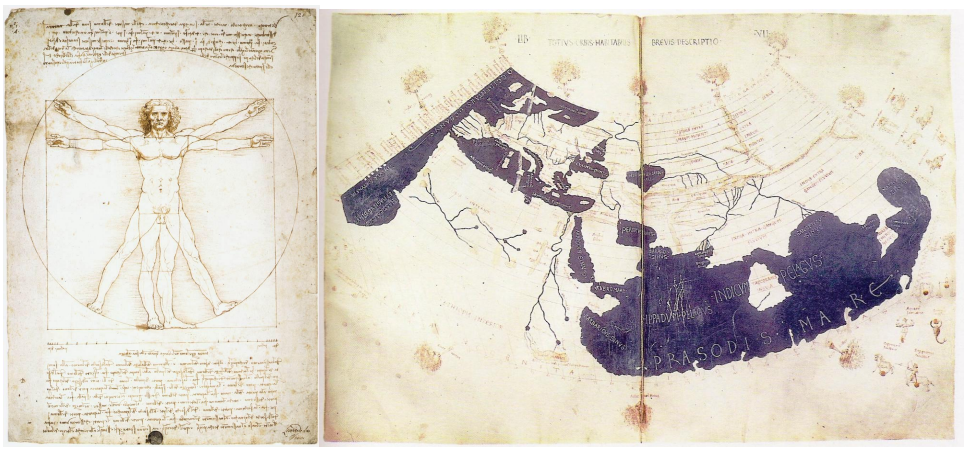
Pozorný čitateľ iste neprehliadol, že staršie modely procesorov neboli schopné spracovávať tak obrovské množstvo údajov takou rýchlosťou, aká by bola potrebná. Z tohto dôvodu musela byť prevažná väčšina vedeckých výpočtov a simulácií realizovaná na sálových *superpočítačoch*, ktoré uspokojili ich nároky na pamäť a výkon. Takéto riešenie však má viacero nevýhod, medzi ktoré patria hlavne:

- **Cena** – Služby superpočítačového centra sú spravidla veľmi drahé
- **Dostupnosť** – Možnosť využívať tieto služby má len úzky okruh ľudí

Našťastie, neustály vývoj v informačných technológiách priniesol na trh nové, výkonnejšie procesory, rýchlejšie, väčšie a lacnejšie pamäte a nové technológie, ktoré umožňujú realizáciu náročných výpočtov a simulácií na štandardných počítačoch. .

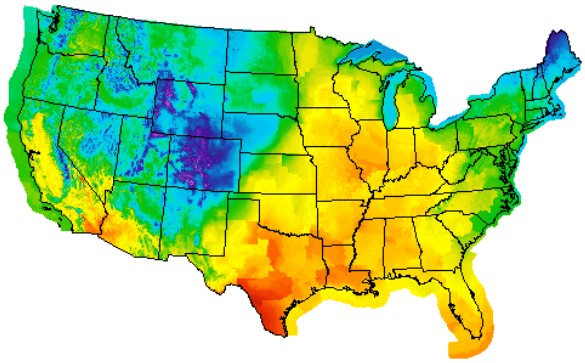
1.2 Čo je to vizualizácia

Slovo „vizualizovať“ pochádza z latinského „visualis“, čo v preklade znamená „týkajúci sa pohľadu“. Vizualizácia je sprostredkovanie zrakového vnemu (obrazu) niečoho, čo je abstraktné, resp. nie je viditeľné. [The Oxford English Dictionary, 1989] Začalo sa používať v 19. storočí v súvislosti z rozmachom fotografie a filmu, avšak prvé vizualizácie vytvorené človekom vznikli oveľa skôr. Ako príklad môžeme uviesť nákresy vynálezcu Leonarda Da Vinciho zo 14. storočia alebo Ptolemaiove kartografické mapy z 2. storočia, obr.1.



Obr.1: Ukážky historických vizualizácií, zdroj [Wikipedia]

V súčasnosti sa význam slova „vizualizácia“ spája viac-menej s informačnými technológiami, presnejšie s počítačovou grafikou, ktorá využíva výpočtovú silu počítačov a grafických akcelerátorov na hromadné spracovanie a efektné zobrazovanie dát, ktoré by boli inak ťažko predstaviteľné alebo opísateľné. Na obr.2 je napríklad vizualizovaná teplota na celom území USA.



Obr.2: 2D vizualizácia teplôt na území USA, zdroj [Weather]

Cieľom je teda čo najpresnejšie a najnázornejšie zobraziť daný súbor údajov, ktorý je charakterizovaný danými vstupnými hodnotami, teda ohodnotením všetkých vstupných parametrov. Výstupom procesu vizualizácie je spravidla obrazová informácia, teda obrázok alebo animácia, čo závisí od typu.

Rozlišujeme dva prípady vizualizácií:

- **Statická vizualizácia** – vstupné dáta sa nemenia
- **Dynamická vizualizácia** – vstupné dáta sa môžu meniť

V prvom prípade teda máme statické dáta, z ktorých výpočtom získame na výstup statický *obrázok*. V druhom prípade nám pribudol ďalší rozmer – čas, pričom v každom časovom okamihu sa môžu meniť jednak vstupné hodnoty a taktiež parametre procesu, ako napríklad rýchlosť priebehu alebo presnosť výpočtu. Týmto spôsobom dostávame na výstup sadu obrázkov, z ktorých každý reprezentuje jeden časový okamih. Tieto obrázky môžeme následne spojiť do výslednej *animácie*.

1.3 Význam vizualizácií

Existuje viacero oblastí ľudského bádania, v ktorých sa spracúva veľké množstvo údajov viacerých druhov. Ide o vedné disciplíny, ako fyzika, matematika, biológia, ale aj napríklad ekonómia alebo manažment. Uvažované dáta sú obvykle značne objemné, majú viacero

rozmerov a charakteristík, čo predstavuje pre ľudský mozog problém. Jedným z riešení, ako takýmto dátam porozumieť a vytvoriť si v nich prehľad, je proces vizualizácie, ktorý nám umožňuje zvoliť si:

- **charakteristiku (charakteristiky)**, ktoré sa budú zobrazovať, pričom zvyšné dáta sa nebudú brať do úvahy, čím dosiahneme lepšiu prehľadnosť.
- **metódu**, ktorou sa budú dáta zobrazovať, pričom výber závisí od konkrétnych dát.
- **rozsah** zobrazovaných dát (z hľadiska času, priestoru), ktorý sa bude brať do úvahy.
- **presnosť** vizualizácie, od ktorej priamo závisí výpočtová (teda aj časová) náročnosť celého procesu. Čím väčšiu presnosť chceme docieliť, tým viac dát treba spracovať a tým dlhšie bude trvať celý výpočet. V prípade, že si zo spomínaných dôvodov nemôžeme dovoliť vysokú presnosť, jednou z možností na zvýšenie kvality výsledného produktu je použitie interpolačných, resp. aproximačných techník, ktoré podrobnejšie spomenieme v kapitole 2.3.3 a 5.6.

Vhodnou voľbou uvedených parametrov získame z objemného vstupného súboru dát výslednú vizualizáciu, obsahujúcu len tie prvky, ktoré potrebujeme, navyše prezentované spôsobom, ktorý je prispôsobený ľudskému vnímaniu. Výsledok nájde uplatnenie najmä pri zobrazovaní rôznych nameraných hodnôt, napríklad animácia vytvorená na základe seizmogramami nameraných amplitúd zemetrasenia v jednotlivých bodoch a časových okamihoch, môže poskytnúť názorný prehľad o priebehu celého zemetrasenia v oblasti. Ďalším príkladom z medicínskej oblasti môže byť priebeh mozgovej aktivity získaný z nameraných hodnôt elektrických impulzov mozgovej kôry. Okrem nameraných hodnôt, môžu byť vstupom aj dáta získané výpočtom, čo vytvára priestor pre rôzne vedecké simulácie a experimenty. V prípade seizmologického výskumu je vizualizácia dobrou pomôckou pri predpovedaní pohybov zemskej kôry, pri odhadoch rizík s tým spojených a celkovo na pochopenie dynamického správania našej planéty.

1.4 Cieľ práce

Koncepcia práce vychádza zo spolupráce Katedry počítačovej grafiky a spracovania obrazu FMFI UK s Katedrou fyziky Zeme a planét FMFI UK. Autormi návrhu sú Prof. RNDr. Peter Moczo, DrSc. a Mgr. Jozef Kristek, PhD. z fyzikálnej katedry. Úlohou práce je implementovať vizualizáciu vypočítaných dát fyzikálneho modelu popisujúceho propagáciu

seizmických vĺn cez dané prostredie. Získané dáta prostredia a vlnového poľa budú názorne zobrazené, pričom bude možné nastavovať množstvo parametrov. Výstupom programu budú buď jednotlivé obrázky alebo animácia vlnového poľa. Výsledná aplikácia pobeží na výkonnejšom osobnom počítači s grafickou kartou podporujúcou štandard OpenGL. Podrobnejšie, viď kapitolu 4.11.

Aplikácia je primárne určená ako pomôcka pri výskume seizmických vlnových polí, výstupy budú slúžiť tiež ako prezentačný materiál použitých metód výpočtu vlnového poľa.

1.5 Zadanie

Zadanie bolo konštruované spolu s autormi návrhu z fyzikálnej katedry a je zosúladené z ich požiadavkami na finálny produkt. Z globálneho hľadiska je potrebné vytvoriť používateľsky prítulné *GUI*, ktoré by zastrešovalo jednotlivé požadované prvky. Úlohu teda môžeme rozdeliť do niekoľkých častí.

- **Načítať zo vstupu namerané hodnoty terénu**
- **Zobraziť scénu**
  - V scéne bude umiestnená nádoba tvaru kvádra, ktorá bude obsahovať jednotlivé prvky vizualizácie.
  - Nádoba sa bude dať ľubovoľne otáčať, posúvať a škálovať.
  - V scéne bude umožnený voľný pohyb kamery.
  - Nastavenia polohy nádoby a kamery sa budú dať uložiť/obnoviť.
  - Bude možné vyrezať časť nádoby rovinami rovnobežnými so súradnicovými osami a spustiť vizualizáciu vo vyrezanej časti alebo v doplnku k vyrezanej časti.
  - Nádobu bude možné prerezať ľubovoľnou rovinou a spustiť vizualizáciu v časti „pred“ alebo „za“ rovinou.
- **Zobraziť terén**
  - Kvôli jednoduchosti a prehľadnosti sa bude zobrazovať len časť terénu, tzv. rozhranie, ktoré oddeľuje oblasti s rozdielnymi fyzikálnymi charakteristikami s hľadiska seizmologického výskumu (hustota, rýchlosť propagácie vĺn).
  - Členitosť rozhrania bude zvýraznená, farbou, textúrou alebo osvetlením.

- Možnosť meniť transparentnosť rozhrania kvôli lepšej viditeľnosti vlnového poľa.

- **Načítať zo vstupného súboru vypočítané amplitúdy vlnového poľa**
- **Zobraziť vlnové pole**
  - Na vizualizáciu vlnového poľa sa použije „hmlový efekt“, pričom čím bude v danom bode amplitúda vlnenia väčšia, tým bude hmla hustejšia.
  - Zobrazovať sa budú jednotlivé časové snímky vlnového poľa, pričom výber želaného snímku bude realizovaný pomocou časovej osi.
  - Užívateľ bude mať možnosť výberu charakteristiky poľa, ktorá sa bude vizualizovať (U-zložka amplitúdy vychýlenia, V-zložka, W-zložka alebo výslednica).
- **Export**
  - Želaný rozsah vizualizácie sa vyznačí na časovej osi.
  - Užívateľ si zvolí formát výstupu (séria obrázkov alebo animácia), v prípade animácie si zvolí spôsob kódovania (kodek).
  - Exportovať sa bude zvolený rozsah snímok - animácia(video formát AVI) alebo obrázok zodpovedajúci určitému časovému okamihu(grafický formát BMP)

.

2 Základné pojmy

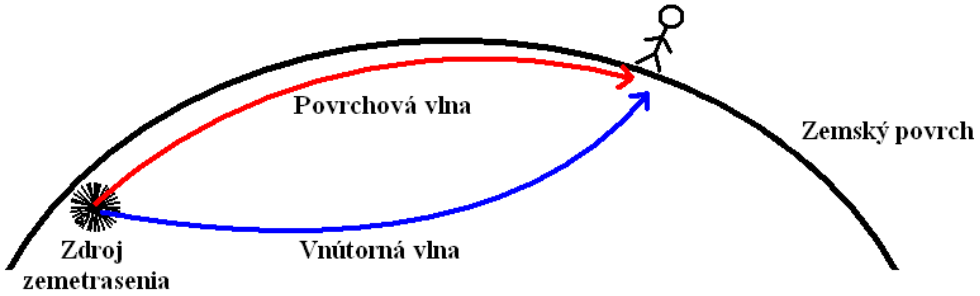
2.1 Seizmické vlnenie

Materiál prezentovaný v tejto kapitole je čerpaný zo zdrojov [Janotka04] a [Chopra02].

Predmetom nášho skúmania bude elastické vlnenie, šíriace sa v danom médiu všetkými smermi, pričom budeme predpokladať, že médium je perfektne elastické a izotropné. Z fyzikálneho hľadiska elastické vlnenie predstavuje šírenie deformácií a napätí, ktoré sú navzájom definované Hookovým zákonom. Teleso na ktoré pôsobia vonkajšie deformujúce sily mení jednak svoj objem, ale aj tvar. Hovoríme o objemovej a tvarovej deformácii. Ak deformačná sila prestane pôsobiť, teleso nadobudne svoju pôvodnú formu. Takéto teleso označujeme ako elastické. Deformácie telesa pod vplyvom pôsobenia síl vyvolávajú vždy vznik napätí na jeho povrchu. Medzi deformáciami a napätiami existuje fyzikálna väzba, vyjadrená už spomenutým Hookovým zákonom. Šírenie elastických vĺn nie je teda ničím iným, ako všesmerovým šírením deformácií a napätí v prostredí, kde pôsobia vnútorné, alebo vonkajšie sily. Deformácie spôsobujú pohyb hmotných častíc prostredia, cez ktoré prechádzajú a tieto vykonávajú krátkodobé kmity. Tento pohyb sa postupne rozširuje od jednej častice k druhej. Vzniká elastická vlna, šíriaca sa v prostredí určitou rýchlosťou. Každý typ deformácie je spojený so vznikom určitého typu elastického vlnenia. V danom prostredí sa môžu šíriť dva základné typy vĺn, ktoré delíme ďalej na:

- **Vnútorné** – šíria sa vo vnútri média
  - Primárne (kompresné, pozdĺžne) vlny (P-vlny)
  - Sekundárne alebo tiež priečne vlny (S-vlny)
- **Povrchové** – šíria sa na povrchu (podobne ako vlny na mori)
  - Love-ove vlny (L-vlny)
  - Rayleigh-ove vlny (R-vlny)

Oba typy sú pre názornosť vyobrazené na obr.3.

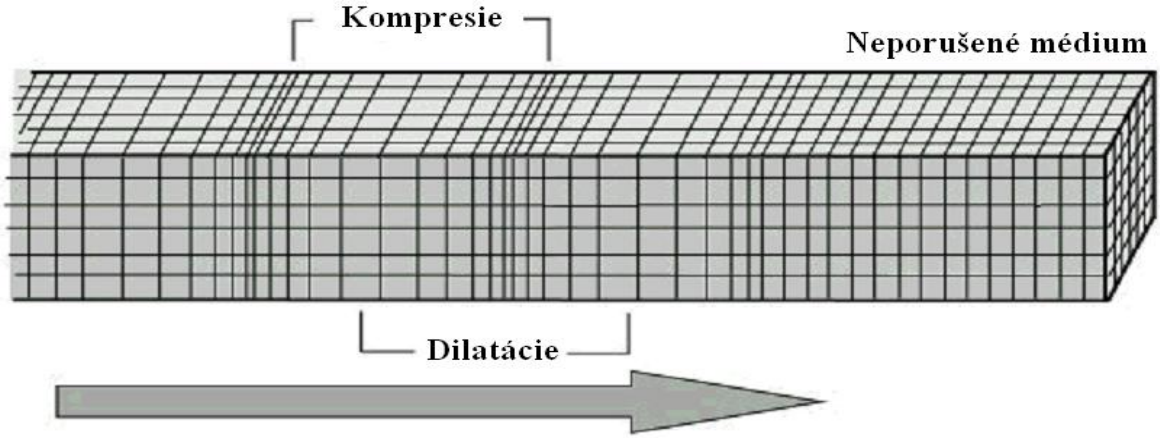


Obr.3: Vnútorné a povrchové vlny.

Pozdĺžne vlny P (obr.4) sú spojené len s objemovými deformáciami. Pri jej šírení kmitajú hmotné častice prostredia v smere šírenia pozdĺžnej vlny. Rýchlosť šírenia pozdĺžnej vlny označujeme ako V_P. Vypočítame ju podľa vzťahu

$$V_P \equiv \sqrt{\frac{l+2m}{r}}$$

kde λ a μ sú tzv. Lame-ove koeficienty (pružné moduly) a ρ je objemová hustota prostredia, v ktorom sa pružné vlny šíria. Jedná sa o najrýchlejší typ seizmických vĺn, ktoré sa šíria v Zemskej kôre rýchlosťou až 6 km za sekundu. Sú schopné prechádzať cez pevné (kamenné vrstvy Zemskej kôry) ako aj cez kvapalné prostredia (voda, tekuté vrstvy Zemskej kôry). Ich šírenie sa dá prirovnať šíreniu zvukových vĺn vzduchom.



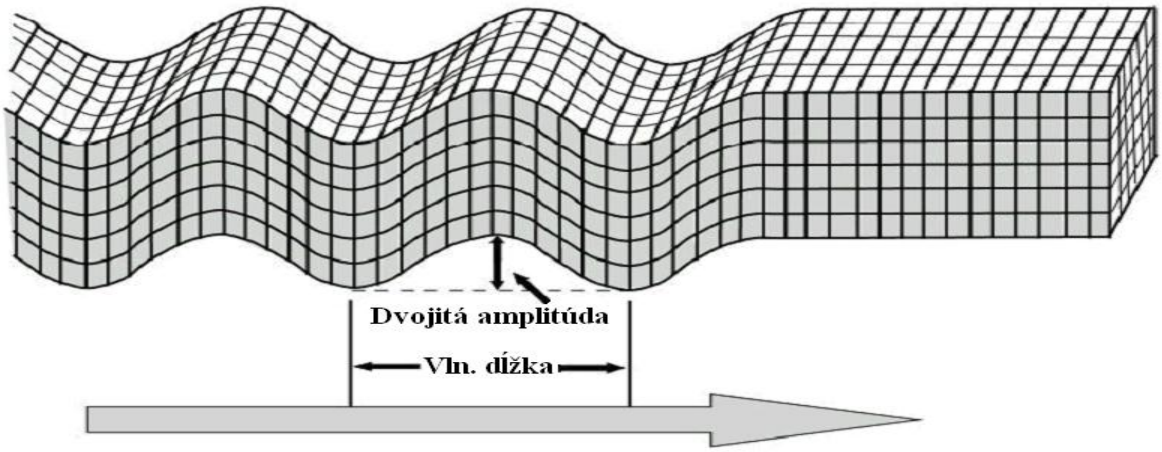
Obr.4: P-vlna, zdroj [Chopra02]

Priečne vlnenie S (obr.5) je viazané len na tvarové deformácie. Pri jej šírení prostredím sa hmotné častice pohybujú kolmo na smer šírenia vlny. Ako možno vidieť z obr.6, podmienka kolmosti pohybu častíc prostredia na smer šírenia priečnej vlny je splnená tak v rovine vertikálnej (SV), ako aj v rovine horizontálnej (SH). Existujú teda dva typy priečnych

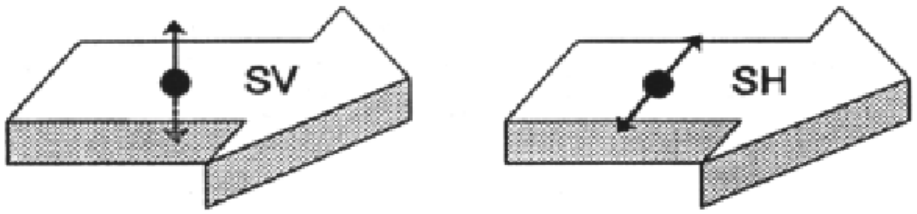
vĺn – SV a SH. SV hýbu prostredím vo vertikálnom (zhora nadol) a SH v horizontálnom smere (zo strany na stranu). Rýchlosť šírenia V_S vypočítame podľa vzťahu

$$V_S \equiv \sqrt{\frac{m}{r}}$$

Priečne vlny sa môžu šíriť len v pevných prostrediach a ich rýchlosť je menšia, ako rýchlosť šírenia pozdĺžnej vlny v danom prostredí, v Zemskej kôre je to 3,6 km za sekundu.



