

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Spracovanie modelov vo forme mračna bodov
DIPLOMOVÁ PRÁCA

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

SPRACOVANIE MODELOV VO FORME MRAČNA BODOV

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	9.2.1 Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	doc. Ing. Branislav Sobota, PhD.
Konzultant:	doc. Ing. Branislav Sobota, PhD.

Abstrakt v SJ

Táto práca sa zaoberá implementáciou Delaunayovej triangulácie v trojrozmernom priestore využitej na rekonštrukciu povrchu. Táto triangulácia je jedna z najpoužívanějších typov. Algoritmus opísaný v práci je založený na myšlienke inkrementálneho algoritmu. Práca sa tiež zaoberá možnosťou paralelizácie, následnom implementovaní a odskúšaní algoritmu aj na GPU.

Kľúčové slová

Delaunay, triangulácia, GPU

Abstrakt v AJ

This paper describes the implementation of Delaunay triangulation in three dimensions used to reconstruct the surface of model. This triangulation is one of the most common types. The algorithm described in the work is based on the idea of incremental algorithms. The paper also describes parallelization of algorithm, implementation and testing the algorithm on the GPU.

Kľúčové slová v AJ

Delaunay, triangulation, GPU

TECHNICKÁ UNIVERZITA V KOŠICIACH

Fakulta elektrotechniky a informatiky

Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: 9.2.1 Informatika

Študijný program: Informatika

Názov práce:

Spracovanie modelov vo forme mračna bodov
Processing of models in the form of point clouds

Študent (tituly, meno, priezvisko): **Bc. Dominik Januvka**

Školiteľ (tituly, meno, priezvisko): **doc. Ing. Branislav Sobota, PhD.**

Školiace pracovisko: **Katedra počítačov a informatiky**

Konzultant práce (tituly, meno, priezvisko):

Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

1. Teoreticky naštudovať problematiku počítačovej grafiky, priestorového modelovania a triangulácie
2. Navrhnuť a implementovať programové vybavenie pre proces triangulácie mračna bodov s výstupom do niektorého zo štandardných formátov na základe pokynov vedúceho práce s prípadným využitím technológie GPGPU
3. Experimentálne overiť implementáciu na vybraných príkladoch objektov a v prípade možnosti aj spracovania výstupu z 3D skenera
4. Vypracovať dokumentáciu podľa pokynov vedúceho práce

Jazyk, v ktorom sa práca vypracuje:

slovenský

Termín pre odovzdanie práce:

27.04.2012

Dátum zadania diplomovej práce:

31.10.2011


prof. Ing. Ján Kollár, CSc.
vedúci garantujúceho pracoviska




prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som celú diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry a informačných zdrojov.

Košice, 15. máj 2012

.....

vlastnoručný podpis

Predhovor

Problematika Delaunayovej triangulácie je známa ako matematický problém už dlhšiu dobu. V počítačovej grafike si táto metóda našla miesto v oblasti rekonštrukcie povrchov, hlavne z dát získaných z rôznych skenerov. V tejto práci sa zaoberám spracovaním týchto údajov a následným využitím tejto metódy v implementácii problému triangulácie. Výsledný program je schopný bežať ako na bežných procesoroch, tak aj na heterogénnych platformách s využitím procesorov grafických kariet (kariet na ktorých sú možné spúšťať paralelne výpočty). Paralelná časť programu je implementovaná v štandarde OpenCL.

Využitím inkrementálneho algoritmu a niekoľkých konštánt zvolených užívateľom pri štarte, je algoritmus schopný zrekonštruovať takmer každý typ povrchu s vysokou úspešnosťou celkového výsledku.

Aj keď triangulácia samotná sa častejšie vyskytuje pri triangulácií dvojrozmernej, mojim zámerom bolo využiť Delaunayjovu metódu pre trojrozmerný priestor. Práca sa tiež venuje optimalizácií neusporiadaného mračna bodov, ako aj jej implementácií na GPU.