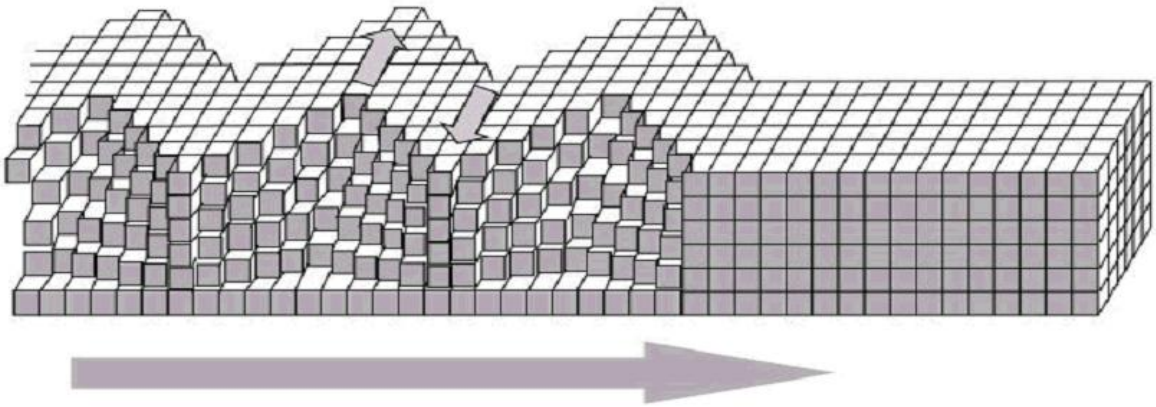
Obr.5: S-vlna (presnejšie SV-vlna), zdroj [Chopra02]



Obr.6: SV a SH-vlny, zdroj [Janotka04]

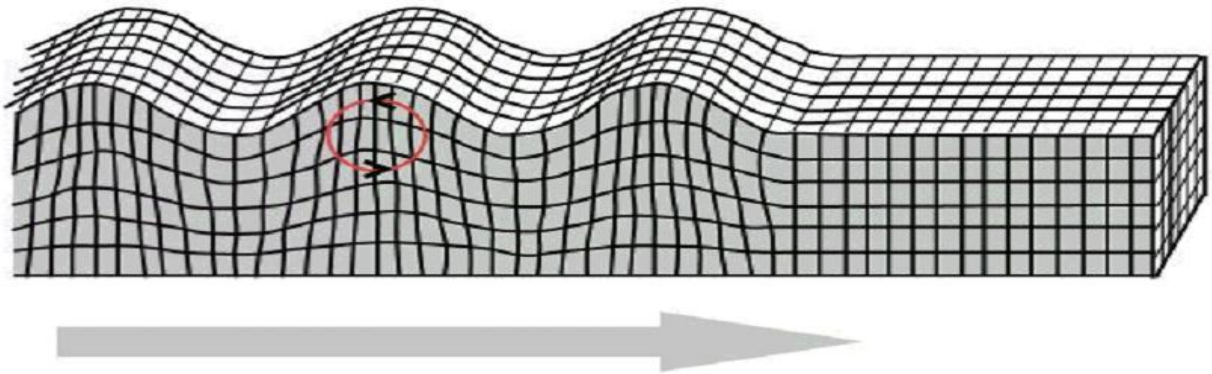
V dôsledku budenia seizmickej energie vznikajú okrem vnútorných vĺn aj intenzívne povrchové vlny. Tieto sa šíria po povrchu prostredia a pri zemetrasení ich najviac vnímame. Rozlišujeme dva druhy povrchových vĺn: Love-ove vlny a Rayleigh-ove vlny.

Prvým typom povrchových vĺn sú tzv. Love-ove vlny (obr.7), ktoré boli pomenované podľa britského matematika menom A.E.H. Love, ktorý v roku 1911 navrhol matematický model týchto vĺn. Love-ova vlna vzniká len za podmienky, že na voľnom polpriestore leží vrstva konečnej mocnosti. L vlna vyvoláva kmitanie častíc prostredia v horizontálnej rovine, kolmo na smer šírenia L vĺn, ide teda o priečne vlnenie typu SH. Ide o najrýchlejší typ povrchových vĺn, rýchlosť šírenia je väčšia, než rýchlosť V_{S1} vo vrstve a menšia, než V_{S2} v polpriestore, na ktorom vrstva leží.



Obr.7:L-vlna, zdroj [Chopra02]

Ďalším typom povrchových vĺn sú tzv. Rayleigh-ove vlny (obr.8), pomenované podľa Lorda Rayleigh-a, ktorý dokázal ich existenciu v roku 1885. Tieto vlny rolujú pozdĺž povrchu podobne ako rolujúce vlny na mori. R-vlny sú šírením objemovej aj tvarovej deformácie. Sú to vlny disperzné (rýchlosť sa mení z frekvenciou vlny) a ich rýchlosť je menšia, než rýchlosť priečnych vĺn (V_R =0.9*V_S). Nútia kmitať častice prostredia po eliptických dráhach vo vertikálnej rovine. Väčšina otrasov počas zemetrasení je spôsobená práve týmto typom vĺn.



Obr.8: R-vlna, zdroj [Chopra02]

Spomenuli sme si teda rôzne typy seizmických vĺn, ktoré sa vyskytujú pri zemetraseniach. V ďalšom sa obmedzíme len na P a S vlny, ktoré sa najčastejšie používajú na fyzikálne simulácie z dôvodu jednoduchosti a prehľadnosti výstupu. Pri počítaní simulácií sú potrebné rýchlosti P a S vĺn v uvažovaných médiách a tiež hustota týchto médií. Nasledujúca tabuľka (tab.3) zobrazuje tieto údaje pre vybrané prostredia.

Prostredie	Hustota ($kg*m^{-3}$)	Rýchlosť P-vĺn ($m*s^{-1}$)	Rýchlosť S-vĺn ($m*s^{-1}$)
Vzduch	1,21	332	-
Voda	1000	1400-1500	-
Ropa	820	1300-1400	-
Oceľ	7850	6100	3500
Betón	2300	3600	2000
Žula	2600	5500-5900	2800-3000
Pieskovec	2323	1400-4300	700-2800
Vápenec	2000	5900-6100	2800-3000
Piesok	1201	200-1000	80-100
Hlina	1000-1900	1000-2500	400-1000

Tab.3: Hustota a rýchlosti P a S vĺn pre jednotlivé prostredia

Seizmické vlnenie sa teda šíri od zdroja všetkými smermi určitou rýchlosťou, ktorá je podmienená fyzikálnymi vlastnosťami prostredia a typom šíriacej sa vlny. Častice prostredia sú prechádzajúcou seizmickou vlnou uvádzané do pohybu, ktorého charakter závisí tiež od typu šíriacej sa seizmickej vlny. V našej simulácii nás bude zaujímať práve tento pohyb, pričom budeme vizualizovať jeho amplitúdu.

2.2 Fyzikálny model

V nasledujúcom popíšeme spôsoby a metódy, ktorými možno reálne modelovať propagáciu seizmických vĺn cez dané médium, [Moczo04].

Uvažujme Karteziánsku súradnicovú sústavu *(x, y, z)*. Nech sú ρ (objemová hustota prostredia) a λ, μ (Lame-ove koeficienty, pružné moduly) funkciami priestorových súradníc *x, y, z*. Nech sú vektor posunutia $\bar{u}(u, v, w)$, tenzor napätia $t_{ek}; e,k \in \{x,y,z\}$ a objemová sila $\bar{f}(f_x, f_y, f_z)$ funkciami priestorových súradníc *(x, y, z)* a času *t*. Na popis seizmického vlnenia slúžia vlnová rovnica a Hookov zákon, ktoré môžeme v prípade perfektne elastického a izotropného prostredia vyjadriť ako:

$$r u_{tt} = t_{xx,x} + t_{xy,y} + t_{xz,z} + f_x \qquad t_{xx} = (l+2m)u_x + l v_y + l w_z \qquad t_{xy} = m(u_y + v_x)$$
$$r v_{tt} = t_{xy,x} + t_{yy,y} + t_{yz,z} + f_y \qquad t_{yy} = l u_x + (l+2m)v_y + l w_z \qquad t_{xz} = m(u_z + w_x)$$
$$r v_{tt} = t_{xz,x} + t_{zy,y} + t_{zz,z} + f_z \qquad t_{zz} = l u_x + l v_y + (l+2m)w_z \qquad t_{yz} = m(v_z + w_y)$$

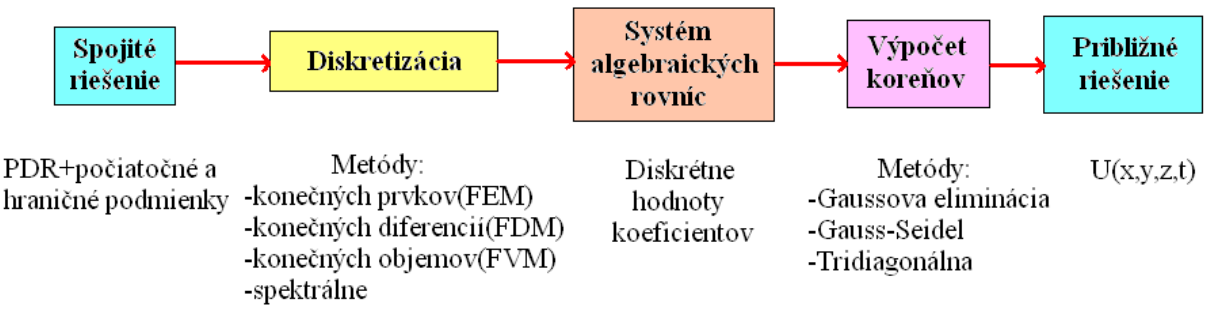
kde $u_{tt} = \frac{\partial^2 u}{\partial t^2}, t_{xx,x} = \frac{\partial t_{xx}}{\partial x}, u_x = \frac{\partial u}{\partial x}$, atď. Na simuláciu propagácie seizmických vĺn cez dané médium je potrebné tieto rovnice vyriešiť. Jedná sa o parciálne diferenciálne rovnice (ďalej len PDR), ktoré sa dajú riešiť dvoma spôsobmi:

- **Analyticky**
- **Numericky**

Analytické riešenia sú síce presné, ale riešiť sa nimi dajú len pomerne jednoduché lineárne parciálne diferenciálne rovnice, pričom sa musíme obmedziť na jednoduché počiatočné a hraničné podmienky. Pri analytickom riešení týchto rovníc sa používajú rôzne metódy, napríklad metóda separácie premenných, metóda variácie konštánt alebo Laplaceova transformácia, podrobnosti v [Farlo93]. Na modelovanie zložitých procesov, ako je napríklad šírenie vĺn vo vnútri Zeme, ktoré je vo všeobecnosti heterogénne so zložitou sústavou vrstiev a blokov s meniacimi sa materiálovými vlastnosťami, si s jednoduchými rovnicami nevystačíme. Preto našu pozornosť obrátime na približné numerické metódy riešenia.

2.2.1 Numerické riešenie PDR

Numerické riešenie spočíva v prevode spojitého popisu javu (pomocou parciálnej diferenciálnej rovnice) na diskrétny popis (sústava algebraických rovníc), z ktorého vieme získať riešenie omnoho efektívnejšie. Samozrejme, numerické metódy nikdy nie sú absolútne presné, ale pre každú vopred danú chybu *e* vieme nájsť riešenie, ktorého chyba je menšia ako *e*. Vždy sa snažíme dosiahnuť čo najväčšiu presnosť pri zachovaní čo najmenšej zložitosti. Vyššie opísaný postup je názorne ilustrovaný na obr.9.



Obr.9: Postup pri numerickom riešení PDR

Hlavnou časťou numerického riešenia je diskretizácia PDR, ktorá spojitú funkciu v PDR reprezentuje konečným počtom koeficientov. Jednotlivé metódy sa líšia práve spôsobom

reprezentácie koeficientov a tiež technikou, akou aproximujú derivácie v PDR. Neexistuje univerzálna metóda schopná efektívne riešiť ľubovoľný problém, a preto je kľúčové zvoliť si k danému problému, resp. PDR tú najvhodnejšiu. Medzi najznámejšie a najpoužívanejšie patria metóda konečných prvkov, metóda konečných diferencií, metóda konečných objemov alebo tiež spektrálne metódy. Pri modelovaní seizmického vlnenia môžeme požiadavky, ktoré musí spĺňať numerická metóda, zhrnúť do nasledujúcich bodov:

- musí byť schopná počítať s heterogénnym prostredím vrátane materiálových rozhraní
- musí počítať s reálnou atenuáciou
- musí umožňovať dostatočný frekvenčný rozsah vlnení

Súčasne by sme mali zohľadniť nasledujúce kritériá:

- **Numerická presnosť** – odhady chýb
- **Numerická stabilita** – analýza stability metódy, podmienky konvergencie
- **Numerická efektivita** – v spojitosti s časovou a pamäťovou zložitosťou
- **Spoľahlivosť a flexibilita metódy** – možnosť použiť na skupinu problémov

2.2.2 Metóda konečných diferencií

Po zvážení všetkých aspektov jednotlivých metód, dospeli seizmológovia k názoru, že najvhodnejšou metódou na modelovanie seizmických vlnení a pohybov Zeme je metóda konečných diferencií (FDM). FDM patrí do skupiny tzv. grid–point (grid=mriežka, point=bod) metód, pretože výpočtová oblasť (doména) sa pokryje priestorovou a časovou mriežkou, pričom všetky funkcie sú reprezentované len ich hodnotami v jednotlivých mrežových bodoch, čím je zabezpečená diskretizácia PDR. Tieto hodnoty sa použijú aj na výpočet diferencií, ktorými sa vo FDM aproximujú derivácie funkcií. Funkčné hodnoty medzi mrežovými bodmi sa neberú do úvahy. Rozmiestnenie mrežových bodov môže byť vo všeobecnosti ľubovoľné, ale v značnej miere ovplyvňuje výslednú presnosť metódy.
Medzi hlavné výhody FDM patria:

- je použiteľná na simuláciu komplexných a zložitých modelov
- je relatívne presná, na zvýšenie presnosti existuje možnosť použiť schémy vyšších rádov, ale zníži sa tým efektivita a stabilita metódy
- je výpočtovo efektívna, má prijateľné nároky na pamäť a čas výpočtu
- ľahko sa programuje

- umožňuje paralelné spracovanie na viacerých procesoroch alebo počítačoch

Hlavnou nevýhodou FDM je problém s implementáciou hraničných podmienok, ktorý sa prejaví hlavne pri použití metódy na komplexné a zložité modely.

Aplikácia FDM na daný problém (reprezentovaný PDR) spočíva v nasledujúcich krokoch:

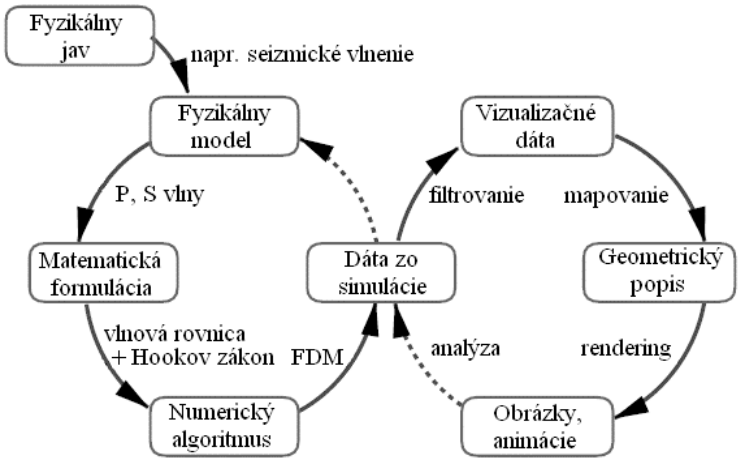
- Konštrukcia diskrétneho modelu problému pomocou konečných diferencií
  - pokrytie výpočtovej oblasti priestorovou a časovou mriežkou
  - výpočet aproximácií k deriváciám funkcií, k počiatočným a hraničným podmienkam, samozrejme len v mrežových bodoch
  - zostavenie konečného systému algebraických rovníc
- Analýza modelu
  - konzistencia systému a rád aproximácie
  - stabilita
  - konvergencia
- Samotný numerický výpočet

Podrobným popisom jednotlivých krokov sa nebudeme v tomto texte zaoberať, podrobnejšie informácie nájde čitateľ v [Aki02], [Moczo98], [Moczo00] a [Moczo04]. Výstupom metódy FDM bude súbor amplitúd vlnenia *(u, v, w)* v smere súradnicových osí *(x, y, z)* v jednotlivých bodoch mriežky, ktoré je možné vizualizovať rôznymi spôsobmi, ktoré si v ďalšom texte podrobnejšie popíšeme.

2.3 Vizualizácia s využitím počítačovej grafiky

V nasledujúcej kapitole si stručne popíšeme princíp vizualizácie dát na počítači s využitím možností, ktoré nám ponúka moderný hardware a software.

Ako sme spomenuli v kapitole 1.2, vizualizácia má za úlohu transformovať neprehľadné dáta v počítačovej podobe (súbor núl a jednotiek) do vizuálnej podoby (obrázok, animácia), pričom výsledok by mal byť maximálne zrozumiteľný ľudskému vnímaniu. Tento proces je ilustrovaný na obr.10.

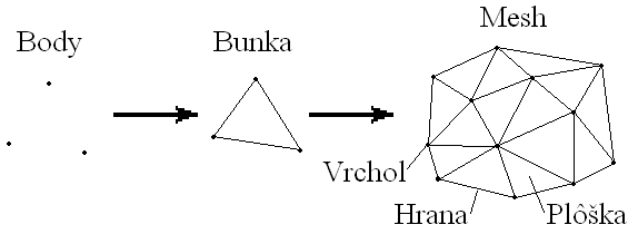


Obr.10: Proces simulácie a vizualizácie

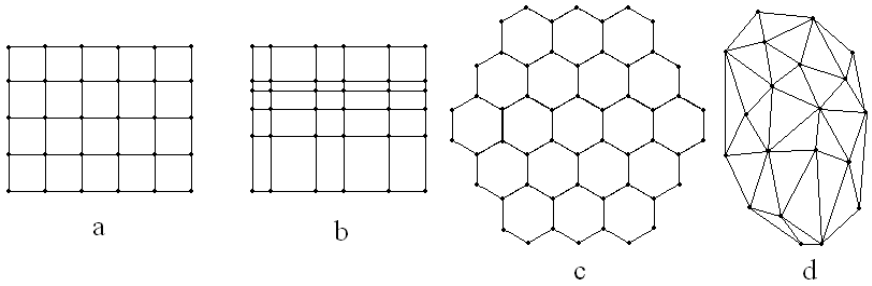
Obr.10 ilustruje vizualizáciu seizmických dát získaných numerickým výpočtom, ale vstupom procesu môžu byť tiež dáta namerané senzormi alebo syntetické dáta určené najmä na testovanie funkčnosti procesu vizualizácie.

Vstupné dáta sa v prvej fáze procesu prefiltrujú, teda vyberú sa z nich len tie, ktoré máme záujem zobraziť, pričom ich typ (rozmer) nemusí korešpondovať s rozmerom domény. Príkladom môže byť vizualizácia sily prúdenia vzduchu pri predpovedi počasia. V tomto prípade má doména rozmer 3 (priestorové súradnice *(x, y, z)* s funkčnou hodnotou sily prúdenia *f*) a vizualizačné dáta sú len dvojrozmerné (súradnice *(x, y)* na mape s funkčnou hodnotou sily prúdenia *f* reprezentovanou napríklad farbou).

V druhej fáze sa snažíme získané dáta akýmsi spôsobom pretransformovať do geometrickej podoby zobraziteľnej na počítači. Preto v závislosti od toho, čo chceme zobraziť, vytvárame zo vstupných dát (zvyčajne bodov) geometrické primitívy (bod, úsečka, trojuholník, štvorsten,...), s ktorými pracuje grafický hardvér. V prípade, že chceme zobraziť zložitejší objekt, generujeme z vytvorených primitív (buniek), alebo priamo zo vstupných bodov pomocou algoritmov počítačovej grafiky, siete (*mesh*). Na obr.11 je znázornené generovanie trojuholníkového meshu.



Obr.11: Generovanie 2D meshu



Obr.12: Rôzne typy 2D meshov

Z hľadiska štruktúry rozdeľujeme meshe na štruktúrované a neštruktúrované. V prípade štruktúrovaných majú všetky body meshu rovnaký počet susediacich buniek, čo v prípade neštruktúrovaných neplatí. Štruktúrované meshe nazývame tiež *gridy*. Na obr.12 vidieť rôzne typy 2D meshov, a) uniformný grid, b) obdĺžnikový grid, c) šesťuholníkový grid, d) neštruktúrovaný trojuholníkový mesh. Z hľadiska dimenzie rozlišujeme 2D a 3D meshe, iné rozmery sa v praxi takmer nepoužívajú. Od typu spravidla závisí reprezentácia meshu v dátovej štruktúre ako aj voľba algoritmov na generovanie, spracovanie a vykresľovanie meshu do obrazového priestoru.

2.3.1 Generovanie meshu

Na generovanie meshov z n-tice vstupných bodov existuje v počítačovej grafike nepreberné množstvo algoritmov, z ktorých budú pre nás zaujímavé najmä algoritmy *triangulácie v rovine*, keďže budeme zobrazovať trojuholníkový mesh (bude ním rozhranie vrstiev, viď. kapitolu 1.5, 4.3.2 a 5.4). Triangluláciou množiny bodov v rovine rozumieme hranovo maximálny planárny graf, daný *n* bodmi. Triangulácií *n* bodov vo všeobecnosti existuje exponenciálny počet, no pre účely počítačovej grafiky je najvhodnejšia ekviangulárna triangulácia. Pri zobrazovaní trojuholníkov s veľkými rozdielmi vo vnútorných uhloch (dlhých a úzkych trojuholníkov) vznikajú chyby z dôvodu počítačovej aritmetiky. Ekviangularita, teda rovnosť vnútorných uhlov trojuholníkov triangulácie, je však dosiahnuteľná iba v triangulácii z rovnostranných trojuholníkov, čo možno docieliť len pre veľmi špecifický vstup. Pod ekviangularitou sa preto v praxi rozumie podobná veľkosť vnútorných uhlov, teda ekviangulárnou triangluláciou nazývame takú triangluláciu, ktorá minimalizuje maximálny uhol. Za predpokladu, že žiadne 4 vstupné body nie sú kocirkulárne (neležia na 1 kružnici), ekviangulárnu triangluláciu získame postupným budovaním lokálne ekviangulárnych trojuholníkov. Takáto triangulácia sa nazýva Delaunayova. Medzi jej ďalšie užitočné

vlastnosti patrí aj kritérium prázdneho kruhu, ktoré platí pre každú dvojicu trojuholníkov a hovorí, že opísaná kružnica jedného z nich neobsahuje vo svojom vnútri zostávajúci vrchol druhého trojuholníka. Konkrétnu aplikáciu Delaunayovej triangulácie pre náš konkrétny prípad spolu s algoritmom na jej vytvorenie uvedieme v kapitole 5.4. Podrobnejšie informácie sa čitateľ dozvie zo zdroja [Chalmo01], z ktorého sme čerpali pri písaní materiálu o trianguláciách.

2.3.2 Reprezentácia meshu

Vo všeobecnosti môžeme mesh reprezentovať viacerými spôsobmi ([Luber00]):

- zoznamom vrcholov (explicitne, implicitne)
- globálnym indexom vrcholov
- zoznamom buniek (explicitne, implicitne)
- globálnym indexom buniek
- zoznamom hrán
- zoznamom plôšok (v prípade 3D meshu, napríklad, keď chceme vyjadriť hranicu)

Uveďme si príklad pre 3D uniformný grid, ktorý neskôr využijeme pri práci s vlnovým poľom. Nech R_x, R_y, R_z sú počty vrcholov, $\Delta x, \Delta y, \Delta z$ sú rozpätia vrcholov v jednotlivých smeroch gridu a $(x_0, y_0. z_0)$ sú súradnice ľavého dolného predného vrcholu. Potom môžeme jednotlivé spôsoby reprezentácie vyjadriť:

- zoznam vrcholov explicitne: *RX*RY*RZ* vrcholov indexovaných cez $(i, j, k); 0 \le i < R_x, 0 \le j < R_y, 0 \le k < R_z$
- zoznam vrcholov implicitne: $\forall(i, j, k): vrchol(x_0 + i\Delta x, y_0 + j\Delta y, z_0 + \Delta z); 0 \le i < R_x, 0 \le j < R_y, 0 \le k < R_z$
- globálny index vrcholu: $I = i + jR_x + kR_xR_y$, kde $i = I \bmod R_x, j = (I \operatorname{div} R_x) \bmod R_y$, $k = (I \operatorname{div} R_x) \operatorname{div} R_y;\ 0 \le I < R_xR_yR_z$
- zoznam buniek explicitne (v danom prípade to budú kocky): $(N_x-1)*(N_y-1)*(N_z-1)$ buniek indexovaných cez $(i, j, k); 0 \le i < N_x - 1, 0 \le j < N_y - 1, 0 \le k < N_z - 1$

- zoznam buniek implicitne (v danom prípade to budú kocky):

$$\forall(i,j,k): bunka\left\{\begin{matrix}(x_i,y_j,z_k),(x_i+\Delta x,y_j,z_k),(x_i+\Delta x,y_j+\Delta y,z_k),(x_i,y_j+\Delta y,z_k),\\(x_i,y_j,z_k+\Delta z),(x_i+\Delta x,y_j,z_k+\Delta z),(x_i+\Delta x,y_j+\Delta y,z_k+\Delta z)\end{matrix}\right\}$$

$0 \le i < R_x, 0 \le j < R_y, 0 \le k < R_z$

- globálny index bunky: $I = i + j(R_x - 1) + k(Rx-1)(Ry-1)$, kde $i = I \bmod (R_x-1)$, $j = (I \operatorname{div} (R_x-1)) \bmod (R_y-1)$, $k = (I \operatorname{div} (R_x-1)) \operatorname{div} (R_y-1)$;

$0 \le I < (R_x-1)(R_y-1)(R_z-1)$

- zoznam hrán: $(r,s); V_r(i,j,k), V_s(l,m,n), 0 \le i,l < R_x, 0 \le j,m < R_y, 0 \le k,n < R_z,$ $(i,j,k) \ne (l,m,n) \wedge abs(i-l)+abs(j-m)+abs(k-n)=1$
- zoznam plôšok (vyjadrujeme hranicu): budú to indexy (r,s,t,u) tých vrcholov, ktoré všetky ležia na jednej zo „stien“ a zároveň súčet počtu hrán medzi každými dvoma z nich je 4 (presná matematická formulácia presahuje rámec tohto textu)

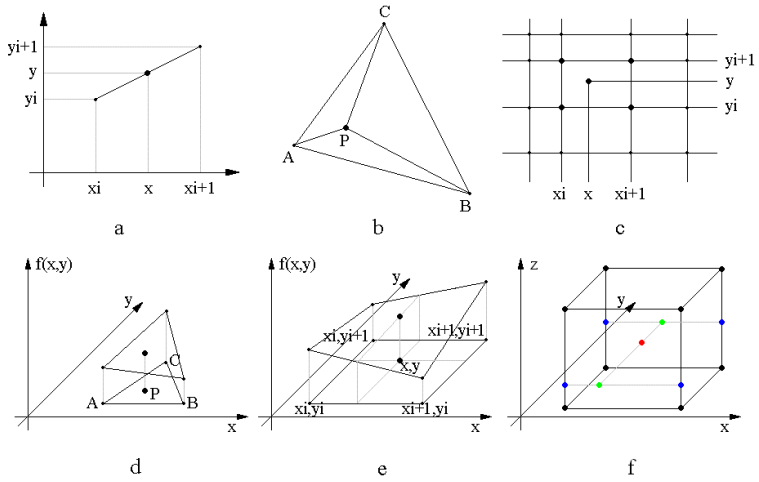
Neštruktúrovaný 3D mesh, ktorý neskôr využijeme pri práci s rozhraním, možno reprezentovať nasledovne:

- zoznam vrcholov: $(x_0, y_0, z_0), (x_1, y_1, z_1), \ldots, (x_{n-1}, y_{n-1}, z_{n-1})$; kde n je počet vrcholov
- zoznam buniek (v tomto prípade plôšok): $(index_{Vi}, index_{Vj}, index_{Vk}), \ldots$; kde V_i, V_j, V_k sú vrcholy

2.3.3 Interpolácia

Často sa stáva, že objekty alebo javy, ktoré chceme vizualizovať sú spojité a naše dáta sú dané diskrétne, len v určitých bodoch (napríklad máme vizualizovať teplotu na danom území, viď. obr.2, pričom k dispozícii máme len teploty v určitých bodoch). Riešením, ako chýbajúce časti dopočítať, je *interpolácia*, ktorá nám umožňuje vyjadriť hodnotu ľubovoľného bodu objektu pomocou hodnôt susedných bodov. Techniky interpolácie môžeme rozdeliť do dvoch hlavných skupín. Sú to metódy globálne, ktoré pracujú s objektom ako celkom a lokálne, ktoré pracujú na bunkovej úrovni. Globálne metódy vo všeobecnosti lepšie aproximujú daný objekt, sú však zložitejšie a pomalšie ako lokálne metódy. Patria medzi ne napríklad Hermitova, Beziérova, alebo interpolácia pomocou splajnov. Detaily nájde čitateľ v [Aken02]. V našom prípade si však vystačíme aj s lokálnymi technikami interpolácie, ktoré sú väčšinou už implementované v hardwari grafickej karty, teda ich za nás automaticky robí počítač. Patria medzi ne napríklad lineárna, barycentrická, bilineárna alebo trilineárna interpolácia.

Pri lineárnej interpolácii (obr.13a) má pre dané body $(x_0, y_0), \ldots, (x_n, y_n)$ v rovine hľadaný bod $x_i \le x \le x_{i+1}$ funkčnú hodnotu $f(x) = (1-u)y_i + uy_{i+1}$, kde $u = \frac{x - x_i}{x_{i+1} - x_i} \in \langle 0,1 \rangle$. Táto technika sa však v praxi používa len zriedka, pretože len veľmi hrubo aproximuje daný objekt.



Obr.13: Rôzne typy interpolácií

Pri barycentrickej interpolácii (obr.13b) pre dané body *A,B,C* v rovine hľadáme vyjadrenie bodu *P* v tvare $P = aA + bB + cC$, kde $a + b + c = 1 \wedge a, b, c \in \langle 0,1 \rangle$. Koeficienty a, b, c nazývame aj barycentrické súradnice bodu *P* a pokiaľ by bol niektorý z nich záporný alebo >1, bod *P* by ležal mimo trojuholníka *ABC*. Platia pre ne nasledujúce vzťahy:

$$a = \frac{d(P,B,C)}{d(A,B,C)}, b = \frac{d(A,P,C)}{d(A,B,C)}, c = \frac{d(A,B,P)}{d(A,B,C)}, kde\ d(U,V,W) = \det\begin{pmatrix} U_x & U_y & 1 \\ V_x & V_y & 1 \\ W_x & W_y & 1 \end{pmatrix}.$$ Techniku možno použiť aj v prípade, že pracujeme s 3D meshom s trojuholníkovou doménou (obr.13d), pričom platí nasledujúci vzťah: $f(P) = af(A) + bf(B) + cf(C)$, kde funkcia *f* môže udávať napríklad farbu alebo výšku (hĺbku). Barycentrická interpolácia sa v počítačovej grafike využíva najmä pri vyfarbovaní trojuholníkov, vyjadrovaní *normál* a mapovaní *textúr*.

Bilineárna interpolácia pre dané body $(x_0, y_0), \ldots, (x_n, y_n)$ obdĺžnikovej domény využíva na výpočet hodnoty $f(x, y)$; $x_i \le x \le x_{i+1}$, $y_i \le y \le y_{i+1}$ funkčné hodnoty jeho 4 susedov (obr.13c, 13e) podľa nasledujúceho vzťahu:

$$f(x,y)=(1-v)\left[(1-u)f(x_i,y_j)+uf(x_{i+1},y_j)\right]+v\left[(1-u)f(x_i,y_{j+1})+uf(x_{i+1},y_{j+1})\right]; kde\, u=\frac{x-x_i}{x_{i+1}-x_i},$$

$$v=\frac{y-y_i}{y_{i+1}-y_i}, u,v\in\langle 0,1\rangle$$

Využíva sa najmä pri vykresľovaní a vyfarbovaní štvoruholníkov, mapovaní textúr, a pri zmenách atribútov rastrových objektov.

Trilineárna interpolácia funguje analogicky ako bilineárna, len pracujeme v 3D priestore. Pre dané body $(x_0, y_0, z_0), ..., (x_n, y_n, z_n)$ 3D obdĺžnikového gridu využíva na výpočet polohy bodu (x, y, z); $x_i \le x \le x_{i+1}$, $y_i \le y \le y_{i+1}$, $z_i \le z \le z_{i+1}$ polohu jeho 8 susedov (obr.13f). Výsledná trilineárna interpolácia je zložením troch lineámych interpolácií v každom smere (x, y, z) v ľubovoľnom poradí. V našom prípade sme najprv lineárne interpolovali modré body v smere *z*, následne z nich zelené body v smere *x* a napokon sme dostali výsledný červený bod interpoláciou v smere *y*. Podrobnejšie informácie o interpolačných technikách nájde čitateľ v [Aken02].

2.3.4 Rendering

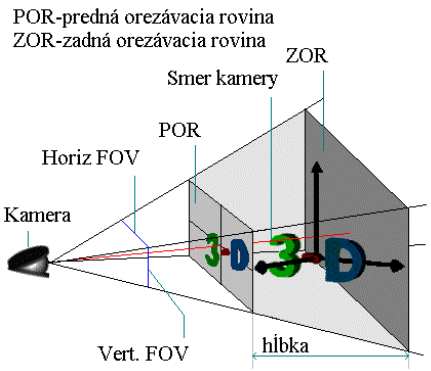
Vytvorené geometrické štruktúry predstavujú geometrický popis vizualizovaných dát a teda ich môžeme zobraziť v obrazovom priestore. Dimenzia tohto priestoru priamo závisí od typu (dimenzie) zobrazovaných dát, podrobnosti sú uvedené v tab.4.

Dim dát	Bunka	Mesh	Výsledok	Obraz. dim
0D	Bod	-	Body	1D,2D,3D
1D	Úsečka, strana, hrana	Lomená čiara, mnohouholník	Body, čiary	2D,3D
2D	Trojuholník, obdĺžnik	2D mesh	Body, čiary plôšky	2D,3D
3D	Štvorsten, hranol	3D mesh	Body, čiary, plôšky, *voxely*	3D

Tab.4: Proces mapovania vizualizačných dát

V počítačovej grafike sa najčastejšie stretávame s 2D a 3D vizualizáciami, teda dimenzia obrazového priestoru je 2 alebo 3.

Zjednotenie štruktúr v obrazovom priestore nazývame aj scénou. Na získanie výsledných obrázkov, alebo animácií je potrebné previesť *rendering scény*, teda projekciu z obrazového priestoru do priestoru obrazovky. V prípade 2D obrazového priestoru ide o jednoduchý proces, kde si užívateľ zvolí *pohľadové okno*, (obdĺžnik, ktorý vymedzí časť obrazového priestoru), obsah ktorého sa zobrazí na obrazovku. Pokiaľ rozmery okna nesúhlasia s rozmermi výsledného obrazu, použijú sa metódy interpolácie. 3D prípad je trochu zložitejší (obr.14). Užívateľ si musí zvoliť *pohľadový ihlan*, teda polohu, smer, šírku záberu kamery (*FOV – field of view*), minimálnu a maximálnu zobrazovaciu vzdialenosť (*POR, ZOR*). Pohľadový ihlan (*frustrum*) sa spolu s ostatnými parametrami (ako napríklad farby, materiály primitív, poloha a vlastnosti svetiel v scéne) použije pri výpočte projekcie scény. Výsledok sa prípadne upraví na požadovanú veľkosť. Algoritmy počítačovej grafiky, ktoré sa využívajú pri procese renderingu sú podrobne popísané v [Chalmo01] a v [Aken02].



Obr.14: Projekcia 3D do 2D

Rendering scény, ako aj iné algoritmy a metódy použité v procese vizualizácie sú už priamo implementované v grafickom hardwari. Ovládanie funkcií hardwaru je však realizované pomocou jazyka nižšej úrovne, čo zabezpečuje rýchlosť a efektivitu, ale na druhej strane to neprispieva k užívateľskej prítulnosti. Preto boli vyvinuté softwarové rozhrania na komunikáciu s hardwarom, grafické knižnice, ktoré umožňujú využívať všetky funkcie hardwaru v štandardných programovacích jazykoch. Medzi najznámejšie patria OpenGL a Microsoft ® DirectX ®, podrobnejšie sa im budeme venovať v kapitole 3.4.

3 Prehľad problematiky

3.1 Úvod

V tejto kapitole predvedieme viaceré prístupy k metódam vizualizácie seizmického vlnenia, porovnáme ich a vyberieme tie, ktoré sa budú svojimi vlastnosťami najviac vyhovovať našim požiadavkám. V tomto smere si budeme najviac všímať tieto vlastnosti metód:

- **Názornosť –** Uvedená vlastnosť je dominantným kritériom výberu metódy. Použitá metóda vizualizácie musí byť dostatočne názorná, výstup musí byť na prvý pohľad čitateľný a pochopiteľný.
- **Výpočtová nenáročnosť** –Metóda musí byť dostatočne rýchla, pretože aplikácia musí pracovať v rozumnom čase aj na bežne dostupných osobných počítačoch.
- **Jednoduchá implementovateľnosť** – Metóda musí byť jednoducho a efektívne implementovateľná na počítači.
- **Presnosť** – Ovplyvňuje kvalitu výslednej vizualizácie. Presnosť je priamo úmerná výpočtovej náročnosti, preto budeme voliť výslednú presnosť s ohľadom na rozumný čas výpočtu.
- **Modifikovateľnosť** – Možnosť meniť parametre metódy užívateľom.

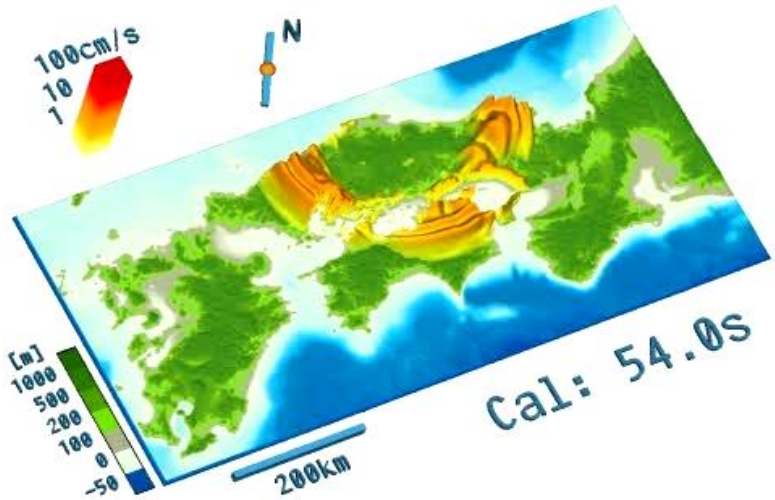
V ďalšej časti uvedieme vlastnosti a porovnáme dve najpoužívanejšie systémy grafických knižníc, OpenGL a Microsoft ® DirectX ®, z ktorých si vyberieme jednu, z ktorou bude naša aplikácia pracovať.

3.2 Spôsoby vizualizácie seizmického vlnenia

V praxi poznáme mnoho metód na vizualizáciu vlnových polí. Môžeme ich rozdeliť do dvoch hlavných skupín.

- **Metódy povrchovej vizualizácie** – Samotný proces vizualizácie dát prebieha len na vopred zvolenom priereze terénu, v rovine.
- **Metódy objemovej vizualizácie** – Vizualizácia prebieha vo vnútri terénu, v celom objeme.

Typickým príkladom metódy povrchovej vizualizácie je metóda, ktorú použil v roku 2003 T. Furumura, detaily v [Furu03]. Článok sa venuje pozorovaniu a vizualizácii zemetrasenia v Tottori-ken Seibu v Japonsku v roku 2000. Autori vytvorili 3D simuláciu propagácie seizmických vĺn s použitím detailnej pod–povrchovej štruktúry a vhodného modelu zemetrasenia, typického pre túto oblasť. Získanú numerickú simuláciu potom porovnali s nameranými hodnotami z terénu. Čo sa týka matematických metód, autori použili hybridnú FPSM/FDM metódu, pričom FPSM (Fourierova pseudo spektrálna metóda) bola použitá v horizontálnom smere a štandardná FDM vo vertikálnom smere s väčšou presnosťou (s jemnejším rozdelením). Dôvodom použitia hybridnej metódy bolo minimalizovať množstvo správ potrebných pri paralelnom výpočte na viacerých počítačoch, čo umožnilo výrazne skrátiť výpočtový čas. Celá simulácia pokrývala oblasť 819x409x167 km, ktorá pozostávala z 5 geo–morfologických vrstiev. Model oblasti bol diskretizovaný, pričom krok v horizontálnom smere činil 1,6 km a krok vo vertikálnom smere 0,8 km. Autori počítali so všetkými spomínanými druhmi seizmických vĺn (P, S, L, R).Výpočet bol realizovaný na *clustri* štyroch 2,2 GHz Intel Xeon počítačov spojených *Gigabitovým Ethernetom*. Proces spotreboval počas behu 5 GB pamäte a na výpočet 6000 časových okamihov potreboval 20hodín. Jeden časový snímok ilustruje obr.15.



Obr.15: Povrchová vizualizácia, prevzaté z [Furu03]

Z uvedeného obrázku je vidieť, že autori zobrazili terén ako rovinu, pričom farebne odlíšili výšku, resp. hĺbku v jednotlivých bodoch. Propagáciu vĺn prostredím znázornili ako vychýlenie bodov terénu smerom nahor o veľkosť amplitúdy zemetrasenia v danom bode a danom časovom okamihu. Veľkosť vychýlenia je tiež rozlíšená odtieňmi farieb. Tento

spôsob vizualizácie nás ale obmedzuje na pozorovanie zemetrasenia len na povrchu, pričom nemáme žiadnu možnosť pozorovať dianie pod povrchom. Na druhej strane je však popísaná metóda pomerne rýchla a pri rozumných obmedzeniach je vhodná na implementáciu na osobnom počítači. Medzi rozumné obmedzenia rátame zníženie rozlíšenia terénu (zníženie presnosti) alebo zjednodušenie matematického modelu (napr. budeme uvažovať len P a S vlny).

Jednoduchou modifikáciou hore uvedenej metódy vieme získať ďalšiu metódu, ktorej hlavnou črtou bude chvenie terénu. Matematická metóda pre nás momentálne nie je podstatná, zameriame sa len na spôsob vizualizácie. Uvažujeme ľubovoľnú matematickú metódu, ktorej výstupom je súbor amplitúd vychýlení v jednotlivých bodoch terénu. Pokiaľ bude v danom bode amplitúda ***a*** nenulová, tak k nej pripočítame resp. odpočítame hodnotu ***f(a,t)***, čo je vlastne funkcia amplitúdy a času a až potom ju nanesieme do daného bodu (ako v predchádzajúcej metóde). Funkcia ***f()*** je periodická funkcia, ktorá zabezpečí efekt chvenia terénu, teda daná oblasť terénu sa bude chvieť priamo úmerne veľkosti amplitúd v bodoch oblasti. Môžeme ju zvoliť napríklad:

$$f(a)=a*\sin(t).$$

Pri použití uvedenej funkcie ***f()*** sa bude k danému bodu v závislosti od času pripočítavať hodnota z intervalu <-a, a>, čo indukuje značné výkyvy. Pre zjemnenie chvenia a zvýšenie reálnosti môžeme funkciu ***f()*** podľa potreby upraviť.

Z uvedeného teda vyplýva, že metódy povrchovej vizualizácie používajú spravidla moduláciu povrchu terénu, čo prináša isté výhody a nevýhody. Medzi hlavné výhody patrí nesporne rýchlosť, keďže zobrazovanie je viazané len na povrchovú časť (čo samozrejme neplatí o matematickom modeli, ktorý je potrebné počítať v celom objeme, ale to nás momentálne nezaujíma). Nevýhodou je už spomínané obmedzenie sa na povrch a nemožnosť vidieť priebeh pod povrchom v celom objeme domény.