Obsah

Zoznam obrázkov	8
Zoznam tabuliek	10
Zoznam symbolov a skratiek	11
Slovník termínov	12
Úvod	13
1 Úvod do problematiky rekonštrukcie povrchu	14
1.1 Čo je to triangulácia.....	14
1.2 3D skenery.....	14
1.3 Zobrazovanie	14
1.3.1 Mriežkový model	14
1.3.2 Polygóny	15
1.3.3 Vlastnosti trojuholníka.....	16
2 Triangulácia.....	17
2.1 Typy triangulácií	17
2.2 Delaunayova triangulácia	18
2.3 Nejednoznačnosť určenia polohy bodu	20
2.4 Algoritmy	22
2.5 Euler-Poincaré Formula.....	23
3 Paralelná architektúra.....	25
3.1 Úvod do paralelných výpočtov na GPU	25
3.1.1 Architektúra GPU	26
3.1.2 Programovací model GPU	27
3.1.3 Pamäťový model GPU	29
3.2 OpenCL	30
3.3 Zrýchlenie a efektívnosť	32
4 Návrh systému	33
4.1 Motivácia	33
4.2 Požiadavky	33
4.3 Definovanie pojmov pri reprezentácii údajov	35
4.4 Optimalizácia.....	36
4.4.1 Blízke body	36

4.4.2	Metrika	37
4.4.3	Výber štartovacieho bodu	40
4.5	Algoritmus triangulácie	41
4.5.1	Paralelný algoritmus triangulácie	44
4.6	Paralelný algoritmus optimalizácie	45
4.6.1	Blízke body	45
4.6.2	Metrika	46
4.7	Výber kandidátov pre trianguláciu	46
4.8	Čítanie a zápis 3D dát	48
4.8.1	Formát Obj	48
4.8.2	Formát WRL	49
4.8.3	Formát ASE	50
4.8.4	Vstup	52
4.8.5	Výstup	52
4.9	Náhodné vstupné body	52
4.10	Modely a skeny	53
5	Generovanie povrchu	55
5.1	Chyby a úspešnosť	55
5.1.1	Post produkcia modelov	57
5.2	Spracovanie dát z 3D skenera	59
5.3	Výkonnosť	61
5.3.1	Merané zariadenia	61
5.3.2	Zrýchlenie a efektívnosť	62
5.3.3	Porovnanie	62
5.3.4	Štatistické súbory	66
6	Záver	68
	Zoznam použitej literatúry	70
	Prílohy	73

Zoznam obrázkov

Obr. 1 Delaunayove kružnice a mriežka v 2D [7]	18
Obr. 2 Delaunayova triangulácia aplikácie, výstup v 2D	19
Obr. 3 Výpočet polohy bodu d voči kružnici určenej bodmi abc [19]	21
Obr. 4 Bod voči kružnici [18].....	21
Obr. 5 Bod voči kružnici, priblíženie [18].....	22
Obr. 6 Komplikované teleso [16]	24
Obr. 7 Paralelne násobenie polí [30]	26
Obr. 8 Architektúra GPU [25]	27
Obr. 9 Organizácia jadier GPU [24]	28
Obr. 10 Hierarchia pamäte OpenCL [24]	31
Obr. 11 Proces skenovania a rekonštrukcie povrchu modelu.....	33
Obr. 12 Prehľadová schéma implementácie	35
Obr. 13 Spájanie blízkych bodov, model "hlava" (horný rad zľava: pôvodný model, tolerancia 0,02 s časom 44 minút, dolný rad: tolerancia 0,04 s časom 9 minút, tolerancia 0,1 s časom 5 minút).....	37
Obr. 14 Porovnanie povrchov s rôznymi vstupnými konštantami.....	39
Obr. 15 Porovnanie parametrov výberu bodu do triangulácie (priemerná vzdialenosť naľavo, minimálna vzdialenosť napravo)	40
Obr. 16 Algoritmus Delaunayovej triangulácie	43
Obr. 17 Paralelne vytváranie nových hrán a zároveň spúšťanie procesov	44
Obr. 18 Porovnanie výberu kandidátov (horný rad zľava: pôvodný model, podľa menšej gule, spodný rad: podľa väčšieho uhlu, menšej vzdialenosti od hrany)	48
Obr. 19 Príklad výstupného súboru Obj	49
Obr. 20 Príklad výstupného súboru WRL	50
Obr. 21 Príklad výstupného súboru ASE.....	51
Obr. 22 Generovanie náhodných bodov v 3D	53
Obr. 23 Proces triangulácie, vstup, výstup	55
Obr. 24 Kritické oblasti modelov	56
Obr. 25 Ukážka post produkcie v prostredí 3D studio Max. Zmena preferencií materiálu (vľavo), alebo modifikátorom (vpravo).....	58
Obr. 26 Porovnanie "surového" modelu a post produkcie.....	58
Obr. 27 Porovnanie výstupov z triangulácie 3D skenera (hore) a triangulácie tejto práce (dole)	59