Ďalšou metódou, ktorú si spomenieme, bude metóda objemovej vizualizácie, prezentovaná v [Biel00]. Ide o metódu v rámci *Quake Project* (Projekt Zemetrasenie), ktorý je zameraný na simulácie a predpovede silných pohybov zeme pri zemetraseniach v panvách pri mestách Los Angeles a San Francisco. Tieto lokality neboli vybrané náhodne. Sú to totiž najviac osídlené seizmicky aktívne oblasti v USA, sú známe ich geologické zloženia vrátane systémov zlomov a k dispozícii je mnoho zdokumentovaných miestnych zemetrasení, ktoré sú

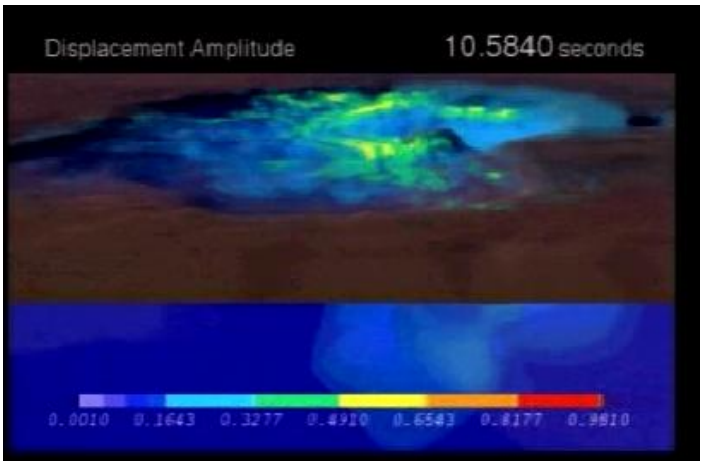
vhodné na porovnávanie s numerickými simuláciami. Získané poznatky sa využívajú najmä pri navrhovaní nových, ale aj pri opravách starších objektov a stavieb, u ktorých sa kladie dôraz na bezpečnosť, vzhľadom na vysoké riziko seizmických otrasov. V našom konkrétnom prípade autori vytvorili 40 sekundovú 3D simuláciu zemetrasenia *Northridge* v doline San Francisca v roku 1994. Simulácia pokrýva oblasť 50x50x10 km, čo činí po diskretizácii vyše 17mil. bodov a vyše 90mil. *tetrahedrálnych* elementov. Výsledný *mesh* poskytuje frekvenčné rozlíšenie 1 Hz a priestorové 20 m. Matematický výpočet bol realizovaný kombináciou metód, metódou FEM v priestorovej oblasti a FDM v časovej oblasti. Simulácia bola realizovaná na 256-procesorovom superpočítači Cray T3E-900 v superpočítačovom centre v Pittsburghu. Vyžiadala si 24 GB pamäte a 4 hodiny výpočtového času na spracovanie 6000. časových okamihov.

Celkový pracovný postup simulácie môžeme zhrnúť do nasledujúcich bodov.

1. **Generovanie *multiresolution mesh*u** (viď nižšie) zo vstupných dát, ktorý pokrýva všetky vlnové dĺžky (samozrejme s danou presnosťou).
2. **Rozdelenie meshu do subdomén**, ktoré prislúchajú jednotlivým procesorom tak, aby sa minimalizovalo množstvo správ potrebných na komunikáciu medzi procesormi.
3. **Diskretizácia vlnových rovníc** prechádzajúcich vĺn s použitím FEM.
4. **Integrácia výsledného semidiskrétneho systému** diferenciálnych rovníc v čase s použitím FDM.
5. **Vizualizácia** a konečné úpravy výslednej propagácie vĺn.

Krok číslo 1 možno chápať ako fázu predspracovania, ktorá nevyžaduje paralelný počítač. Autori článku vo svojich metódach používajú tzv. *multiresolution mesh*, ktorý môžeme v našej terminológii prirovnať k obdĺžnikovému gridu (na rozdiel od uniformného gridu nemá všade rovnaké rozlíšenie, teda diskretizačný krok sa môže meniť). Spomínané meshe sú v prípade veľkých paniev zložených najmä zo sedimentov (o ktorých sa stále bavíme), v ktorých sa môže vlnová dĺžka seizmických vĺn značne líšiť v jednotlivých subdoménach celej domény, veľmi výhodné a značne redukujú počet bodov meshu, pretože veľkosti elementov môžeme prispôsobiť vlnovým dĺžkam propagujúcich vĺn v danej časti demény. Tým získame samozrejme väčšiu presnosť a na dôvažok si môžeme dovoliť aj väčšie kroky v čase Δt pri nezmenenej stabilite metódy v 3. kroku. Kroky 2, 3 a 4 vyžadujú už samotné paralelné spracovanie, ktorého výstupom je súbor vektorov posunutí a vektorov pôsobiacich síl

v každom bode meshu v každom 30. časovom okamihu. Tieto dáta sa následne vizualizujú a upravujú do konečnej podoby v 5. kroku. Výsledok simulácie ilustruje obr.16.



Obr.16: Objemová vizualizácia, prevzaté z [Biel00]

Uvedená metóda objemovej vizualizácie zobrazuje terén ako nádobu tvaru kvádra, v ktorej sú body s rozdielnou charakteristikou farebne odlíšené. Samotná simulácia prebieha v celom objeme nádoby, pričom jednotlivým tetrahedrálnym elementom je pridelená farba podľa veľkosti amplitúdy seizmického vlnenia v danom bode a danom časovom okamihu. Vzniká tým hmlový efekt, ktorý efektívne a názorne popisuje priebeh zemetrasenia v danom objeme terénu. Metódu nie je problém modifikovať, aby umožnila užívateľovi upravovať objemovú nádobu reznými rovinami tak, aby mohol sledovať priebeh simulácie na prierezoch. Rýchlosť metódy vizualizácie závisí hlavne na veľkosti vstupných dát a tiež na presnosti, s ktorou chceme celú simuláciu počítať. Pri rozumnej voľbe spomínaných parametrov bude možné metódu úspešne implementovať aj na osobnom počítači.

Všetky metódy spomínané v tejto kapitole svojím spôsobom vizualizovali propagáciu seizmických vĺn cez dané médium. Po zvážení výhod a nevýhod jednotlivých metód sa ďalej budeme zaoberať objemovou vizualizáciou ilustrovanou na príklade vyššie, keďže najviac korešponduje so zadaním diplomovej práce.

3.3 Objemová vizualizácia

Vo všeobecnosti k problému objemovej vizualizácie existujú 3 prístupy:

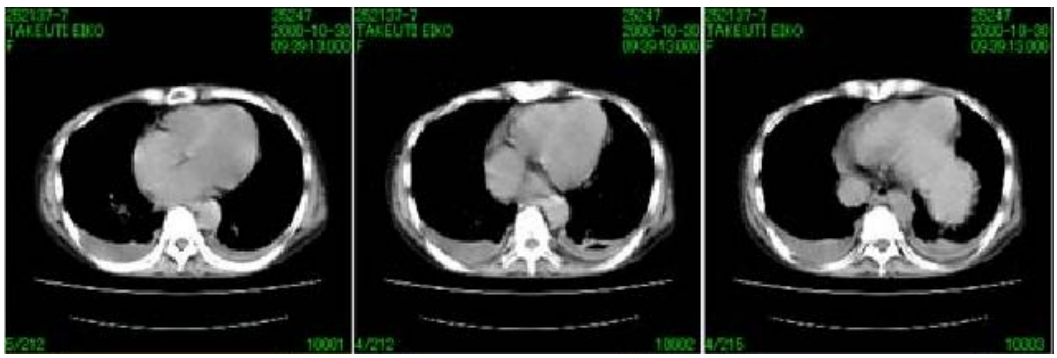
- Vizualizácia objemu pomocou 2D rezov (2D prístup)
  - s rezmi kolmými na súradnicové osi.

- so šikmými rezmi.

- Nepriama objemová vizualizácia, reprezentácia pomocou povrchov (iso-povrchy), z matematického hľadiska ide o redukciu problému do 2D.
  - Opaque-cube algoritmus
  - Marching-cube algoritmus
  - Marching-tetrahedra algoritmus
- Priama objemová vizualizácia, zobrazujeme reálne 3D objekty, ktoré vyžarujú svetlo a každý z nich má definovanú priesvitnosť (transparentnosť).
  - Ray-casting algoritmus
  - Textúry

3.3.1 Vizualizácia 2D rezov

Uvažujme rezy objemových dát rovnobežnými rovinami kolmými na niektorú zo súradnicových osí *x, y, z*, napríklad rovinami z=-10, z=-9, ... , z=0. Na začiatku zobrazíme, napríklad rez rovinou z=0 a dovolíme užívateľovi interaktívne meniť z-ovú súradnicu, teda pohybovať rovinou rezu. Vizualizácia jednotlivých rezov je v tomto prípade pomerne jednoduchá, stačí zobraziť príslušnú hladinu buniek. Na zvýraznenie môžeme použiť funkciu farby alebo výškovú funkciu. Príklad výsledných obrázkov je uvedený na obr.17.



Obr.17: Obrázky rezov tomografických dát

V prípade, že použijeme roviny, ktoré nie sú kolmé na žiadnu zo súradnicových osí, je problém o niečo zložitejší. Na určenie, ktoré bunky sa budú vykresľovať je možné použiť napríklad algoritmus 3D rasterizácie ([Shirley02]) alebo jednoduché porovnávanie vzdialeností od roviny orezávania.

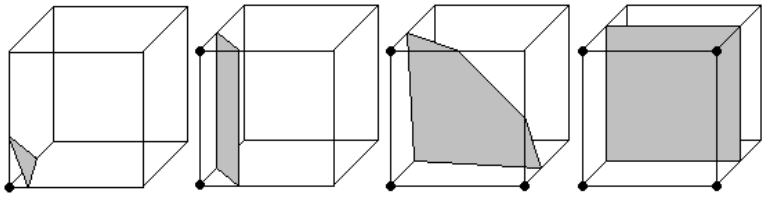
Obidva uvedené postupy vizualizácie 2D rezov sú použiteľné aj v 3D prípade (pri orezávaní) a budú implementované v našej aplikácii, podrobnosti uvedieme v kapitolách 4.3 a 5.3.4.

3.3.2 Nepriama objemová vizualizácia

Pri nepriamej objemovej vizualizácii reprezentujeme dané objemové dáta povrchovo pomocou aproximácií iso–povrchov (isosurfaces). Nech $f(x, y, z)$ je „rozumná“ (diferencovateľná v každom bode, s nemiznúcimi deriváciami) skalárna funkcia, potom iso–povrchmi nazývame hladiny funkcie $f :\{(x, y, z)| f(x, y, z)=c\}$. Na vytváranie aproximácií iso–povrchov môžeme použiť napríklad algoritmy Opaque–cube, Marching–cube alebo Marching–tetrahedra. Prvé dva spomenuté algoritmy fungujú len na štruktúrovaných uniformných gridoch, posledný možno použiť na ľubovoľný mesh.

Myšlienka algoritmu Opaque–cube spočíva vo vyhľadávaní všetkých buniek gridu (v tomto prípade ide o kocky), ktoré pretína daný iso–povrch $f(x, y, z)=c$ nasledujúcim spôsobom. Program prechádza cez všetky vrcholy gridu (i, j, k) a zisťuje hodnotu $f(i, j, k)$. Pokiaľ je $f(i, j, k)>c$, označí príslušný vrchol znamienkom +, inak ho označí znamienkom –. Bunky, ktoré majú vrcholy označené znamienkom + aj – sa vykreslia. Metóda poskytuje len veľmi hrubú reprezentáciu iso–povrchu, keďže jednotlivé bunky buď sú, alebo nie sú viditeľné. Na zlepšenie výsledku je možné originálne bunky rozdeliť na menšie časti (príslušné hodnoty sa dopočítajú interpoláciou hodnôt vrcholov materskej bunky), ktoré sa použijú pri výpočte namiesto materských buniek.

V prípade algoritmu Marching–cube postupujeme podobne ako pri Opaque–cube, označíme všetky vrcholy znamienkami + a –. Pre každú bunku (kocku) teda dostávame 8–bitový vektor pozostávajúci zo znamienok jednotlivých vrcholov, čo predstavuje 256 možností prieniku s iso–povrchom. Mnohé z nich sú však symetrické, čo redukuje počet možností na 15. V ďalšom, v závislosti na susedných bunkách, nám ešte pribudne zopár špeciálnych možností. Niektoré z možností prieniku ilustruje obr.18. Zvýraznené vrcholy majú znamienko + , zvyšné –. Sivé oblasti sú príspevkami danej bunky k výslednému povrchu (po triangulácii).



Obr.18: 4 z 15 možných prípadov prieniku pri algoritme Marching-cube

Problémom je začleniť susediace bunky do kategórií špeciálnych prípadov tak, aby v iso–povrchu nevznikali diery, čo niekedy nie je možné. Vo všeobecnosti môžeme vlastnosti algoritmu Marching–cube zhrnúť:

- Generuje pomerne veľa trojuholníkov.
- Generuje veľmi veľké a naopak veľmi malé trojuholníky.
- Rozlišuje veľa špeciálnych prípadov.
- V niektorých prípadoch generuje neošetriteľné diery.

Vstupom algoritmu Marching–tetrahedra môžu byť ľubovoľné (neštruktúrované) meshe. Bunkami takýchto meshov sú štvorsteny, ktoré algoritmus spracováva podobne ako predchádzajúce dva algoritmy, teda označí všetky vrcholy znamienkami + a –. Rozdiel spočíva v počte možností prieniku. Kým pri predchádzajúcom algoritme ich bolo 15, pri štvorstenoch nastávajú len 2 netriviálne možnosti (po odčítaní symetrií):

- Jeden vrchol + a dva – (opačný prípad analogicky), priesečníkom je trojuholník.
- Dva vrcholy + a dva –, priesečníkom je štvoruholník, ktorý po triangulácii pomocou kratšej uhlopriečky dáva dva trojuholníky.

Vlastnosti algoritmu Marching–tetrahedra možno zhrnúť:

- Tiež generuje veľa trojuholníkov rôznych veľkostí.
- Rozlišuje menej špeciálnych prípadov ako Merching–cube.
- Nevznikajú problémy s konzistenciou susedných buniek.
- Výsledkom je hrubšia reprezentácia ako pri Marching–cube.

Podrobnejšie sa s metódami nepriamej objemovej vizualizácie čitateľ oboznámi v [Ning93]. Celkovo môžeme medzi výhody metód nepriameho prístupu zaradiť:

- Jasný a prehľadný výstup.
- Metódy využívajú možnosti štandardného grafického hardwaru, keďže výstupom sú trojuholníky.

Naopak medzi nevýhody patria:

- Je potrebná segmentácia objemových dát, čo môže byť v niektorých prípadoch zložité a náročné na čas výpočtu.
- Problémy pri pozvoľna sa meniacich dátach.
- Pri komplexných scénach metódy generujú príliš veľa trojuholníkov.
- Strata informácie v porovnaní s originálnymi objemovými dátami.

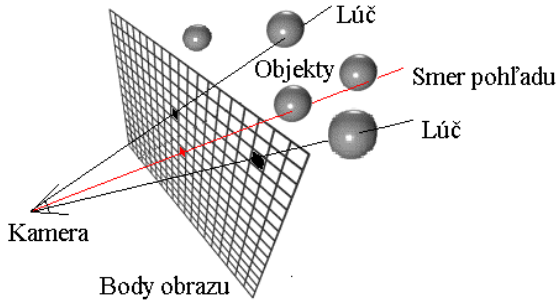
Uvedenými metódami teda nie je možné efektívne zobrazovať naše dáta do podoby hmlového efektu.

3.3.3 Priama objemová vizualizácia

Na rozdiel od nepriamych metód, v priamych metódach zobrazujeme celé vstupné dáta, s kompletnou vonkajšou a vnútornou štruktúrou, teda nestrácame žiadne informácie. Každá bunka je považovaná za polo–transparentný objekt (s určitou transparentnosťou) vyžarujúci svetlo (s určitou intenzitou). Jednotlivé metódy riešenia sú založené na zákonoch fyziky, ako vyžarovanie, absorpcia, rozptyl, odraz alebo lom, preto je týmto spôsobom možné vizualizovať aj javy spojené s plynmi, teda aj nami požadovanú hmlu. Nevýhodami techník priamej objemovej vizualizácie sú veľká pamäťová a výpočtová náročnosť, ktorá sa dá znížiť špeciálnymi algoritmami a špeciálnym hardware. Na riešenie možno použiť 2 hlavné stratégie, objektovo orientovaný prístup a obrazovo orientovaný prístup. V prvom prípade prechádzame všetkými bunkami (voxelmi) objemových dát, pre každý voxel určíme príslušné body (pixely) obrazového priestoru, ku ktorým prispieva svojou projekciou a vypočítame príspevky k intenzitám (teda k farbe) jednotlivých pixelov. Myšlienka druhej stratégie je presne opačná, ku každému pixelu obrazového priestoru určíme voxely objektového priestoru, ktoré ho svojou projekciou ovplyvňujú. Následne vypočítame farebnú hodnotu pixelu z hodnôt intenzít a transparentností príslušných voxelov. Keďže obidve stratégie sú pamäťovo a výpočtovo veľmi náročné, výber závisí hlavne od možností optimalizácie a od konkrétnych vstupných dát. V našej aplikácii sme si zvolili stratégiu obrazového prístupu.

3.3.4 Ray–casting

Najpožívanejšou technikou pri priamej objemovej vizualizácii je metóda ***Ray-castingu*** (vysielania lúčov), ktorej princíp ilustruje obr.19.



Obr.19: Ray–casting

Na začiatku si určíme polohu kamery, smer pohľadu a veľkosť výsledného obrazu. Každým bodom obrazu následne vedieme lúč, pričom jeho zdrojom je kamera a smer vypočítame pomocou smeru pohľadu a polohy bodu v obrazovej mriežke. V ďalšom zisťujeme prieniky objektov v scéne (v našom prípade sú to voxely) s lúčom, resp. polpriamkou, ktorá predstavuje lúč. Po zistení všetkých prienikov daného lúča s voxelmi v scéne si zoradíme príslušné voxely, ktorým patria prieniky od najvzdialenejšieho od kamery k najbližšiemu.V tomto momente máme 3 problémy ([Luber00]):

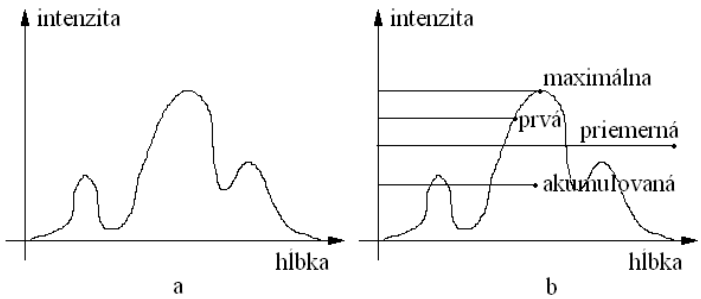
1. Priradiť voxelom hodnoty intenzity vyžarovania a transparentnosti v závislosti od vstupných dát a voľby užívateľa.
2. Navrhnúť spôsob vzorkovania intenzít pozdĺž lúča, teda vybrať voxely, ktoré budú prispievať svojimi intenzitami do výslednej intenzity.
3. Navrhnúť spôsob, akým určíme výslednú intenzitu (farbu) obrazového bodu z čiastkových hodnôt intenzity a transparentnosti zasiahnutých objektov.

Prideľovanie transparentností jednotlivým voxelom závisí od spôsobu, akým chceme dáta vizualizovať. Pokiaľ chceme zobraziť všetky voxely s rovnakou mierou, nastavíme všetky transparentnosti na rovnakú hodnotu, ktorá závisí od toho, ako „hlboko“ chceme do objektu vidieť. Pokiaľ chceme zobraziť len časť voxelov s istou hodnotou (napríklad pri vizualizácii ľudského tela chceme vidieť rozhranie kosť–svaly) a zvyšné chceme postupne odfiltrovať (spriehľadniť), môžeme použiť na prideľovanie hodnôt transparentnosti voxelom klasifikáciu pomocou iso–povrchov ([Levoy88]). Intenzity vyžarovania nastavíme buď priamo na základe vstupných dát (funkciou hodnoty voxelu), alebo na zvýšenie kvality výsledného obrazu môžeme použiť metódu lokálneho osvetľovania [Phong75], [Shirley02], pričom za vektor normály použijeme normalizovaný vektor gradientu hodnôt vrcholov gridu, ktorý určuje

voxely. Na výpočet hodnoty gradientu mimo vrcholových bodov použijeme trilineárnu interpoláciu.

Jednou z techník vzorkovania intenzít je metóda presnej integrácie. Najprv nájdeme všetky voxely, ktoré daný lúč pretína. Pre každý nájdený voxel potom zistíme dĺžku lúča, ktorá leží vo vnútri a použijeme ju ako váhu k intenzite daného voxelu. Keďže musíme počítať priesečníky, táto metóda je výpočtovo veľmi náročná. Inou možnosťou je uniformné vzorkovanie, pri ktorom prechádzame pozdĺž lúča konštantným krokom a po každom kroku zoberieme vzorku intenzity príslušného voxelu v ktorom sa práve nachádzame. Metóda je postačujúco presná a v porovnaní s presnou integráciou veľmi rýchla a jednoduchá. Ďalšou možnosťou sú rasterizačné techniky, napríklad 3D verzia Bresenhamovho rasterizačného algoritmu ([Shirley02]), ktoré sú síce rýchle a jednoducho implementovateľné, ale ich výstup nie je vždy presný.

V ďalšom popíšeme spôsoby, ako zo získaných intenzít a transparentností (ďalej len dát) vypočítame výslednú farbu pixelu na obrazovke. Predpokladajme, že voxely z ktorých sme získali dáta boli usporiadané pozdĺž lúča (podľa hĺbky vzostupne alebo zostupne, viď vyššie). Získané intenzity potom možno zakresliť do grafu (obr.20a).