Obr. 28 Bližší pohľad na porovnanie výstupov z triangulácie 3D skenera (vľavo) a triangulácie tejto práce (vpravo)	60
Obr. 29 Teoretický výsledok GPGPU výpočtov v porovnaní s CPU	63
Obr. 30 Časová závislosť procesu metriky od počtu vstupných bodov	64
Obr. 31 Časová závislosť procesu spájania bodov od počtu vstupných bodov	64
Obr. 32 Časová závislosť triangulačného procesu náhodne generovaných bodov	65
Obr. 33 Graf závislosti triangulácie v závislosti od vstupných modelov	66
Obr. 34 Príklad štatistického súboru	67

Zoznam tabuliek

Tabuľka 1 Prehľad špecifikácie testovaných zariadení	61
Tabuľka 2 Zrýchlenie a efektívnosť	62

Zoznam symbolov a skratiek

2D	dvojdimenzionálny, dvojrozmerný priestor
3D	trojdimenzionálny, trojrozmerný priestor
MB	megabajt, jednotka veľkosti binárnych dát v miliónoch
MHz	jednotka frekvencie, vyjadruje počet taktov za sekundu v miliónoch

Slovník termínov

CUDA je paralelná výpočtová architektúra od NVIDIE.

GPU je grafický procesor.

GPGPU plní úlohy, bežne vykonávané na procesore, prostredníctvom GPU.

JOCL prepojenie Javy a OpenCL.

OpenCL je platformovo nezávislý otvorený štandard pre paralelné programovanie heterogénnych systémov.

Triangulácia (v počítačovej grafike) je proces vytvárania povrchu modelov z 2D alebo 3D bodov vytváraním trojuholníkov.

Úvod

Téma triangulácie je vždy aktuálna z dôvodu hľadania optimálneho algoritmu tvorby povrchu modelu z rôznych skenovacích zariadení. Veľké množstvo algoritmov má ako výsledok nesprávne striangulovaný povrchov a výpočet trvá veľmi dlho. Hľadanie vhodného, do značnej miery tiež paralelného, algoritmu nám dnes dovoľuje aj rozšírenosť a popularita GPGPU grafických kariet.

V súčasnej dobe je možné nájsť grafické karty s povolenými výpočtami takmer vo všetkých nových desktopových konfiguráciách a po novom aj v notebookoch. Technológia CUDA a jej podobné umožňujú využiť vysoký potenciál paralelného výpočtového výkonu na výpočty aj iného ako len grafického charakteru. Spojením Javy a OpenCL technológie máme v rukách silný a platformovo nezávislý nástroj na vyvíjanie aplikácií pre heterogénne systémy.

Prínosom tejto práce je vznik programového nástroja založeného na vyššie spomínaných algoritmoch a platformách, ktorý sa dá využiť ako nástroj pre spracovanie modelov vo forme neusporiadaného mračna bodov, teda ako výstupu z rôznych 3D skenovacích zariadení. Ten sa dá využiť v rôznych odvetviach každodenného života, či už v zdravotníctve, armáde, pri vývoji hier, projektovaní strojárenských riešení, kartografií, atď.