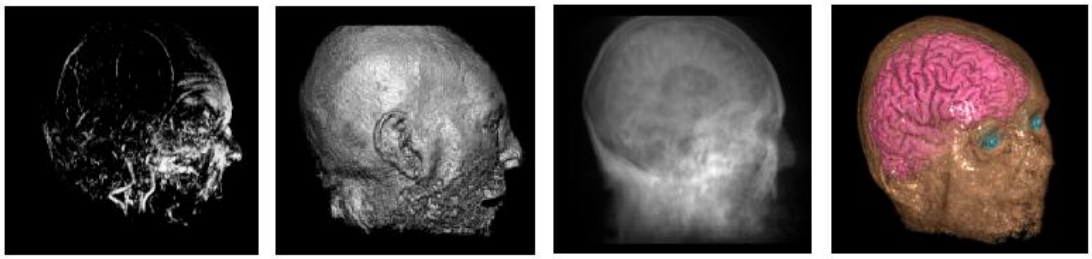


Obr.20: Spôsoby získavania intenzít

Zo získaných intenzít teraz vyberieme tie, ktoré použijeme pri výpočte výslednej (obr.20b). Dĺžka vodorovných úsečiek predstavuje hĺbku, do ktorej prezeráme intenzity. Poznáme viacero metód výberu ([Huang02]):

- Vyberieme lokálne alebo globálne maximum intenzít, obr.21a. Metóda sa používa pri vytváraní angiogramov pomocou magnetickej rezonancie.
- Vyberieme prvú intenzitu z danou hodnotou *I*, obr.21b. Výsledkom je obraz iso–povrchu s funkciou $f(x, y, z)=I$

- Vypočítame priemernú intenzitu spomedzi všetkých, obr.21c. Výsledkom je obraz pripomínajúci RTG snímok.
- Sčítavame intenzity na základe hodnôt transparentnosti, pričom nemusíme prejsť všetkými, podrobnejší popis nižšie. Výsledok závisí od zvolených hodnôt transparentnosti, obr.21d.



Obr.21: Ray–castované obrazy s použitím rôznych metód výberu intenzít, zdroj [Huang02]

Základom metódy je sčítací operátor, ktorý možno vyjadriť ako $I_{out} = (1-a)I_{in} + aI$; kde I_{in} je vstupná intenzita, I_{out} výstupná intenzita, a transparentnosť a I je intenzita daného voxelu. V závislosti od spôsobu prechádzania vstupných voxelov (sú usporiadané zozadu dopredu) môžeme napísať algoritmy metódy:

- prechádzanie zozadu dopredu

```
float Back_To_Front(I0,…,In, a 0,…, a n)
{
      I=I0;
      for(i=1, i<n+1, i++)
            I=(1–a i)I + a iIi;
      return I;
}
```

- prechádzanie spredu dozadu

```
float Front_To_Back(I0,…,In, a 0,…, a n)
{
      I=0;C=1;
      for(i=n, i>0, i--){
            I=I + Iia i*C;
            C=C*(1-a i);
            };
```

```
        I=I + C*I0;
        return I;
}
```

V tomto prípade existuje aj efektívnejší algoritmus, v ktorom nemusíme prejsť všetky intenzity, zastavíme sa v prípade, že hodnota C klesne pod stanovenú hranicu e, napríklad keď je hodnota C blízko 0 (ďalšie voxely by už aj tak neprispievali do výsledku, neboli by viditeľné).

```
float Front_To_Back_Optimized(I0,…,In, a 0,…, a n, e )
{
        I=0;C=1;i=n;
        while(C>e  && i≥0){
                I=I + Ii a i*C;
                C=C*(1- a i);k--;
                };
        if(i<0) I=I + C*I0;
        return I;
}
```

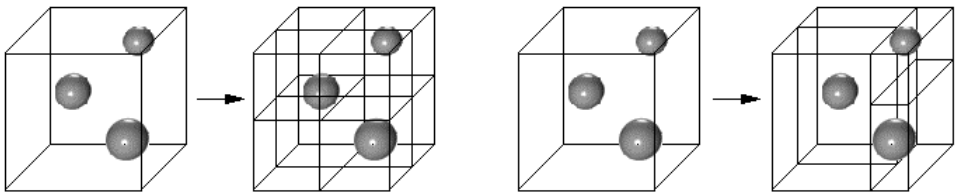
3.3.5 Optimalizácia algoritmu ray–castingu

Algoritmus ray–casting je obrazovo orientovaný, teda pre každý pixel obrazu musíme nájsť všetky objekty v scéne, ktoré ovplyvnia jeho intenzitu (farbu). Hľadanie zahŕňa zisťovanie prieniku lúča vedeného daným pixelom so všetkými objektmi v scéne. Myšlienka optimalizácie spočíva v obmedzení počtu objektov, ktoré treba otestovať na prienik s lúčom. Scénu rozdelíme na bunky jednoduchého tvaru, z ktorými sa jednoducho počíta prienik, napríklad na kocky alebo kvádre. Vzniknuté bunky možno ďalej deliť, pričom delenie ukončíme v prípade, keď sa už v bunke nenachádza žiaden objekt (označíme ju ako prázdnu) alebo len istý malý počet objektov (označíme ju ako plnú). Takto nám vznikne hierarchická štruktúra, ktorú môžeme reprezentovať stromom. Typ a vlastnosti stromu závisia od spôsobu rozdelenia scény. Počítanie prienikov s objektmi scény sa teda redukuje na prechádzanie stromom, pri ktorom zisťujeme prienik s aktuálnou bunkou (jednoduchý výpočet) a v závislosti od výsledku buď skončíme, alebo sa presunieme do špecifickej bunky na nižšiu úroveň. V prípade, že sa dostaneme do plnej bunky na najnižšej úrovni, otestujeme na prienik

všetky objekty nachádzajúce sa v danej bunke. Uvedeným spôsobom sme zredukovali počet testovaných objektov na minimum, čo značne zníži výpočtovú náročnosť metódy ray–castingu. Najviac používanými priestorovými hierarchickými štruktúrami v počítačovej grafike sú *Octree* (Oktálový strom) a *BSP–Tree* (Binary Space Partition, binárne rozdelenie priestoru), [Havran00], [Aken02].

V prípade oktálového stromu (obr.22a) sa scéne opíše kocková obálka s hranou veľkosti 2^n, ktorá tvorí prvú bunku a zároveň koreň stromu. Kocka sa následne rozdelí na 8 zhodných pod–kociek, atď. Delenie ukončíme v prípade prázdnej kocky alebo v prípade dosiahnutia limitu objektov (viď vyššie). Vo výslednom strome má teda každý vrchol 0 alebo 8 nasledovníkov.

V prípade BSP–stromu (obr.22b) sa scéne opíše kvádrová obálka, ktorá tvorí prvú bunku a koreň stromu, pričom nemáme špeciálne požiadavky na veľkosť hrany. Bunka sa následne rozdelí na 2 časti rovinou kolmou na niektorú súradnicovú os (*ortogonálny BSP–strom*), alebo ľubovoľnou rovinou (*polygonálny BSP–strom*). Častejšie sa však používa ortogonálna forma. Delenie ukončíme v prípade prázdnej bunky alebo v prípade dosiahnutia limitu objektov (viď vyššie). Výsledný strom je teda binárny.



Obr.22: Octree a BSP–tree

V našom prípade použijeme štruktúru oktálového stromu, pretože je jednoduchší na implementáciu a výhodnejší z hľadiska objektov, keďže vizualizujeme voxely v tvare kociek.

Podrobnejšie informácie o stromoch octree a BSP ako aj o rôznych iných metódach delenia priestoru nájde čitateľ v [Havran00] a [Aken02].

3.3.6 Objemová vizualizácia pomocou textúr

Nasledujúce metódy objemovej vizualizácie ([Sala03]) môžeme považovať za objektovo orientované, keďže pracujeme so systémom rezov objemového priestoru, z ktorých vytvoríme sadu textúr, ktoré následne mapujeme na polygóny (ktoré samozrejme korešpondujú s rezmi) do obrazového priestoru. Textúry majú formát RGBA, teda je v nich uložená jednak

informácia o intenzite (vo farebných zložkách RGB) a tiež o transparentnosti (A). Pri mapovaní sa automaticky vykonáva kompozícia intenzít pomocou príslušných hodnôt transparentnosti. Väčšinu operácií pritom vykonáva grafický hardware a preto sú tieto metódy veľmi efektívne.

Prvou z textúrových metód je metóda vizualizácie pomocou 2D textúr. Objemové dáta narežeme systémom rezov kolmých na súradnicové osi ***(x, y, z)***, pričom potrebujeme všetky 3 sady rezov, čo spôsobuje veľké pamäťové nároky. Z rezov potom hardwarovou bilineárnou interpoláciou získavame textúry, ktoré použijeme na polygóny pri vykresľovaní. Nevýhodou metódy je rozdielny vzorkovací krok pri vzorkovaní intenzít pri zmene polohy kamery, keďže systémy rezov sa zmenou polohy kamery nemenia. Výsledkom sú nepresné výpočty výsledných intenzít.

Namiesto 2D textúr môžeme použiť aj 3D textúry, ktoré predstavujú objemový textúrový objekt. 3D textúry nanešťastie nie sú podporované všetkými grafickými kartami, preto táto metóda nie je použiteľná na všetkých počítačoch. Podobne ako v predchádzajúcom prípade, objemové dáta narežeme systémom rezov, pričom teraz budú rezy rovnobežné s projekčnou rovinou, teda nemáme problém s rozdielnym vzorkovacím krokom. Keďže máme k dispozícii 3D textúrovú pamäť, stačí nám 1 sada rezov,. Z rezov vytvoríme trilineárnou interpoláciou textúry, ktoré mapujeme na polygóny do obrazového priestoru. Metóda je viazaná na veľkosť 3D textúrovej pamäte a teda pre veľké objemové dáta (rádovo $>512^3$) je jej použitie obmedzené. Podrobnejšie v [Sala03].

3.3.7 Porovnanie

Uviedli sme si 2 prístupy k priamej vizualizácii objemových dát, v nasledujúcom si porovnáme hlavné klady a zápory jednotlivých metód. Údaje sú uvedené v tab.5.

Metóda\Atribút	2D rezy	Iso–povrchy	Ray–casting	2D textúry	3D textúry
Kvalita	+	+	+ + +	+	+ +
Rýchlosť	+ +	+ +	– –	+ +	+
Obmedzenie	Len 2D	Strata info, nevie „hmlu“	Časové nároky	Pamäťové nároky	Špeciálny hardware, pamäťové nároky

Tab.5: Porovnanie metód priamej objemovej vizualizácie

Po zvážení výhod a nevýhod sme si v prípade našej aplikácie vybrali metódu Ray–castingu. Intenzity vyžarovania voxelov určíme z hodnôt vrcholov, pričom intenzitu vo vrcholoch udáva absolútna hodnota amplitúdy výchylky v určenom smere v danom vrchole 3D gridu. Transparentnosti voxelov neuvažujeme, pretože na vizualizáciu jedného typu média postačí aj jedna charakteristika (intenzita). Na výpočet intenzít použijeme metódu uniformného vzorkovania pozdĺž lúča. Získané intenzity postupne sčitujeme, pokiaľ nedosiahneme požadovanú úroveň alebo neprekročíme určenú vzdialenosť od kamery, viď kapitolu 4.3.3.

3.4 Grafické knižnice

V tejto kapitole si spomenieme dve najpoužívanejšie systémy grafických knižníc sprostredkujúce programátorovi všetky funkcie grafického hardwaru. Sú to OpenGL, a Microsoft ® DirectX ®.

3.4.1 OpenGL

OpenGL, momentálne vo verzii 2.0, je užívateľsky prítulné prostredie na vývoj interaktívnych 2D a 3D grafických aplikácií. Od svojho uvedenia v roku 1992 sa stalo svetovo najrozšírenejším a najpodporovanejším grafickým programátorským rozhraním (application programming interface - API) na vývoj 2D a 3D grafických aplikácií na rôznych počítačových platformách. Poskytuje nepreberné možnosti renderingu, mapovania textúr, špeciálnych efektov a rôznych iných vizualizačných funkcií. Výsledkom je vysoká kvalita obrazu a rýchlosť, samozrejme v závislosti od použitého grafického hardwaru. Čo sa týka programovania, OpenGL je skôr procedurálne ako popisné rozhranie. Napríklad na vyrenderovanie červenej gule musí programátor špecifikovať príslušné poradie príkazov na nastavenie kamery, pohľadu, modelovacích transformácií a nakoniec vykresliť guľu pomocou geometrických primitív (napríklad trojuholníkov) v červenej farbe. V iných systémoch (napríklad VRML [VRML]), ktoré sú popisné, stačí špecifikovať typ objektu, súradnice a farbu. Nevýhodou použitia procedurálneho rozhrania na dosiahnutie želaného výsledku je potreba detailne špecifikovať všetky operácie a vykonať ich v presnom poradí. Na druhej strane však takýto prístup umožňuje vysokú flexibilitu pri vytváraní výsledného obrazu. Najsilnejším argumentom pri výbere rozhrania je fakt, že popisné rozhranie môže byť vybudované na procedurálnom ale naopak to neplatí. OpenGL by sa dalo charakterizovať ako

„skladací jazyk“, jednotlivé funkčné kúsky sa môžu skladať a kombinovať ako tehly pri stavaní domu. Ďalším aspektom OpenGL je voľnosť špecifikácie, teda dve rôzne implementácie OpenGL nikdy nie sú úplne identické, inak povedané je málo pravdepodobné, že vyrenderujú rovnaký obraz. Umožňuje to implementovať OpenGL na širokú škálu hardwarových platforiem. Pokiaľ by bola špecifikácia príliš exaktná, obmedzovalo by to použitie OpenGL ako štandardu na rôznych grafických akcelerátoroch. Výhody OpenGL môžeme zhrnúť do nasledujúceho prehľadu:

- Je to priemyselný štandard nezávislý na výrobcovi a platforme.
- Rokmi overená stabilita a spoľahlivosť.
- Portabilnosť; OpenGL aplikácia pobeží na ľubovoľnom grafickom adaptéri s podporou štandardu OpenGL nezávisle na operačnom systéme.
- Stále sa vyvíja a prispôsobuje novým možnostiam grafického hardwaru, pričom nové verzie sú spätne kompatibilné so starými.
- Je prispôsobiteľný na použitie v rôznych systémoch od spotrebnej elektroniky cez osobné počítače až po superpočítače.
- Je jednoduchý na používanie, dobre štruktúrovaný s intuitívnymi príkazmi.
- Je dobre zdokumentovaný; existuje veľa kníh zaoberajúcich sa tematikou OpenGL, tiež je k dispozícii množstvo ukážkového zdrojového kódu.

Podrobnejšie špecifikácie OpenGL nájde čitateľ na [OpenGL], odkiaľ bol čerpaný materiál na tvorbu kapitoly o OpenGL.

3.4.2 Microsoft ® DirectX ®

Microsoft ® DirectX ®, momentálne vo verzii 9.0c, bol uvedený na trh v roku 1995 a je považovaný za štandard pre programovanie multimediálnych aplikácii na platforme Microsoft ® Windows ®. Je to súbor multimediálnych aplikačných programátorských rozhraní (API) zahrnutý v operačných systémoch Microsoft ® Windows ®. Predstavuje štandardnú vývojovú platformu pre osobné počítače s týmto operačným systémom, pričom poskytuje programátorom prístup k špeciálnym funkciám hardwaru bez nutnosti programovania v jazyku nižšej úrovne. Na rozdiel od OpenGL, Microsoft ® DirectX ® okrem grafických zariadení podporuje aj programovanie zvukových kariet (mixovanie a prehrávanie hudby a zvukov), komunikačných zariadení (sieťové karty, modemy) a vstupných zariadení, ako joystickov,

klávesníc a myší. Koncepcia systémového riešenia spočíva v rozdelení funkcionality do viacerých modulov, v závislosti od funkcie a ovládaného zariadenia. V nasledujúcom prehľade uvádzame popis najpoužívanejších modulov aj s popisom.

- Microsoft ® DirectDraw ® - implementuje programovanie 2D grafiky
- Microsoft ® Direct3D ® - implementuje programovanie 3D grafiky
- Microsoft ® DirectSound ® - implementuje prehrávanie hudby
- Microsoft ® DirectMusic ® - implementuje hudobný syntetizátor
- Microsoft ® DirectInput ® - implementuje programovanie vstupných zariadení
- Microsoft ® DirectPlay ® - implementuje programovanie komunikačných zariadení

Systém programovania je veľmi podobný ako u OpenGL, tiež ide o procedurálny typ rozhrania. Medzi výhody Microsoft ® DirectX ® patria:

- Balík je priamo dodávaný s operačným systémom Microsoft ® Windows ®,.
- Je optimalizovaný na operačný systém Microsoft ® Windows ®.
- Možnosť programovať aj iné ako grafické zariadenia.
- Podpora a pravidelné aktualizácie výrobcom.
- Prepracovaná dokumentácia a programátorská podpora prostredníctvom Microsoft ® DirectX ® SDK (software development kit, balík na vývoj softwaru).

Podrobnejšie špecifikácie Microsoft ® DirectX ® nájde čitateľ v [DirectX], odkiaľ bol čerpaný materiál na tvorbu tejto kapitoly.

3.4.3 Porovnanie

Predstavili sme 2 druhy grafických rozhraní, v nasledujúcom ich porovnáme. Čo sa týka funkcionality a možností z hľadiska programovania 2D a 3D grafiky, sú obidva systémy takmer ekvivalentné. Systém programovania, dosahovaná kvalita zobrazenia a výkon sú porovnateľné. Z hľadiska podporovaných zariadení a technológií je výhodnejší Microsoft ® DirectX ®, ktorý predstavuje komplexnejšie riešenie ako OpenGL, v našom prípade nás však zaujíma len programovanie 3D grafiky a preto pre nás tento aspekt nie je rozhodujúci. Na druhej strane, štandard OpenGL je platformovo nezávislý a portabilný na ľubovoľný operačný systém, čo predstavuje istú výhodu, pretože vo vedeckých kruhoch je okrem systému Microsoft ® Windows ® rozšírený tiež operačný systém Linux. Po zvážení všetkých výhod a

nevýhod sme sa rozhodli v našej aplikácii využiť rozhranie OpenGL. Presný popis funkcií rozhrania nájde čitateľ v [OpenGL].

4 Návrh softwarového diela

4.1 Úvod

V tejto kapitole sa budeme podrobnejšie venovať návrhu softwarového diela, ktorý bude obsahovať:

- Názov aplikácie a logo
- Výber algoritmov
- Opis vstupných dát
- Výber jazyka a platformy
- Univerzálnosť
- Použité knižnice
- Hlavné triedy
- Formáty výstupu
- Používateľské GUI
- Hardwarové požiadavky

4.2 Názov aplikácie a logo

Primárnym cieľom našej aplikácie je 3D vizualizácia seizmických vlnových polí a preto sme ju pomenovali výrazom **SeismoVis**, ktorý vznikol skrátením dvoch slov **Seismo**logical **Vis**ualization. V čase písania tejto diplomovej práce (apríl 2005), žiadna iná aplikácia nemala pomenovanie SeismoVis (podľa vyhľadávača Google, www.google.com). Tematikou seizmológie sme sa inšpirovali aj pri tvorbe loga a ikony aplikácie, ktoré predstavujú vlniace sa médium, obr.23.

Obr.23: Logo a ikona aplikácie SeismoVis

4.3 Výber algoritmov

Naša aplikácia sa bude skladať z viacerých logických častí (viď kapitolu 1.5), ku ktorým si v nasledujúcich kapotolách v skratke popíšeme použité algoritmy, ktoré budeme neskôr implementovať.

4.3.1 Vstupno–výstupné operácie so súbormi

Patria sem načítanie dát terénu, vlnového poľa, čítanie a zapisovanie konfiguračných súborov, a tiež export výstupných súborov. Na realizáciu uvedených vstupno–výstupných operácií použijeme existujúce rutiny a funkcie programovacieho jazyka.

4.3.2 Zobrazenie scény a rozhrania

Na vytvorenie scény a zobrazovanie prvkov vizualizácie, teda rozhrania a vlnového poľa budeme využívať možnosti OpenGL (viď kapitolu 3.4.1), v ktorom sú všetky potrebné funkcie (vykresľovanie geometrických primitív, posúvanie, rotácia, škálovanie, farby, textúry, priesvitnosť) už implementované. Na realizáciu fixovanej a voľnej kamery v scéne využívame funkcie OpenGL a implementáciu [Štefček02]. Načítané dáta rozhrania predstavujú body hĺbkovej mapy, ktorej doména je uniformný (štvorcový) grid, pričom niektoré z nich majú nulovú hĺbku a nemajú pre nás informačnú hodnotu. Tieto po načítaní odfiltrujeme a ďalej budeme uvažovať len zvyšné body s nenulovou hĺbkou. Za účelom požadovaného zobrazenia rozhrania (ako farebný osvetlený povrch, príp. s textúrou) je potrebné zo vstupných bodov vytvoriť mesh. Najvhodnejším riešením je trojuholníkový mesh, pričom na jeho generovanie použijeme inkrementálny algoritmus Delaunayovej triangulácie s obmedzením na dĺžku strany, ktorý podrobnejšie popíšeme v kapitole 5.4. V prípade, že chceme rozhranie osvetliť, je potrebné vypočítať pre každý vrchol meshu jeho normálu. Algoritmus výpočtu je veľmi priamočiary, uvedieme ho v kapitole 5.4. Pri použití funkcie priesvitnosti rozhrania musíme pri vykresľovaní jednotlivých trojuholníkov zachovať usporiadanie od najvzdialenejšieho trojuholníka od kamery po najbližší (algoritmus z–buffer, [Shirley02], implementovaný v grafickom hardwari). Preto treba pri každej zmene pohľadu trojuholníky znova usporiadať, napríklad známym algoritmom rýchleho triedenia Quick–sort ([Shirley02]).