Zdrojom informácií sa z prevažnej časti stal internet a zahraničné publikácie. Práca rozoberá teóriu triangulácie, jej paralelizácie a implementovaní v OpenCL technológii.

1 Úvod do problematiky rekonštrukcie povrchu

1.1 Čo je to triangulácia

Veľa odvetví využíva skenovacie zariadenia a výstupy z nich je potrebné správne a v krátkom čase spracovať. 3D skenovanie sa dá využiť v rôznych odvetviach, napríklad v zdravotníctve, armáde, pri vývoji hier, pri geologických prieskumoch, pri rôznych simuláciách, projektovaní priemyselných riešení, kartografií a mnoho ďalších. Tento proces vyžaduje rýchly a hlavne spoľahlivý nástroj pre spracovanie získaných dát. Výsledkom triangulácie je vo všeobecnosti 2D alebo 3D mriežka tvorená (naskenovanými) bodmi a hranami medzi nimi (tie vypočítal algoritmus). V počítačovej grafike, hlavne v oblasti rekonštrukcie povrchov, sa často používa a je aj časom overená Delaunayova triangulácia (pozri [1]).

1.2 3D skenery

3D skener je určený pre snímanie trojrozmerných predmetov. Je založený na kombinácii snímania (skenovania) pomocou kamery aretovanej na podstavci, alebo statickej kamery na otočnom podstavci. Proces snímania spočíva v vyhotovení obrázku zvoleného predmetu z rôznych uhlov, pričom platí, že čím je predmet nepravidelnejší, tým je potrebných viac pohľadov aby bol získaný sken celého objektu. Z výsledných načítaných údajov je systém schopný uložiť mračno bodov do zvoleného formátu. V praxi sa väčšinou používajú 3D nedeštruktívne skenery, ktoré skenované súčiastky v procese skenovania nezničia. Do kategórie najčastejšie používaných skenerov patria optické, laserové, mechanické a magneticko-rezonančné skenery [36].

Magneticko-rezonančné skenery sú skenery snímajúce vnútorné tvary objektu využívajúc magnetickú rezonanciu. Používané sú hlavne v zdravotníctve. Výstup zo skenovacích zariadení je súbor množiny bodov s informáciami o ich polohe (súradnice x , y a z) a o ich farbe.

1.3 Zobrazovanie

1.3.1 Mriežkový model

3D model sa dá v počítačovej grafike reprezentovať rôzne. Jednou z metód je použitie mriežkového modelu. Je to súbor množín vrcholov a hrán prepájajúcich vrcholy. V jednej množine sú uchovávané postupne označené body so súradnicami v

troch rozmeroch. V druhej množine je uchovaná informácia o tom, ako sú navzájom prepojené tieto vrcholy. Každá hrana má dva vrcholy, povedzme pravý a ľavý. Každá hrana je taktiež postupne označovaná a sú jej priradené práve dva existujúce vrcholy, a to spomínaný pravý a ľavý. Takýto model neuchováva žiadne informácie o povrchu modelu.

1.3.2 Polygóny

Základným stavebným prvkom pri zobrazovaní modelov v počítačovej grafike je polygón - mnohouholník.

Mnohouholník (všeobecne n -uholník) je časť roviny vymedzená úsečkami, ktoré spájajú určitý počet bodov (najmenej tri), z ktorých žiadne tri susedné neležia na jednej priamke (v tom prípade sú kolineárne). Tiež sa dá povedať že mnohouholník je obmedzená časť roviny ohraničená uzatvorenou lomenou čiarou. Body, ktoré určujú mnohouholník, sa nazývajú vrcholy mnohouholníka. Úsečky, ktoré spájajú susedné vrcholy, sa nazývajú strany mnohouholníka. Počet vrcholov, strán a vnútorných uhlov v jednom mnohouholníku je rovnaký a tento počet určuje názov mnohouholníka (trojuholník, štvoruholník, päťuholník atď.) [2].