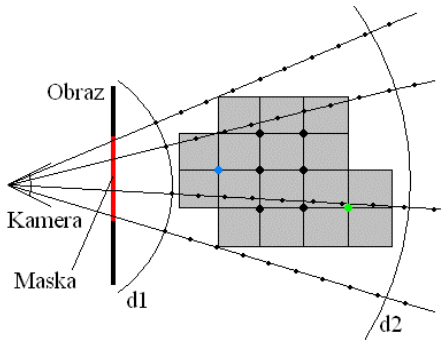
4.3.3 Zobrazenie vlnového poľa

Načítané vlnové pole predstavuje uniformný 3D grid s rozmermi *(RX, RY, RZ)* s vektorom amplitúd posunutí v smeroch *(x, y, z)* v jednotlivých vrcholoch. Z 3D gridu vytvoríme jednoduchým spôsobom (viď kapitolu 2.3.2) voxelovú mriežku. Tým sme dostali pole *DV (RX–1)(RY–1)(RZ–1)* voxelov, pričom v niektorých vrcholoch je vektor amplitúdy posunutia nulový, čo znamená, že na vizualizácii nebudú viditeľné. To spôsobuje značné plytvanie časom vykresľovania, a preto si po načítaní každého snímku uložíme do pamäte zoznam *DE* indexov tých vrcholov, v ktorých je absolútna hodnota veľkosti vektora amplitúdy väčšia ako zvolené e . V prípade použitia vyrezávania rovinami sa zoznam uložených indexov upraví podľa konkrétneho nastavenia rovín užívateľom. Následne sa upraví aj pole voxelov *DV*. Ponecháme si len tie voxely, ktoré obsahujú aspoň jeden vrchol s indexom uloženým v zozname *DE*. V ďalšom budeme uvažovať iba upravené zoznamy vrcholov a voxelov. Zobrazovanie vlnového poľa bude prebiehať v dvoch etapách. V oboch etapách bude možné zobrazovať len skalárne objemové dáta, preto budeme brať do úvahy vždy len jednu zložku vektora amplitúdy *(x, y, z)* v danom bode (amplitúdu v smere *x*, v smere *y*, alebo v smere *z*) alebo veľkosť vektora výslednice, ktorý vypočítame zo vzťahu: $v = \sqrt{x^2 + y^2 + z^2}$. Výhodou obmedzenia je najmä prehľadnejší a čitateľnejší výstup. Samotný výber druhu zobrazenia je ponechaný na užívateľa a bude sa dať v ľubovoľnom okamihu meniť.

V prvej etape sa zobrazí náhľad, ktorý vykreslí len vrcholy 3D gridu, pričom ich hodnota bude reprezentovaná odtieňom bielej (vrcholy s kladnou amplitúdou) alebo červenej (so zápornou amplitúdou) farby. V tejto fáze prebieha zobrazovanie s použitím funkcií OpenGL, pričom vlnové pole je včlenené priamo do scény s rozhraním. Kvalita výstupu nie je veľká, poskytuje len základné informácie o vlnovom poli, ale zobrazovanie je rýchle a plynulé, čo dáva užívateľovi možnosť nastavenia kamery, ktoré bude použité v druhej etape pri finálnom renderingu.

V druhej etape sa z uložených voxelov vytvorí priestorová dátová štruktúra oktálový strom (viď kapitolu 3.3.5), s pomocou ktorej sa výrazne urýchli metóda ray–castingu. V ďalšom kroku zobrazíme vrcholy vlnového poľa pomocou OpenGL (s parametrami kamery z 1.etapy) bielou a červenou farbou, pričom zvyšné časti scény nezobrazujeme. Výsledkom bude obraz s čiernym pozadím, na ktorom bude oblak bielych a červených bodov. Daným bodom určíme obálku (bounding element, opísaný obdĺžnik), ktorý následne rozšírime do všetkých strán

o veľkosť priestorovej uhlopriečky voxela, pretože na hranici môžu „vyčnievať“ voxely, ktoré majú len jeden vrchol uložený v zozname *DE*. Výsledný obdĺžnik nám definuje masku výsledného obrazu, na ktorej sa oplatí spúšťať algoritmus ray–castingu. Intenzity zvyšných bodov sa neoplatí počítať (boli by nulové), keďže lúče vedené bodmi mimo obdĺžnika nikdy nepretnú uvažované voxely vlnového poľa, čo prinesie ďalšie urýchlenie renderingu. V tejto fáze neuvažujeme rozhranie, ktoré sa bude renderovať zvlášť. Posledným prípravným krokom je zistenie minimálnej d_1 a maximálnej vzdialenosti d_2 kamery a vrcholov vlnového poľa, pričom od d_1 odpočítame a k d_2 pripočítame dĺžku priestorovej uhlopriečky voxela, obr.24.



Obr.24: Upravená metóda ray–castingu

V ďalšom spustíme pre body na maske algoritmus ray–castingu, ktorý bude pre jednotlivé pixely masky počítať intenzitu (farbu) nasledovným spôsobom. Cez daný bod masky vystrelíme lúč, pričom vo vzdialenosti d_1 od kamery začneme s uniformným vzorkovaním hodnôt intenzít a transparencií (viď kapitolu 3.3.4). Vzorkovaciu frekvenciu si určíme vopred. V každom bode odčítame hodnoty intenzity a transparentnosti z daného voxelu, v ktorom sa práve nachádzame, čo zistíme prejdením oktálového stromu (viď kapitolu 5.6), kde ako parameter uvedieme súradnice bodu. Konkrétne hodnoty potom zistíme priemerovaním alebo trilineárnou interpoláciou hodnôt vrcholov nájdeného voxela (viď kapitolu 2.3.3). Získanými hodnotami upravíme výslednú intenzitu zjednodušeným algoritmom `Front_To_Back_Oprimized` (viď kapitolu 3.3.4, 5.6) a pokračujeme ďalším krokom. Vzorkovanie ukončíme po prekročení vzdialenosti d_2 alebo pri dosiahnutí hodnoty intenzity 1 (maximálna intenzita farebnej zložky). Po ohodnotení všetkých bodov masky nám vzniká vyrenderovaný obraz vlnového poľa, ktorý v ďalšom kroku sčítame po pixeloch (každú farebnú zložku zvlášť) s projekciou zvyšnej scény (pomocou OpenGL) a dostávame finálny obrázok, ktorý môžeme priamo uložiť ako bitovú mapu alebo poslať na ďalšie spracovanie do výslednej animácie.

4.3.4 Export

Užívateľ má možnosť voľby, aký typ exportu požaduje. V prípade, že chce jednotlivé obrázky, tak sa po vyrenderovaní každej časovej snímky prekopíruje obsah výsledného obrázku do bitovej mapy, ktorá sa následne uloží do súboru. Pokiaľ sa užívateľ rozhodne pre animáciu, najprv sa inicializuje rozhranie pre tvorbu videa, kde si užívateľ zvolí typ kódovania (kodek), od ktorého závisí veľkosť a kvalita výslednej animácie. V našej aplikácii využívame rozhranie Video for Windows (VFW), ktoré je produktom spoločnosti Microsoft ®. Následne sa spustí rendering vybraných snímok, pričom obsah finálneho obrázku sa zakaždým pošle rozhraniu. Na konci sa rozhranie zavrie a výsledná animácia sa uloží do súboru.

4.4 Opis vstupných dát

Vstupnými údajmi pre našu aplikáciu budú súbory rozhrania a vlnového poľa získané z Katedry Zeme a Planét. Popis rozhrania je v textovom súbore *Layer.grd*, v ktorom každý riadok predstavuje jeden bod s nasledujúcou štruktúrou:

```
<int x> <int y> <float d>,
```

kde *x, y* sú súradnice bodu v gride a *d* je hĺbka daného bodu. Ide teda o hĺbkovú mapu, ktorej doménou je uniformný grid. Celkový rozmer gridu je 331x270 bodov.

Popis vlnového poľa pozostáva z 1197 časových snímok, s krokom 0,0555 s. Každá snímka je reprezentovaná binárnym súborom *WF_xxxxx.dat*, v ktorom je 3 krát uložená dátová štruktúra 3D uniformného gridu s rozmermi 331x270x31 bodov so skalárnymi hodnotami vo vrcholoch, ktoré predstavujú amplitúdy výchyliek daných bodov v smere x, v smere y a v smere z.

```
float*** WaveU;
float*** WaveV;
float*** WaveW;
```

Ďalšie parametre vlnového poľa sú dané v textovom súbore *wavefld.cfg*, ktorý obsahuje 10 riadkov s nasledujúcimi údajmi.

`int FirstTick, NumTicks, Interval;` hodnoty určujú prvú časovú snímku, počet snímok a interval medzi nimi

`int MinX, MinY, MinZ, MaxX, MaxY, MaxZ;` hodnoty definujú vymedzovací kváder vlnového poľa v metroch

`int Spacing;` hodnota definuje odstup vrcholov gridu v metroch

`int ScaleX, ScaleY, ScaleZ;` hodnoty definujú škálovanie poľa v smeroch *(x, y, z)* pri vykresľovaní.

Textúra použitá pri textúrovaní sa načítava zo súboru *texture.bmp*.

4.5 Výber programovacieho jazyka a platformy

Výber platformy je veľmi dôležité rozhodnutie, ktoré predurčuje skupinu užívateľov daného softwarového produktu. Po dohode so zadávateľom diela sme dospeli k rozhodnutiu vyvíjať na platforme Microsoft ® Windows ® z dôvodu rozšírenosti a užívateľskej prítulnosti.

Na vývoj aplikácie sme si vybrali programovací jazyk C++, pretože je značne rozšírený a zaužívaný, platformovo nezávislý, s existenciou množstva nástrojov podporujúcich a uľahčujúcich vývoj aplikácií. Vývojovým prostredím bude Borland C++ Builder™ verzie 6.0, pretože ide o balík RAD (Rapid Application Development) nástrojov, ktoré umožňujú rýchly vývoj aplikácií prostredníctvom predprogramovaných komponentov.

4.6 Univerzálnosť

Vyvíjaná aplikácia bude schopná načítať a vizualizovať ľubovoľné rozhranie a príslušné vlnové pole reprezentované súbormi podľa špecifikácie popísanej v kapitole 4.4. Preto ju môžeme v danej problematike označiť ako univerzálnu.

4.7 Použité knižnice a časti zdrojového kódu

Knižnice značne zjednodušujú prácu programátora, ktorý nemusí znova programovať často používané a už naprogramované komponenty a metódy, jednoducho ich len pripojí k svojej aplikácii ako knižnice alebo zdrojové súbory.
V našej aplikácii budeme používať nasledujúce knižnice.

- **OpenGL** (opengl32.dll) – Umožňuje efektívne zobrazovanie 3D scén využívajúc hardware grafickej karty. Knižnica je obsiahnutá v ovládačoch grafickej karty.
- **Video for Windows** (vfw32.lib, msvfw32.dll) – Umožňuje vytváranie animácií zo statických obrázkov. Knižnica vfw32.lib je súčasťou balíka vývojového prostredia

Borland C++ Builder™. knižnica msvfw32.dll je súčasťou operačného systému Microsoft ® Windows ®.

- **Kamera** (Camera.cpp, Camera.h) – Umožňuje realizáciu fixovanej a voľnej kamery v scéne, zdrojové súbory sú čerpané zo [Štefček02].
- **ďalšie** – Rôzne iné knižnice z balíka Borland C++ Builder™ 6.0, ktoré ralizujú zobrazenie okien, tlačidiel a pod.

4.8 Použité triedy

V nasledujúcom popíšeme triedy, ktoré sme použili v aplikácii.

- **TForm1** – Hlavná trieda, ktorá obsluhuje hlavné okno aplikácie a všetky jeho komponenty. Odchytáva a spracováva stlačené klávesy, pohyby a kliknutia myšou a komunikuje s ostatnými triedami.
- **TScene** – Obsluhuje OpenGL okno (vytvorenie, zmena rozlíšenia, uvoľnenie), vykresľuje scénu, spracováva stlačené klávesy, pohyby a kliknutia myšou. Obsahuje všetky dáta scény (rozhranie, vlnové pole, kameru, ray–caster), realizuje osvetľovanie, orezávanie, pohyb objektov.
- **TCamera** – Úlohou triedy je obsluha fixovanej a voľnej kamery v scéne – posunutia, rotácie, približovanie, odďaľovanie.
- **TTerrain –** Obsahuje všetky dáta rozhrania, realizuje vstupy/výstupy so súbormi, generovanie meshu, počítanie normál, triedenie trojuholníkov, vykresľuje rozhranie.
- **TWaveField** – Obsahuje všetky dáta vlnového poľa, realizuje vstupy/výstupy so súbormi, orezávanie, vytváranie voxelov, vykresľovanie vlnového poľa.
- **TRaytracer** – Implementuje metódu ray–castingu, štruktúru oktálového stromu.
- **TAVIGenerator** – Slúži na export animácií.
- **TColorPanel** – Slúži na jednoduchý výber farieb z palety.
- **Vector, Matrix** – Implementujú prácu s vektormi a maticami.

4.9 Formáty výstupu

Štandardne bude program podporovať dva typy obrazových výstupných dát, podľa toho, aký typ výstupu si zvolí užívateľ.

- **Statický výstup** – Výstupom je obrázok v štandardnom formáte bitovej mapy (BMP).
- **Dynamický výstup** – Výstupom je animácia, video v štandardnom formáte AVI.

Okrem obrazových dát program podporuje aj ukladanie nasledujúcich dátových štruktúr rozhrania, z ktorými pracuje.

- **Body** – Uloží sa zoznam bodov, ktoré boli vykresľované počas behu programu, teda len tie, ktorých hĺbka nebola nulová.
- **Trojuholníky** – Uloží sa trojuholníkový mesh, pozostávajúci z vyššie uvedených bodov.
- **Normály** – Pre každý vrchol sa uloží aj normála.

Užívateľ si môže vybrať typ súboru, do ktorého sa štruktúry uložia – textový alebo binárny.

V prípade textového výstupu sa kvôli prehľadnosti jednotlivé dátové štruktúry uložia do samostatných súborov nasledovným spôsobom.

Body – názov súboru *Points.dat* so štruktúrou:

Prvý riadok:

`<int NP> <int Spacing> <int MinX> <int MinY> <int MaxX> <int MaxY> <float MinD> <float MaxD> <int ScaleX> <int ScaleY> <int ScaleZ>`

Kde *NP* je počet bodov, *Spacing* je rozostup, *MinX, MinY, MaxX, MaxY, MinD, MaxD* definujú nádobu (terén), v ktorom je rozhranie, *ScaleX, ScaleY, ScaleZ* sú koeficienty škálovania.

Zvyšné riadky:

`<int X> <int Y> <float D>`

Kde *X, Y* sú súradnice bodu v gride a *D* je hĺbka bodu.

Trojuholníky – názov súboru *Triangles.dat* so štruktúrou:

Prvý riadok:

`<int NT>`

Kde *NT* je počet trojuholníkov.

Zvyšné riadky:

`<int` V_1`> <int` V_2`> <int` V_3`>`

Kde V_1, V_2, V_3 sú indexy do poľa vrcholov pre daný trojuholník.

Normály – názov súboru *Normals.dat* so štruktúrou:

Prvý riadok:

`<int NN>`

Kde *NN* je počet normál.

Zvyšné riadky:

`<float X> <float Y> <float Z>`

Kde *X, Y, Z* sú súradnice vektora normály.

V prípade binárneho výstupu sa všetky dátové štruktúry uložia do spoločného súboru *terrain.bin* nasledujúcim spôsobom. Najprv sa uložia v poradí *NP, Spacing, MinX, MinY, MaxX, MaxY, MinD, MaxD, ScaleX, ScaleY, ScaleZ,* následne pole vrcholov, ďalej *NT*, pole trojuholníkov, *NN* a pole normál. Uvedené výstupy slúžia najmä na zrýchlenie procesu načítavania rozhrania, generovania meshu a počítania normál.

Ďalšími dátami, ktoré program ukladá, sú nastavenia fixovanej a voľnej kamery. V oboch prípadoch sa používa binárny súbor. Nastavenie fixovanej kamery je uložené v súbore *fixcamera.bin*, ktorý obsahuje 3 hodnoty typu **`int`**, rotáciu v smere *x*, v xmere *y* a hodnotu priblíženia. V prípade voľnej kamery je použitý súbor *freecamera.bin*, v ktorom sú uložené 2 matice transformácií 4x4 s hodnotami typu **`double`** (rotačná a translačná) a vektor pohľadu rozložený do 3 smerov súradnicových osí – tri vektory 3x1 s hodnotami typu **`float`** (horný, pravý a predný).

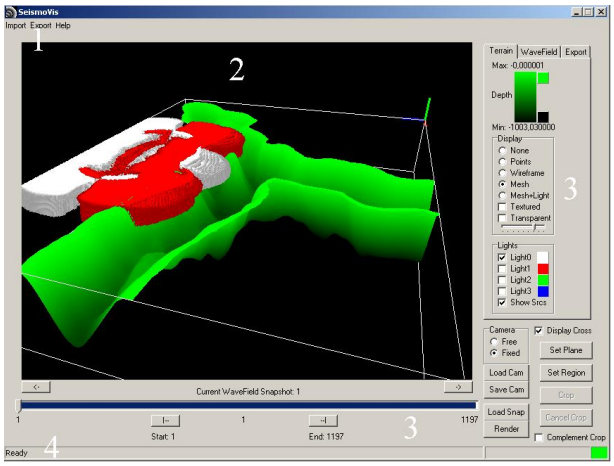
4.10 Používateľské GUI

GUI (Graphic User Interface), alebo grafické užívateľské rozhranie je priamym sprostredkovateľom funkcií programu užívateľovi, a preto musí byť navrhnuté so zmyslom pre ergonómiu a pohodlie užívateľa. Musí spĺňať nasledujúce požiadavky.

- **Jednoduchosť obsluhy** – alebo tiež užívateľská prítulnosť, pričom musíme brať do úvahy cieľovú skupinu užívateľov, ktorým je aplikácia určená a podľa toho zvoliť stupeň zložitosti dizajnu.
- **Efektivita** – Rozhranie by malo vedieť prezentovať celú funkcionalitu aplikácie s ohľadom na logické usporiadanie jednotlivých prvkov.
- **Ladnosť** – Koncepcia rozhrania by mala byť vytváraná so zmyslom pre estetiku, napr. dodržiavanie zarovnania, rovnaký štýl tlačidiel, apod.

Po spustení aplikácie sa užívateľovi zobrazí informačné okno, pokiaľ sa program nenahrá do pamäte počítača. Budú tu uvedené základné informácie, ako názov aplikácie, logo, verzia,

rok vydania, tiež meno programátora a jeho e-mailová adresa. Následne sa zobrazí hlavné okno aplikácie, ako ho ilustruje obr.25.



Obr.25: Hlavné okno aplikácie.

Popis jednotlivých komponentov okna:

1. **Hlavné menu** – ovláda najmä vstupno–výstupné operácie
2. **Samotná zobrazovacia časť** – slúži na zobrazenie 3D scény
3. **Ovládacie prvky** – nastavujú parametre programu
4. **Stavový riadok** – informuje o stave aplikácie

4.11 Hardwarové požiadavky

Úlohou práce je implementovať riešenie pamäťovo aj časovo pomerne náročného problému, teda budeme požadovať výkonný osobný počítač s dostatočne veľkou pamäťou.

4.11.1 Odhad pamäťovej náročnosti.

Uvažujme, že chceme zobraziť rozhranie rozlíšením 331x270 bodov (vo všeobecnosti uvažujeme všetky hodnoty), pričom pre každý bod si pamätáme 3 súradnice. Ďalej chceme zobraziť vlnové pole s rozmermi 331x270x31, pričom pre každý bod si pamätáme 3 hodnoty amplitúdy. Ďalej si pamätáme indexy bodov rozhrania v meshi (pre každý trojuholník 3 indexy), indexy zobrazovaných vrcholov (vo všeobecnosti všetkých), zoznam voxelov vlnového poľa (pre každý voxel 8 indexov), obrazové dáta renderovaného obrazu (výška*šírka*3), štruktúru oktálového stromu (v každej kocke 2 body teda 6 súradníc) a mnoho iných údajov. Predpokladajme, že všetky údaje sú reálne čísla, na uloženie ktorých potrebujeme 4 bajty. Z toho dostávame výraz: 331*270* 3*4bajty + 331*270*31*3*4bajty +

331*270*4bajty + 331*270*31*4bajty + 331*270*31*8*4bajty + 640*480*3*4bajty + 331*270*31*6*4bajty = 195MB, pričom sme uvažovali extrémny prípad. Pri danom rozlíšení rozhrania a vlnového poľa je teda horná hranica použitej pamäte okolo 200MB, preto odporúčame počítač vybaviť minimálne 512MB pamäťou.

4.11.2 Odhad časovej náročnosti.

Najviac náročnou metódou aplikácie z hľadiska času je bezpochyby metóda ray–castingu. Uvažujme znova extrémny prípad, majme vlnové pole o rozmeroch 331x270x31 voxelov a výsledný obraz s rozlíšením 640x480 bodov, pričom predpokladajme, že maska pokrýva celý obraz a všetky voxely majú nenulovú hodnotu amplitúd. Vyhľadanie voxela v oktálovom strome trvá rádovo $\log_8(512*512*512)$ operácií, pre každý pixel bude vyhľadaní približne 1000 (naprieč poľom), potom ďalších rádovo 1000 operácií je potrebných na poskladanie výslednej intenzity. Celkový počet operácií pre jeden časový okamih teda môžeme rádovo vyjadriť vzťahom: $640*480*\log_8(512*512*512)*2000 = 4{,}3$ mld. operácií. Počítajme, že na vykonanie jednej operácie je potrebných rádovo 10 operácií v pohyblivej rádovej čiarke, z čoho dostaneme výpočtovú náročnosť 43000 MFLOPS. Podľa tab.2 by vyrenderovanie jednej snímky na počítači Intel Pentium 4 3,6 GHz trvalo približne 9 sekúnd, celá animácia (1197 snímok) teda 180 minút. Do tohto času však nie je započítaný čas potrebný na načítanie časových snímok zo súborov z pevného disku, ktorý závisí od konkrétnej konfigurácie počítača. Pre reálny vstup by však mal stačiť procesor s taktom 1GHz pri zachovaní času renderovania približne 15 hod.

4.11.3 Zhrnutie

Minimálne požiadavky na beh našej aplikácie sú nasledujúce:

- Počítač PC alebo kompatibilný s 32 bitovým operačným systémom Microsoft ® Windows ®
- Procesor Intel Pentium III alebo AMD Athlon 1GHz
- Pamäť 512 MB RAM
- Grafická karta Nvidia Geforce 2 alebo ekvivalent s podporou OpenGL
- 10 MB voľného miesta na pevnom disku pre potreby aplikácie, vstupné súbory nerátame

5 Implementácia

5.1 Úvod

V tejto kapitole sa budeme podrobnejšie venovať samotnej implementácii našej aplikácie. Rozdelili sme ju do viacerých celkov, ktoré na seba logicky a časovo nadväzujú. Najskôr popíšeme dizajn a funkcionalitu aplikácie, neskôr sa budeme venovať jednotlivým metódam, ktoré sme navrhli v kapitole 4.3. Pri popise dátových štruktúr a metód budeme používať konštrukcie jazyka C++ a tiež dátové typy a funkcie zabudované vo vývojovom prostredí Borland C++ Builder™ 6.0. Podrobnosti nájde čitateľ v [Borland02].

5.2 Hlavné okno GUI

Hlavné okno aplikácie, hlavný formulár (obr.25) poskytuje užívateľovi všetky funkcie programu. Reprezentovaný je triedou *TForm1*, ktorá používa inštancie tried T*Scene*, *TAVIGenerator* a T*ColorPanel*. Sú nimi *Scene*, *AVIGenerator* a *ColorPanel*. Zdrojový kód je v súboroch *Main.h* a *Main.cpp*. Priamymi funkciami formulára sú riadenie kamery a procesu ray–castingu. Okrem toho obsahuje množstvo komponentov vykonávajúcich rôzne funkcie. Po aktivácii komponentu (napríklad kliknutím myšou) sa vykoná príslušná funkcia formuláru, alebo inštancie triedy, ktorú trieda *TForm1* používa. Najdôležitejšími komponentmi sú hlavné menu v hornej časti, panel s kresliacou plochou v strede, súbor ovládacích prvkov napravo a dole a stavový riadok dole.

5.2.1 Hlavné menu

Funkciou hlavného menu sú najmä vstupno–výstupné operácie, teda načítavanie dát scény a export. Prvá položka v *Import*, *Import Terrain from Text* slúži na načítanie dát rozhrania z originálneho textového súboru *Layer.grd*, druhá položka *Import Terrain from Binary* načíta dáta rozhrania z binárneho súboru *terrain.bin*, *Export Terrain to Text* a *Export Terrain to Binary* slúžia na uloženie rozhrania do textových alebo binárneho súboru (viď kapitolu 4.9). *Initialize WaveField* slúži na inicializáciu vlnového poľa, pričom bude načítaná aktuálne vybraná časová snímka na časovej osi. V nasledujúcej skupine *Export* sú možnosti *Export to BMPs* a *Export to AVI*, ktoré spustia ray–casting a vygenerujú z vyznačeného intervalu

snímok, obrázky vo formáte BMP alebo animáciu vo formáte AVI. Poslednou skupinou je *Help*, ktorá obsahuje položku *About* (zobrazí informácie o aplikácii).

5.2.2 Panel s kresliacou plochou

Panel s kresliacou plochou je určený na zobrazovanie samotného okna rozhrania OpenGL. Pri štarte aplikácie sa pointer na panel spolu s rozmermi panelu pošlú inicializačnej funkcii *Scene.Main*, ktorá zabezpečí vytvorenie OpenGL okna. Počas behu programu kresliaca plocha zabezpečuje prekresľovanie OpenGL okna, odchytáva udalosti spojené s ľavým tlačidlom myši a posiela ich *Scene*.

5.2.3 Ovládacie prvky

Ovládacie prvky napravo sú rozdelené do 5 častí:

- Nastavenia scény – Možnosti výberu *Camera*, tlačidlá *Load Cam* a *Save Cam*, možnosť *Display Cross*
- Nastavenia rozhrania – Hárok *Terrain*
- Nastavenia vlnového poľa – Hárok *WaveField*
- Nastavenie exportu – Hárok *Export*
- Nastavenia orezávania – Tlačidlá *Set Plane*, *Set Region*, *Crop* a *Cancel Crop*, možnosť *Complement Crop*
- Nastavenia časových snímkov – Časová os a tlačidlá ß, à, ├, ┤, *Load Snap* a *Render*.

Užívateľ si vyberie z dvoch typov kamier buď fixovanú kameru (možnosť *Fixed*) alebo voľnú kameru (možnosť *Free*). Aktuálne nastavenie pohľadu je možné uložiť tlačidlom *Save Cam* a obnoviť tlačidlom *Load Cam*. Na vypnutie/zapnutie zobrazovania trojrozmerného kríža (je vhodný na lepšiu orientáciu v 3D priestore) slúži možnosť *Display Cross*.

V hárku *Terrain* je po načítaní rozhrania vidno minimálnu a maximálnu hĺbku bodov, ktoré sú reprezentované čiernou a zelenou farbou. Užívateľ si môže jednotlivé farby zmeniť kliknutím na príslušný farebný štvorec. Zobrazí sa farebná paleta, z ktorej sa pravým tlačidlom myši vyberie farba pre minimálnu alebo maximálnu hĺbku. Farby zvyšných hĺbok sa lineárne interpolujú. V možnostiach Display sa nastavuje typ zobrazovania rozhrania. *None* nezobrazí žiadne rozhranie, *Points* zobrazí rozhranie ako bodovú výškovú mapu, *Wireframe* vykreslí

drôtený model, *Mesh* vizualizuje trojuholníkový mesh a *Mesh+Light* rozhranie osvetlí. Možnosť *Textured* zapne textúrovanie rozhrania, *Transparent* zapne priesvitnosť, pričom mieru transparentnosti upravuje posúvač nižšie. V možnostiach *Lights* môže užívateľ zapínať a vypínať svetlá kliknutím na príslušné svetlo (*Light0* až *Light3*), pričom farba svetiel sa nastavuje analogicky ako pri farbe hĺbky. Po zapnutí svetla sa v scéne zobrazí kocka s príslušnou farbou, ktorou je možné hýbať a tým presúvať zdroj svetla. Zobrazovanie zdrojov svetiel je možné vypnúť/zapnúť kliknutím na *Show Srcs*.

V hárku *WaveField* sa po načítaní vlnového poľa zobrazí informácia o počte voxelov, ktoré sa zobrazujú (*# of elements*). Ďalej sa dá zmeniť smer amplitúdy (*WaveField Direction*), ktorého hodnota sa bude vizualizovať na *None* (žiadna), *U* (smer *x*), *V* (smer *y*), *W* (smer *z*) alebo *Resultant* (výslednica). Zobrazovanie záporných hodnôt amlitúd v bodoch je možné vypnúť/zapnúť pomocou *Show Negative Values*. Ďalej môže užívateľ meniť možnosti ray–castingu, konkrétne metódu interpolácie (*Flat* alebo *Trilineárna*) a vzorkovaciu frekvenciu (*Sampling Precision*).

V hárku *Export* je možné zmeniť cesty a názvy výstupných súborov, tiež pozíciu a obsah textu, ktorý bude vložený do obrázkov alebo animácie.

Aplikácia podporuje 2 typy orezávania – pomocou rovín kolmých na osi a pomocou všeobecnej roviny. Začiatok orezávania inicializujú tlačidlá *Set Region*, resp. *Set Plane*. V závislosti od typu orezávania sa zobrazí 6 orezávacích rovín s úchytkami alebo 3 body reprezentujúce všeobecnú rovinu. Pomocou úchytiek resp. bodov je možné rovinami hýbať. Po nastavení rovín je potrebné zvoliť si typ orezania. Pokiaľ má byť v prípade kolmých rovín určený región ponechaný a zvyšok vyrezaný, možnosť *Complement Crop* nesmie byť označená, v opačnom prípade sa označí. V prípade všeobecnej roviny možnosť *Complement Crop* určuje polpriestor (orezávacia rovina je hranicou), z ktorého budú body vyrezané. Na spustenie orezávania nastavenej oblasti slúži tlačidlo *Crop*. V prípade, že nechceme orezávať, zobrazenie rovín vypneme opätovným stlačením *Set Region*, resp. *Set Plane*. Stlačením tlačidla *Cancel Crop* zrušíme vykonané orezanie.

Na časovej osi si užívateľ volí časovú snímku, ktorú chce zobraziť, pričom číslo aktuálne vybranej snímky sa zobrazuje pod osou. Stlačením tlačidla *Load Snap* sa zvolená snímka načíta a zobrazí sa náhľad v OpenGL. Číslo aktuálne zobrazenej snímky je uvedené nad osou. Tlačidlo ß slúži na načítanie a zobrazenie predchádzajúcej snímky, naopak à načíta a

zobrazí nasledujúcu. Pokiaľ si užívateľ želá snímku vyrenderovať (ray–castingom), stlačí tlačidlo *Render*. Zobrazí sa okno s tlačidlami *Close* a *Save BMP*. Tlačidlom *Close* sa preruší proces renderingu a aplikácia sa vráti k zobrazeniu náhľadu. *SaveBMP* slúži na uloženie vyrenderovanej snímky do súboru bitmapy. Tlačidlami ├ a ┤ sa nastavuje rozsah výstupnej animácie, ktorý je zobrazený modrou farbou na časovej osi, navyše čísla začiatočnej a koncovej snímky sú uvedené pod ├ a ┤.

5.2.4 Stavový riadok

Hlavnou funkciou stavového riadku je informovať užívateľa o stave aplikácie. Skladá sa z 3 častí, ľavá časť je vyhradená na slovný popis práve vykonávanej akcie, stredná časť informuje o dosiahnutom pokroku pri vytváraní videa (pomer počtu vygenerovaných k počtu vybraných snímok) a pravá časť svieti na zeleno, pokiaľ je program v nečinnosti a na červeno, keď pracuje.

5.2.5 Riadenie kamery

Po spustení programu sa načítajú nastavenia obidvoch typov kamier zo súborov *fixedcamera.bin* a *freecamera.bin* funkciou *Scene.Camera.Load*. Pri uložení pozície kamery sa zavolá funkcia *Scene.Camera.Save*. Zmenu typu kamery riadi funkcia *Scene.Camera.SetCamera*. Pri zapnutej voľnej kamere sa po stlačení jedného z kláves A, S, D, W, Q, E (reprezentujú pohyb voľnej kamery v smerom doľava, dozadu, doprava, dopredu, dole, hore) vykonajú príslušné funkcie *Scene.Camera.TranslateF* (dopredu/dozadu), *Scene.Camera.TranslateR* (doprava/doľava), *Scene.Camera.TranslateU* (dole/hore). Pri zapnutej fixovanej kamere sa pri otáčaní kolieska na myši (reprezentuje približovanie a odďaľovanie) vykoná jedna z funkcií *Scene.Camera.ZoomIn, Scene.Camera.ZoomOut.* Pri ľubovoľnej kamere sa pri stlačení pravého tlačidla na myši (reprezentuje začiatok rotácie) zavolá funkcia *Scene.InitRotate,* pri držaní tlačidla a pohybe myši je to funkcia *Scene.Rotate* a pri uvoľnení tlačidla (reprezentuje koniec rotácie) funkcia *Scene.EndRotate.*

5.2.6 Riadenie Ray–castingu

Metóda ray–castingu sa uvedie do činnosti v prípade, že si užívateľ zvolil jednu z možností výstupu alebo stlačil tlačidlo *Render*. Priebeh celého procesu možno ilustrovať nasledujúcou schémou:

1. V prípade exportu inicializuj prázdnu animáciu alebo prázdnu bitmapu.
2. Načítaj vybranú časovú snímku, uprav príslušné zoznamy vrcholov a voxelov, vytvor oktálový strom – funkcia *Scene.WaveField.JumpToTick*.
3. Zobraz celú scénu okrem vlnového poľa a výsledný obraz ulož do pamäte.
4. Spusti proces ray–castingu v triede *Scene*– funkcia *Scene.Ray*.
    a. Zisti rozmery okna, polohu kamery, vektor pohľadu rozložený do 3 vektorov v smere súradnicových osí, minimálnu a maximálnu vzdialenosť voxelov od kamery.
    b. Inicializuj Ray–caster – funkcia *Raytracer.Initialize*.
    c. Zobraz v scéne len body vlnového poľa a výsledný obraz ulož do pamäte.
    d. Pomocou uloženého obrazu vytvor masku – funkcia *Raytracer.UpdateMask*.
    e. Pošli ray–casteru oktálový strom a príslušné pole hodnôt vlnového poľa.
    f. Spusti metódu ray–castingu – funkcia *Raytracer.RunRayCast*.
5. Výsledný obraz po zložkách (RGB) sčítaj s uloženým obrazom zvyšnej scény.
6. Zobraz finálnu snímku na obrazovku.
7. V prípade exportu ulož snímku do bitmapy alebo pridaj do animácie.
8. V prípade animácie opakuj kroky 2 až 6 pre celý rozsah zvolených snímok.

5.3 Scéna

Scéna je reprezentovaná triedou T*Scene,* ktorá používa inštancie tried, T*Terrain, TWaveField, TCamera* a T*Raytracer*. Sú nimi *Terrain*, *WaveField*, *Camera* a *Raytracer*. Zdrojový kód je v súboroch *Scene.h* a *Scene.cpp*. Hlavnou úlohou triedy je najmä obsluha OpenGL okna, ďalej riadi vykresľovanie všetkých objektov, osvetľovanie rozhrania a orezávanie.

5.3.1 OpenGL okno

Primárnym účelom OpenGL okna je zobrazovanie scény a jej častí, dá sa však použiť aj na rôzne pomocné projekcie, napríklad pri vytváraní masky pre ray–casting. Po štarte programu je z hlavného formulára zavolaná funkcia *Main(Panel1 à Handle, 640, 480)*, ktorá vytvorí v hlavnom formulári na paneli 1 okno s rozmermi 640x480 bodov. Prácu s oknom počas celého behu programu ilustruje nasledujúca schéma:

1. Vytvor OpenGL okno – funkcie *Main* a *CreateGLWindow*.

2. Nastav parametre zobrazovania (kameru, FOV, farbu pozadia, farebnú hĺbku, ...) – funkcie *InitGL* a *ReSizeGLScene*.
3. Zobraz scénu – funkcia *DrawGLScene*.
    a. Zmaž obrazovku a nastav všade farbu pozadia.
    b. Maticu zobrazenia nastav na identitu.
    c. Uprav kameru podľa aktuálnych hodnôt rotácie a podľa toho uprav maticu zobrazenia – funkcia *Camera.Process*
    d. Uprav vlastnosti objektov (napríklad farbu, transparentnosť, svetlo, ...) – funkcia *GLLight*.
    e. Vykresli objekty.
    f. Opakuj kroky *d* a *e* na vykreslenie všetkých objektov.
    g. Vykresli frame–buffer (doteraz boli všetky operácie vykonávané virtuálne v buffri–zásobníku v pamäti grafického hardwaru).
4. V prípade zmeny polohy alebo rozmeru okna, preveď príslušné nastavenia a zobraz scénu. – funkcia *ReSizeGLScene*.
5. V prípade udalosti z klávesnice alebo myši uprav kameru, vlastnosti objektov, pridaj alebo zruš objekty (ak treba) a zobraz scénu.
6. Pri externej požiadavke na prekreslenie scény zobraz scénu.
7. Pri ukončení programu uvoľni OpenGL okno – funkcia *KillGLWindow*.

5.3.2 Vykresľovanie objektov

V závislosti od aktuálneho nastavenia sa vykresľujú nasledujúce objekty:

- Trojrozmerný farebný kríž – priamo vo funkcii *DrawGLScene.*
- Rozhranie – funkcia *Terrain.Draw.* V prípade zapnutej priesvitnosti rozhrania sa pri každej zmene kamery vykoná utriedenie trojuholníkov – funkcia *SortFaces*.
- Vlnové pole – funkcia *WaveField.Draw.*
- Zdroje svetla – priamo vo funkcii *DrawGLScene*.
- Úchytky a body orezávacích rovín a samotné roviny – funkcie *DrawHandles, DrawCropPoints.*

5.3.3 Osvetľovanie rozhrania

Aplikácia podporuje osvetlenie rozhrania maximálne 4 svetlami, pričom po spustení programu sa ich farba nastaví postupne na bielu, červenú, zelenú a modrú. Po zapnutí osvetľovania užívateľom sa automaticky zapne nulté biele svetlo. Zmenu farieb a stavu svetiel obsluhuje funkcia *ChangeLightProps*. Pred každým zobrazením rozhrania sa púšťa funkcia *GLLight*, ktorá nahráva svetelné dáta do OpenGL. Po stlačení ľavého tlačidla myši v scéne sa skontroluje (funkciou *InteractLights*), či nebol kurzorom zasiahnutý niektorý zdroj svetla nasledujúcim spôsobom. Každý zdroj sa premietne z 3D do 2D a príslušné 2D súradnice sa porovnajú so súradnicami kurzora myši. Pokiaľ sú niektoré dostatočne blízko, tak sa príslušný zdroj označí ako pohybujúci (funkcia *InitLightMove*) a pri pohybe myši sa posúva v rovine kolmej na smer kamery (funkcia *LightMove*). Po uvoľnení tlačidla sa pohyb preruší a zdroj sa odznačí (funkcia *EndLightMove*).

5.3.4 Orezávanie

Aplikácia podporuje 2 spôsoby orezávania scény. Prvým je orezávanie šiestimi rovinami, ktoré sú kolmé na súradnicové osi (po dvojiciach). Jednotlivé roviny majú svoje úchytky, pomocou ktorých sa dajú posúvať v smere príslušnej osi. Druhým spôsobom je orezávanie ľubovoľnou rovinou definovanou 3 bodmi v scéne, ktorými možno ľubovoľne hýbať. Po inicializácii orezávania (funkcia *StartHandling*) sa po stlačení ľavého tlačidla myši v scéne skontroluje (funkciou *InteractHandles* alebo *InteractCropPoints*), či nebola kurzorom zasiahnutá niektorá úchytka alebo bod tým istým spôsobom ako v prípade zdrojov svetiel. Ak áno, tak sa príslušná úchytka (bod) označí ako pohybujúca (funkcia *InitHandleMove*) a pri pohybe myši sa posúva spolu so svojou rovinou v smere príslušnej osi (v prípade bodov ľubovoľne) (funkcia *HandleMove*). Po uvoľnení tlačidla sa pohyb preruší a úchytka (bod) sa odznačí (funkcia *EndHandleMove*). Orezanie realizuje funkcia *Crop*, zrušenie orezania funkcia *CancelCrop*.

5.4 Rozhranie

Rozhranie je reprezentované triedou *TTerrain*, ktorá využíva triedu *Vector* (zdrojové súbory *Vector.h*, *Vector.cpp*). Zdrojový kód je v súboroch *Terrain.h* a *Terrain.cpp*. Úlohou triedy je implementácia metód spojených s rozhraním, teda načítavanie, generovanie meshu, počítanie

normál, vykresľovanie, triedenie trojuholníkov a export. V pamäti sú uložené nasledujúce dátové štruktúry rozhrania:

`Vector3f``* Nodes;` pole súradníc vrcholov

`Vector3i``* Triangles;` pole indexov vrcholov trojuholníkov

`Vector3f``* Normals;` pole normál vrcholov

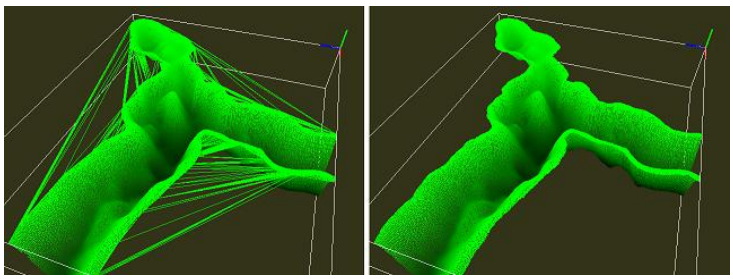
V závislosti od typu vstupného súboru, ktorý môže byť textový alebo binárny, použijeme pri načítavaní rozhrania funkciu *ImportFromText* (príkazom na čítanie súboru je *fscanf*) alebo funkciu *ImportFromBinary* (príkaz *fread*). Pri načítavaní z originálneho textového súboru *Layer.grd* uložíme do dátovej štruktúry len body s nenulovou hĺbkou, pričom mesh a normály musíme vypočítať. Pri použití binárneho súboru naplníme naraz všetky tri dátové štruktúry.

Na generovanie meshu slúži funkcia *Triangulate*, ktorá s pomocou funkcie *Circled* vytvorí z načítaných bodov výškovej mapy 3D mesh s trojuholníkovou doménou. Použitým algoritmom je inkrementálny algoritmus Delaunayovej triangulácie v rovine, ktorý môžeme použiť aj v našom prípade, lebo doména bodov je 2D uniformný grid. Pri vytváraní použijeme iba 2 súradnice bodov *(x, y)* (súradnicu *z* nepotrebujeme), teda pracujeme v rovine. Postup ilustruje nasledujúca schéma:

1. Danej množine bodov opíšeme super–trojuholník tak, aby boli všetky body vo vnútri.
2. Do vznikajúceho meshu postupne pridávame všetky body.
3. Pre všetky doteraz vytvorené trojuholníky otestujeme, či sa pridaný bod nenachádza vnútri opísanej kružnice niektorého trojuholníka (funkcia *Circled*).
4. Ak áno, príslušný trojuholník zo zoznamu vytvorených vymažeme a nahradíme ho tromi novými, ktoré vzniknú spojením pridaného bodu s vrcholmi vymazaného trojuholníka a doplnením príslušných strán.
5. Ak nie, pokračujeme ďalším trojuholníkom v poradí.
6. Po pridaní posledného bodu vymažeme z meshu tie trojuholníky (len strany), ktorých aspoň jeden vrchol patrí super–trojuholníku.
7. Vymažeme aj vrcholy super–trojuholníka.
8. Vzniknutý mesh predstavuje Delaunayovu trianguláciu množiny vstupných bodov.
9. My však požadujeme, aby boli trojuholníky zložené len z bodov, ktoré sú blízko pri sebe. Nechceme, aby tvorili trojuholník aj body z opačných koncov rozhrania, čo je

reálne, pretože zjednotenie trojuholníkov je konvexný obal, obr.26. Preto obmedzíme dĺžku strany.