Najjednoduchším typom mnohouholníka a najvhodnejším pre reprezentáciu modelov je trojuholník.

Trojuholníky majú veľa užitočných vlastností, ktoré nám umožňujú písať veľmi efektívny a jednoduchý kód, ako pre výpočty tak aj pre zobrazovanie. Tri body definujú rovinu, takže trojuholník sa vždy nachádza v rovine ním určenej, kde môžeme využiť mnoho jeho vlastností. Dôležitou vlastnosťou je že každý zložitý polygón sa dá rozdeliť na množinu trojuholníkov. Dokonca aj zložitá 3D zakrivené plochy je možné aproximovať do množiny trojuholníkov. Preto sú trojuholníky základným stavebným prvkom 3D zobrazovania [3].

Zobrazenie modelu v 3D počítačovej grafike sa tiež nazýva renderovanie. Je to konverzia virtuálneho povrchu (alebo mriežky) 3D modelu, uloženého v nejakom formáte, na 2D výstupné zariadenie, napríklad grafický displej. Trojuholníky, respektíve polygóny, môžu v spojení s mriežkovým modelom uchovávať informáciu o povrchu 3D modelu. Každý trojuholník má tri hrany zo súboru existujúcich hrán mriežkového modelu (za predpokladu že ide o jeden a ten istý model) a každá hrana má dva existujúce vrcholy. Keď sa dá vytvoriť takáto štruktúra, môže sa hovoriť o 3D modeli. V tom prípade sa dá táto informácia uchovať a uložiť do niektorého zo štandardných

formátov pre uchovanie 3D modelov. Toto zobrazenie je však nejednoznačné pretože takto zostrojený trojuholník nemá určenú prednú a zadnú stranu, čo sa môže prejaviť ako tmavý trojuholník pri jeho zobrazovaní. Metód na ošetrenie tohto problému je hneď niekoľko, najjednoduchším je priradenie obojstranného materiálu v 3D grafickom editore, alebo orientácia hrán trojuholníka. Z orientovaných hrán sa dajú vypočítať vektory a z vektorov, vektorovým súčinom, aj orientácia (normála) trojuholníka.

1.3.3 Vlastnosti trojuholníka

Je to jeden zo základných rovinných geometrických útvarov, je to 2D útvar, môže sa definovať ako prienik troch pol rovín [5]. Je najjednoduchším typom mnohoúhelníka s tromi stranami (hranami). Predpokladá sa že je trojuholník ABC so stranami AB, BC, AC, označenými ako vektory a , b a c . Trojuholník je zostrojiteľný ak jeho všetky tri vrcholy neležia na jednej priamke, to je možné dokázať nasledovne:

$$(b^{\rightarrow} - c^{\rightarrow}) \times (a^{\rightarrow} - b^{\rightarrow}) = 0^{\rightarrow} \quad (1)$$

Teda ak sa vektorový súčin rovná nule, body sú kolineárne a nemožno zostrojiť trojuholník z daných troch bodov. Ďalšou hlavnou vlastnosťou trojuholníka je možnosť zostrojiť kružnicu jemu opísanú z troch bodov. V 3D kružnicu sa môže interpretovať kvôli výpočtom ako guľu, s tým istým stredom a polomerom ako kružnica. Táto kružnica (resp. guľa) prechádza všetkými bodmi trojuholníka a tak trojuholník leží v kružnici. Stred kružnice opísanej v 3D priestore sa vypočíta nasledovne:

$$p = c + \frac{[\|a-c\|^2(b-c) - \|b-c\|^2(a-c)] \times [(a-c) \times (b-c)]}{2\|(a-c) \times (b-c)\|^2} \quad [6] \quad (2)$$

V zátvorkách $\|.\|$ je dĺžka vektoru a $\times$ je vektorový súčin [6]. Polomer kružnice opísanej sa vyjadruje jednoducho, a to vypočítaním vzdialenosti ľubovoľného bodu trojuholníka od stredového bodu, pomocou známej Pytagorovej vety:

$$a^2 + b^2 = c^2 \quad (3)$$

2 Triangulácia

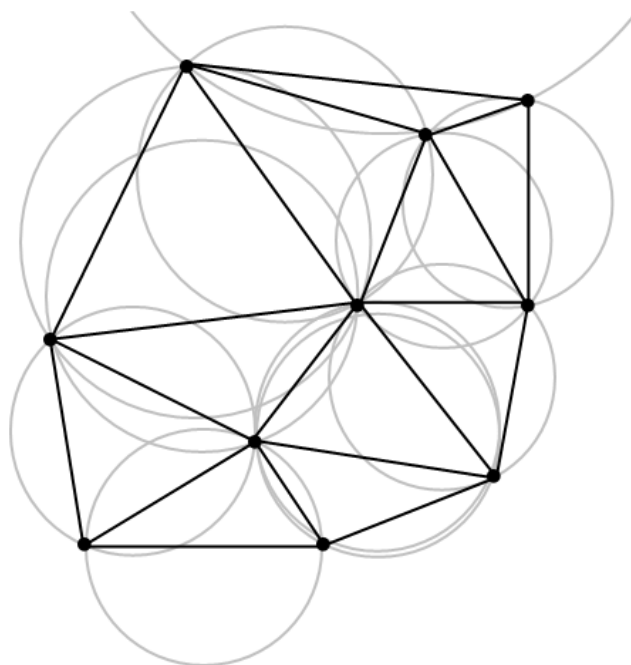
2.1 Typy triangulácií

Existuje mnoho rôznych typov triangulácií, ktoré kladú dôraz na rozličné kritéria. Všeobecne tie najjednoduchšie v 2D sa môže rozdeliť do niekoľkých skupín (podľa [8]):

- Triangulácia optimalizujúca uhly. Táto kategória je najdôležitejšia, pretože trojuholníky neobsahujúce príliš veľké alebo príliš malé uhly a preto sú pre väčšinu aplikácií veľmi užitočné. Dosiahnutie tohto cieľa je možné maximalizáciou minimálnej veľkosti uhlov trojuholníkov (Delaynauova triangulácia), minimalizáciou maximálnych veľkostí uhlov (tzv. min max triangulácia), maximalizáciou sumy uhlov, a kombináciou alebo odvodením týchto spôsobov.
- Triangulácia optimalizujúca dĺžku hrán. Tato metóda je v praxi málo používaná z dôvodu príliš veľkej algoritmickej zložitosti. To obmedzuje jej použitie len na množinu s maximálnym počtom bodov rádovo do niekoľko tisíc maximálne. Tým pádom je táto metóda pre dnešné potreby značne nepoužiteľná. K najznámejším zástupcom tejto metódy patria: triangulácia minimalizujúca celkovú sumu dĺžky hrán (minimal weight triangulation), jej lokálna aproximácia, optimalizujúca dĺžku hrany každých dvoch susedných trojuholníkov (greedy triangulation) a iné.
- Triangulácia optimalizujúca viacero kritérií. Triangulácia tohto typu kladie dôraz na viacero kritérií súčasne. Využívajú sa lokálne výmeny hrán a nedeterministické výpočty, akým je napríklad vývojová optimalizácia. Nevýhodou sú pomalé výpočty spôsobené vývojovými algoritmami, ktoré znižujú využiteľnosť tejto triangulácie.
- Údajovo závislá triangulácia. Rovinné triangulácie dokážu uspokojivo spracovať aj 3D údaje. Výnimku tvoria terény s ostrými výčnelkami. Takýto typ dát je potrebné spracovať špeciálnym spôsobom, údajovo závislou trianguláciou, ktorá berie do úvahy aj výšku bodov (osa Z) [8].