10. Výstupom je trojuholníkový mesh, aký sme požadovali.



Obr.26: Delaunayova triangulácia bez a s obmedzením dĺžky strán

V prípade, že chceme rozhranie osvetliť, potrebujeme poznať pre každý vrchol meshu pred výpočtom Phongovho lokálneho osvetľovacieho modelu (počíta ho grafický hardware) jeho normálu. Počítanie normál pozostáva z dvoch krokov (funkcia *ComputeNormals*). Najprv si určíme normály jednotlivých trojuholníkov meshu, a z nich vypočítame normály vrcholov priemerovaním. Normálu trojuholníka *ABC* vypočítame zo vzťahu $\vec{n} = \frac{(C-A)\times(B-A)}{|(C-A)\times(B-A)|}$, je to teda normalizovaný vektorový súčin vektorov reprezentujúcich 2 strany trojuholníka. Ďalej pre každý vrchol meshu nájdeme všetky incidentné trojuholníky a normálu vrchola vyjadríme ako aritmetický priemer normál príslušných trojuholníkov.

Aplikácia ponúka vykresľovanie rozhrania šiestimi spôsobmi, ktoré realizuje funkcia *Draw*. Nezávisle na zvolenom type zobrazovania sa najprv vykreslia bielymi úsečkami obrysy nádoby, ktoré ohraničujú oblasť vizualizácie. Následne sa vykreslí rozhranie podľa zvoleného typu. Pri prvých troch typoch zadávame zložky farby jednotlivých bodov funkciou *glColor3f* pred *glVertex3f*. Pri kreslení úsečiek a trojuholníkov sa farba mimo vrcholov interpoluje.

- **Body** – použije sa návestie *glBegin(GL_POINTS)*, po ktorom súradnice každého bodu pošleme funkcii *glVertex3f*.
- **Drôtený model** – začíname návestím *glBegin(GL_LINES)*, po ktorom posielame funkcii *glVertex3f* dvojice súradníc bodov, ktoré reprezentujú strany trojuholníkov meshu.
- **Mesh** – začíname návestím *glBegin(GL_TRIANGLES)*, po ktorom posielame funkcii *glVertex3f* trojice súradníc bodov reprezentujúcich trojuholníky meshu.

- **Osvetlený mesh** – ako Mesh, len zapneme aj návestie *glEnable(GL_LIGHTING)* a pred *glVertex3f* pošleme najprv súradnice normály pomocou funkcie *glNormal3f*.
- **Textúrovaný mesh** – ako Mesh, len zapneme aj návestie *glEnable(GL_TEXTURE_2D),* následne načítame textúru funkciou *glBindTexture* a pred *glVertex3f* pošleme najprv textúrové koordináty pomocou funkcie *glTexCoord2f.*
- **Priesvitný mesh** – ako Mesh, len zapneme aj návestie *glEnable(GL_BLEND)* a namiesto *glColor3f* použijeme *glColor4f*, pričom posledný parameter bude miera transparentnosti.

V prípade použitia transparentnosti treba vždy po zmene polohy alebo smeru kamery utriediť trojuholníky meshu od najvzdialenejšieho po najbližší ku kamere (funkcia *QuickSortFaces*).

Export rozhrania zabezpečujú funkcie *ExportToText* (príkaz *fprintf*) a *ExportToBinary* (príkaz *fwrite*) v závislosti od typu výstupného súboru.

5.5 Vlnové pole

Vlnové pole je reprezentované triedou *TWaveField,* ktorej zdrojový kód je v súboroch *WaveField.h* a *WaveField.cpp*. Úlohou triedy je implementácia metód spojených s vlnovým poľom, teda načítavanie, generovanie voxelov, generovanie a mazanie oktálového stromu, vykresľovanie a orezávanie. V pamäti sú uložené nasledujúce dátové štruktúry vlnového poľa:

`float*** WaveU;` 3D pole amplitúd vlnenia v smere *x*, hodnoty vrcholov gridu

`float*** WaveV;` 3D pole amplitúd vlnenia v smere *y*, hodnoty vrcholov gridu

`float*** WaveW;` 3D pole amplitúd vlnenia v smere *z*, hodnoty vrcholov gridu

`float*** WaveUVW;` 3D pole výsledníc amplitúd vlnenia, hodnoty vrcholov gridu

`bool*** DV;` 3D pole voxelov, ak je hodnota *DV[i,j,k]=true*, potom voxel reprezentovaný ľavým dolným zadným rohom *(i,j,k)* renderujeme a jeho hodnota amplitúdy je *WaveX[i,j,k]*.

`TIndex* DE;` pole indexov tých vrcholov, ktoré vykresľujeme

Na začiatku sa načíta textový súbor *WaveField.cfg* a naplnia sa hodnoty parametrov poľa. Načítanie príslušnej časovej snímky vlnového poľa obsluhuje funkcia *JumpToTick*, zavolá funkciu *Import*, ktorá načíta hodnoty poľa z binárneho súboru *WF_xxxxx.DAT*, kde xxxxx je číslo snímky. Naplnia sa polia *WaveU, WaveV* a *WaveW,* z ktorých sa vypočíta pole *WaveUVW* podľa vzťahu:

$WaveUVW[i,j,k] = \sqrt{WaveU[i,j,k]^2 + WaveV[i,j,k]^2 + WaveW[i,j,k]^2}$. Následne sa zavolá funkcia *GetDrawnElements*, ktorá aktualizuje polia *DV* a *DE* podľa zvoleného typu zobrazovania (U, V, W, Výslednica) a príslušných hodnôt orezávania.

Na generovanie voxelov slúži funkcia *Voxelize*. Výstupom je zoznam vykresľovaných voxelov a ich kvádrová obálka v priestore. Následne sa spustí funkcia *BuildOcTree*, ktorá spolu s *BuildBranch* vytvorí na voxeloch oktálový strom, ktorý sa použije na urýchlenie ray–castingu. Vytváranie stromu popisuje nasledujúca schéma:

1. Z hranolovej obálky voxelov vytvor kockovú obálku s rozmerom 2^n–koreň stromu.
2. Pokiaľ sa v aktuálnej kocke nachádza aspoň jeden voxel (funkcia *IsAnyVoxel*), rozdeľ kocku na 8 rovnakých pod–kociek a spusti algoritmus na každej z nich.
3. Pokiaľ kocka neobsahuje žiadny voxel, označ ju ako prázdnu a vráť sa o úroveň vyššie.
4. Pokiaľ kocka obsahuje práve jeden voxel, označ ju indexom tohto voxela a vráť sa o úroveň vyššie.
5. Algoritmus ukonči, ak sa dostaneš späť do koreňa.

Mazanie oktálového stromu realizuje funkcia *DeleteOcTree*.

Vykresľovanie vlnového poľa v prvej etape (náhľad) zabezpečuje funkcia *Draw*, ktorá vykreslí jednotlivé položky poľa *DE* vo forme bielych (reprezentujú kladné hodnoty amplitúd) a červených bodov (reprezentujú záporné hodnoty amplitúd), pričom príslušný odtieň závisí od absolútnej hodnoty amplitúdy. Druhú etapu realizuje ray–casting, viď. kapitolu 5.6.

Program podporuje 2 typy orezávania, paralelné a všeobecné. Realizujú ich funkcie *CropParallel* a *CropArbitrary*, ktoré nastavia parametre, pričom samotné orezanie je vykonané vo funkcii *SetDrawnElements*. V prípade paralelného orezania sa porovnávajú súradnice jednotlivých voxelov so súradnicami orezávacích rovín. Keďže sú roviny kolmé na súradnicové osi, porovnávanie je veľmi jednoduché. Pri vypnutom komplementárnom orezaní ponecháme body nachádzajúce sa vo vnútri kvádrového regiónu určeného rovinami a zvyšné vyrežeme, v opačnom prípade ponecháme vonkajšie body. Pri všeobecnom orezaní sa vypočítajú orientované vzdialenosti jednotlivých voxelov od orezávacej roviny, pričom hodnoty s rozdielnym znamienkom reprezentujú voxely na rôznych stranách roviny.
V závislosti od zapnutého komplementárneho orezania ponecháme len body s kladnými alebo zápornými hodnotami vzdialenosti.

5.6 Ray–caster

Ray–caster je reprezentovaný triedou *TRaycaster,* ktorá využíva triedu *Vector* (zdrojové súbory *Vector.h*, *Vector.cpp*). Zdrojový kód je v súboroch *Raycaster.h* a *Raycaster.cpp*. Úlohou triedy je implementácia metódy ray–castingu a pomocných metód.

Pred spustením samotného ray–castingu je potrebné nastaviť parametre metódy: rozmery okna, polohu a smer kamery (funkcia *Initialize*), minimálnu a maximálnu vzdialenosť voxelov od kamery (funkcia *WaveField.GetMinMaxDistance*). Následne sa spustí funkcia *UpdateMask*, ktorá nastaví masku bodov okna, cez ktoré sa budú viesť lúče. V tomto okamihu je možné spustiť funkciu *RunOptimizedRaycast*, ktorá cez príslušné body okna vedie lúče a s pomocou *OptimizedRaycast* vyjadruje výsledné intenzity technikou uvedenou v kapitole 4.3.3, pričom vzorkovacia frekvencia je daná hodnotou *SamlingRate*, prechádzanie oktálovým stromom realizuje funkcia *OcTreeIntersect* a interpoláciu funkcia *InterpolateIntensity*. Prechádzanie oktálovým stromom ilustruje nasledujúca schéma:

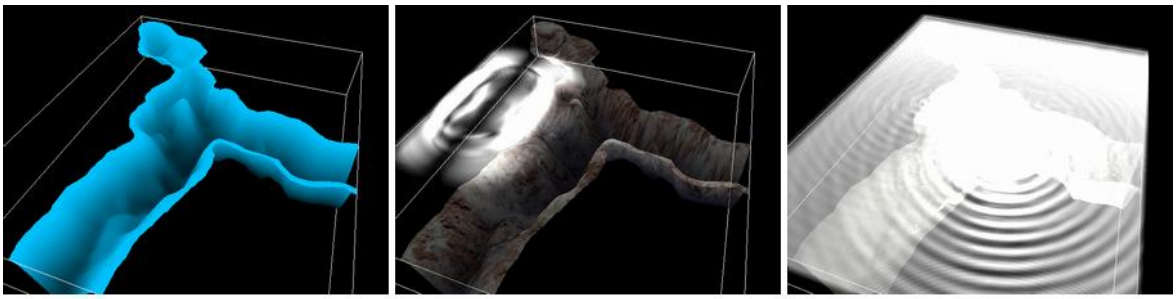
1. Vstupom algoritmu je pointer na aktuálny vrchol (kocka) oktálového stromu (na začiatku koreň) a bod v priestore, ku ktorému chceme zistiť voxel, v ktorom sa nachádza.
2. Ak je kocka označená ako prázdna, skonči a vráť **`false`**.
3. Zisti, či bod zapadá do časti priestoru vymedzenej kockou, ak nie, skonči a vráť **`false`**.
4. Vypočítaj rozmer kocky (metrikou je veľkosť voxelu).
5. Pokiaľ je rozmer 1, nastav výstup *(x,y,z)* na aktuálny voxel, skonči a vráť **`true`**.
6. Ak nie, tak spusti algoritmus pre všetky pod–kocky aktuálnej kocky.
7. Výstupom algoritmu sú indexy voxela *(x,y,z)* vo vlnovom poli v prípade, že sa daný bod v strome nachádza, inak algoritmus vráti hodnotu **`false`**.

Zo získaného voxela sa následne interpoláciou intenzít jeho vrcholov určí hodnota intenzity vo vzorkovanom bode. Za hodnotu intenzity v danom vrchole voxela považujeme absolútnu hodnotu výchylky vo zvolenom smere (*U, V, W*, výslednica). V našom prípade neuvažujeme hodnoty transparentnosti, pretože sú rovnaké vo všetkých vrcholoch (vrcholy sa odlišujú len intenzitou, viď kapitolu 3.3.7). Funkcia *InterpolateIntensity* realizuje 2 typy interpolácie, *flat* a *trilineárnu*. Pri *flat* interpolácii sa hodnota bodu vo voxeli vypočíta jednoduchým aritmetickým priemerom hodnôt 8 vrcholov, v druhom prípade sa použije postup popísaný

v kapitole 2.3.3. Získaná hodnota intenzity sa jednoducho pripočíta k celkovej intenzite pre daný bod obrazu. Pokiaľ celková intenzita presiahne hodnotu 1 (maximálna hodnota intenzity farebnej zložky), vzorkovanie sa ukončí. Výsledná intenzita sa priradí všetkým farebným zložkám prislúchajúceho bodu, čím dosiahneme efekt „bielej hmly“.

6 Výsledky

V tomto odseku uvedieme výstupy našej aplikácie v podobe vyrenderovaných obrázkov a animácií rozhrania a vlnového poľa. Na obr.27a je zobrazené rozhranie v podobe trojuholníkového meshu, pričom farba závisí od hĺbky. Vlnové pole je vypnuté. Obr.27b znázorňuje rozhranie v podobe textúrovaného meshu so zapnutou transparentnosťou, vlnové pole vizualizuje U–zložku prvej snímky a bolo vyrenderované so vzorkovacou frekvenciou 0,05 a zapnutou flat interpoláciou. Obr.27c zobrazuje polotransparentné rozhranie osvetlené 4 bielymi svetlami, vlnové pole predstavuje výslednicu 1184. snímky, presnosť vzorkovania bola 0,05, použitá bola metóda trilineárnej interpolácie.



a b c

Obr.27: Výstupy aplikácie vo forme BMP

Najdôležitejšími výstupmi aplikácie sú však animácie *WaveU.avi, WaveV.avi, WaveW* a *WaveUVW.avi*, ktoré sa nachádzajú na priloženom CD. Animácie zobrazujú polotransparentné textúrované rozhranie, zložky vlnového poľa (U, V, W a výslednica) sa vizualizujú vo všetkých 1197 časových snímkach, pričom vzorkovacia frekvencia je nastavená na 0,05 a použitá je metóda *flat* interpolácie. Tab.6 udáva časy potrebné na vyrenderovanie jednotlivých animácií na rôznych počítačoch.

Počítač\Animácia	WaveU.avi	WaveV.avi	WaveW.avi	WaveUVW.avi
Intel Pentium III 933Mhz	15.7 hod	16.1 hod	15.2 hod	16.8 hod
Intel Pentium 4 2.0 GHz	9.3 hod	9.5 hod	8.8 hod	10.4 hod

Tab.6: Časy renderovania výstupných animácií na rôznych počítačoch

7 Záver

Cieľom diplomovej práce bolo implementovať 3D vizualizáciu seizmických vlnových polí na osobnom počítači s využitím grafického hardwaru. Autormi návrhu sú RNDr. Peter Moczo, DrSc. a Mgr. Jozef Kristek, PhD. z Katedry fyziky Zeme a plánét FMFI UK. Výsledkom je aplikácia *SeismoVis*, ktorá spĺňa všetky podmienky zadania (viď kapitolu 1.5). Pri jej tvorbe sme analyzovali rôzne algoritmy a metódy počítačovej grafiky (viď kapitolu 3), z ktorých sme si vybrali najvhodnejšie podľa zvolených kritérií (viď kapitolu 4). Zvolené techniky sme následne upravili podľa našich predstáv a implementovali do samotnej aplikácie (viď kapitolu 5). Výsledkom sú názorné obrázky alebo animácie rozhrania seizmických vrstiev a vlnového poľa, ktoré môžu byť použité ako prezentačný materiál použitých techník výpočtu vlnového poľa. Aplikácia môže byť tiež využívaná ako pomôcka pri výskume vlnových polí na fyzikálnej katedre.

Samotný program (*SeismoVis.exe*), zdrojový kód (adresár *Source*), konfiguračné súbory, vstupný súbor rozhrania a časť vstupných súborov vlnového poľa (adresár *data*) spolu s podrobným popisom funkcií programu (*SeismoVis.doc*) a ukážkovými animáciami (adresár *export*) sa nachádzajú na priloženom CD.

8 Budúca práca

V budúcnosti je možné do aplikácie doplniť rôzne nové funkcie podľa požiadaviek Katedry fyziky Zeme a planét alebo tiež vylepšiť existujúcu funkcionalitu, napríklad celkovo zrýchliť proces renderingu. Možno pri tom využiť nasledujúce prístupy:

- Viac optimalizovať ray–caster, napríklad použiť zložitejšiu hierarchickú reprezentáciu voxelov.
- Využiť 3D textúry, v prípade zakúpenia modernejšieho grafického hardwaru.
- Implementovať server–client model na distribuované generovanie jednotlivých časových snímok na viacerých počítačoch prostredníctvom počítačovej siete.

9 Zoznam použitej literatúry

[Aken02] AKENINE-MOLLER, T. – HAINES, E. 2002. *Real-Time Rendering (2nd edition)*. AK Peters Ltd. ISBN: 1568811829

[Aki02] AKI, K. – RICHARDS, P.G. 2002. *Quantitative Seismology*. University Science Books. ISBN: 0935702962

[Biel00] BIELAK, J. et al. 2000. The Quake Project. web stránka. http://www-2.cs.cmu.edu/~quake/

[Borland02] Borland Software Corporation. 2002. *Borland C++ Builder™ Help*.

[DirectX] Microsoft ® Corporation. Microsoft ® DirectX ® Technology. web stránka. Apríl 2005. http://www.microsoft.com/windows/directx/default.aspx

[Farlo93] FARLOW, S.J. 1993. *Partial Differential Equations for Scientists and Engineers*. Dover Pubns. ISBN: 048667620X

[Furu03] FURUMURA, T. et al. 2003. Visualization of 3D Wave Propagation from the 2000 Tottori-ken Seibu, Japan, Earthquake: Observation and Numerical Simulation. *Bulletin of the Seismological Society of America*. Vol. 93. pp.870 – 881

[Havran00] HAVRAN, V. 2000. *Heuristic Ray Shooting Algorithms*. Dissertation Thesis. Faculty of Electrical Engineering, Czech Technical University, Prague. elektronický text. http://www.mpi-sb.mpg.de/~havran/DISSVH/dissvh.pdf

[Huang02] HUANG, J. 2002. *Direct Volume Rendering: Ray-casting*. powerpointová prezentácia. www.cs.utk.edu/~huangj/CS594S02/raycasting.ppt

[Chopra02] CHOPRA, V. 2002. *Seismic Waves and Inversion*. powerpointová prezentácia. http://www.ccs.uky.edu/~douglas/Classes/cs521-s02/seismic/seismic.ppt

[Janotka04] JANOTKA, V. 2004 *Seizmický prieskum.* elektronický text. http://www.fns.uniba.sk/~kgf/Seizmika.pdf

[Levoy88] LEVOY, M. 1988. Volume Rendering: Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications.* Vol 8. Number 3. pp 29–37

[Luber00] LUBER, M. – HASTREITER, P. 2000. *Lecture Notes for Scientific Visualization.* Friedrich–Alexander University Erlangen–Nürnberg. elektronický text. http://www9.informatik.unierlangen.de/Teaching/SS2003/Vis/Material/Lecture%20Notes

[Moczo00] MOCZO, P. – KRISTEK, J. – HALADA, L. 2000. 3D 4th-Order Staggered- Grid Finite-Difference Schemes: Stability and Grid Dispersion. *Bulletin of the Seismological Society of America*. Vol. 90. pp. 587 – 603

[Moczo04] MOCZO, P. – KRISTEK, J. – HALADA, L. et al. 2004. *The Finite-Difference Method for Seismologists An Introduction.* Comenius University Bratislava. ISBN: 802232005

[Moczo98] MOCZO, P. 1998. Introduction to Modelling Seismic Wave Propagation by the Finite Difeerence Method. *Lecture Notes.* Kyoto University.

[Ning93] NING, P. – BLOOMENTHAL, J. 1993. An Evaluation of Implicit Surface Tilers, *IEEE Computer Graphics and Applications.* Vol 13. Number 6. pp 33–41.

[OpenGL] The OpenGL Consortium. *The Industry's Foundation for High Performance Graphics*. web stránka. Apríl 2005. http://www.opengl.org.

[Phong75] PHONG, B. T. 1975. Illumination for Computer Generated Pictures. *Communications of the ACM.* Vol. 18. Number 6. pp 311–317

[Sala03] SALAMA, CH.R. 2002. *Volume Rendering – Direct Methods – Hardware Based Methods*. Friedrich–Alexander University Erlangen–Nürnberg. powerpointová prezentácia. http://www9.informatik.uni-erlangen.de/Teaching/SS2003/Vis/Material/Slides%20Volume%20Visualization%202

[Shirley02] SHIRLEY, P. 2002. *Fundamentals of Computer Graphics*. AK Peters Ltd. ISBN: 1568811241

[Štefček02] ŠTEFČEK, M. 2002. Implementácia voľnej kamery v OpenGL. Zdrojové súbory TCamera.cpp, TCamera.h

[VRML] The VRML Consortium. *The Virtual Reality Modeling Language Specification*. web stránka, Apríl 2005. http://vag.vrml.org.

[Weather] The National Oceanic and Atmospheric Administration. *National Weather Service*. web stránka. Apríl 2005. http://www.weather.org.

[Wikipedia] The Wikipedia Consortium. *The Free Encyclopedia*. internetová encyklopédia. Apríl 2005. http://www.wikipedia.org.