2.2 Delaunayova triangulácia

Delaunayova triangulácia je triangulácia konvexnej obálky bodov v množine, kde každá z opísaných kružníc trojuholníka je prázdna kružnica [1].

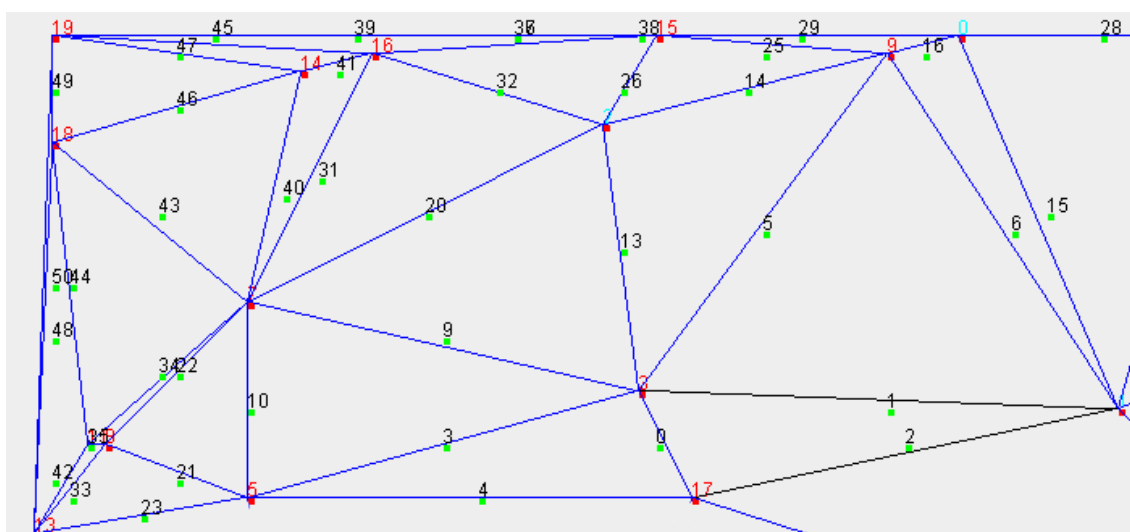


Obr. 1 Delaunayove kružnice a mriežka v 2D [7]

Delaunayove triangulácie sú široko používané v rôznych aplikáciách, pretože majú niekoľko dobrých vlastností, vďaka ktorým sú veľmi užitočné v 2D a 3D simuláciách. Najmä 2D Delaunay triangulácia maximalizuje veľkosť najmenšieho uhla pre všetky možné triangulácie v množine bodov. Preto tento druh triangulácie, založený na trojuholníkoch, robí trojuholníky s vyrovnanejším pomerom uhlov strán ako pri ostatných trianguláciách. Delaunayova triangulácia sa dá doceliť dvoma spôsobmi:

- vytvorením výslednej mriežky zo vstupnej geometrie alebo vstupného súboru bodov, alebo
- prevodom už vytvoreného, nesprávne striangulovaného modelu (mriežky), opätovnou trianguláciou na správnu štruktúru [4].

Druhý prípad sa objavuje v prípadoch keď už bola vygenerovaná triangulácia s nejakou inou triangulačnou metódou alebo bol model deformovaný použitím rôznych transformačných pravidiel [4].



Obr. 2 Delaunayova triangulácia aplikácie, výstup v 2D

Množina náhodne usporiadaných bodov v N rozmernom priestore je nazvaná množinou bodov P . V matematike Delaunayova triangulácia (DT) pre množinu P bodov v rovine je taká triangulácia $DT(P)$, pre ktorú platí že žiaden bod z P nie je vo vnútri kružnice opísanej akéhokoľvek trojuholníka v $DT(P)$. Delaunayove triangulácie prirodzene maximalizujú minimálny uhol všetkých vnútorných uhlov trojuholníkov v triangulácii, až na niektoré extrémny (ako napríklad okrajové trojuholníky pri 2D), majú tendenciu vyhnúť sa úzkym nepravidelným trojuholníkom. Pre množinu minimálne troch bodov ležiacich na rovnakej priamke sa nedá uplatniť Delaunayova triangulácia (pretože body sú kolinéárne a nedá sa zostrojiť trojuholník). Pre štyri alebo viac bodov ležiacich na jednej kružnici (napr. vrcholy obdĺžnika) Delaunayova triangulácie nie je jednoznačná. Každá z možných triangulácií, ktorá rozštiepi štvoruholník na dva a viac trojuholníkov spĺňa Delaunayovu podmienku prázdnej opísanej kružnice. Pre množinu P bodov v n -dimenzionálnom priestore platí, Delaunayova triangulácia je taká triangulácia $DT(P)$ pre ktorú je vyhovená podmienka, že žiaden bod z P nie je vo vnútri opísanej-sféry (pre 2D je to kružnica, pre 3D je to guľa) akéhokoľvek trojuholníka z $DT(P)$. V roku 1934 bola táto triangulácia pomenovaná po ruskom matematikovi menom Boris Delaunay [7].

Viac formálne by sa dala triangulácia popísať takto. Nazvime množinu trojuholníkov, množinou T . V dvoch rozmeroch, triangulácia množiny bodov (vrcholov) P je množina trojuholníkov T , so spoločnými vrcholmi, kde sa zároveň ich

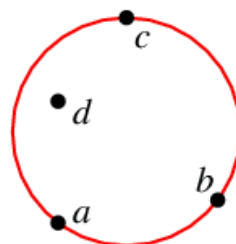
vnútra neprekrývajú vzájomne medzi sebou. Delaunayova triangulácia $DT(P)$, je graf definovaný nasledovne. O každom kruhu v rovine hovoríme že je prázdny, ak neobsahuje vrchol P vo svojej vnútornej časti. Je povolené, ak sa vrchol nachádza na kruhu. Nech u a v sú niektoré dva vrcholy z P . Hrana uv je v $DT(P)$ vtedy a len vtedy, ak existuje prázdny kruh, ktorý prechádza bodmi u a v . Každá hrana splňujúca túto podmienku je Delaunayova hrana. Každá Delaunayova triangulácia množiny vrcholov je unikátna. Opísaná kružnica trojuholníka je unikátny kruh, ktorý prechádza všetkými jeho tromi vrcholmi. Trojuholník sa nazýva Delaunayov vtedy a len vtedy, ak jeho opísaná kružnica je prázdna (neobsahuje žiadne vrcholy). Body ležiace na jednej kružnici sa neberú v úvahu ako body ležiace vo vnútri kružnice, to znamená že spadajú do $DT(P)$. Túto vlastnosť trojuholníka môžeme nazvať podmienkou prázdnej opísanej kružnice Delaunayovho trojuholníka. Tvrdenie že $DT(P)$ je triangulácia a všetky trojuholníky T z $DT(P)$ sú Delaunayove, potom všetky hrany sú Delaunayove, a naopak. Toto tvrdenie platí pretože, ak všetky trojuholníky T sú Delaunayove, potom opísaná kružnica každého trojuholníka je prázdna. Pretože každá hrana patrí nejakému trojuholníku, je každá hrana obsiahnutá v prázdnom kruhu, a je tak Delaunayova [17].

2.3 Nejednoznačnosť určenia polohy bodu

Pri implementácii Delaunayovej triangulácie sa môže vyskytnúť problém pri veľmi veľkých množinách bodov. Algoritmus pre svoje bežanie potrebuje zistiť či bod leží na kružnici, je vo vnútri kružnice, alebo naopak neleží v nej (guľa v 3D). Preto kľúčovým testom používaným v Delaunayovej triangulácii je "inCircle test" (test na polohu bodu voči kružnici).

Štandardný algoritmus pre výpočet zahŕňa stanovenie znamienka determinantu matice 4×4 .

Incircle

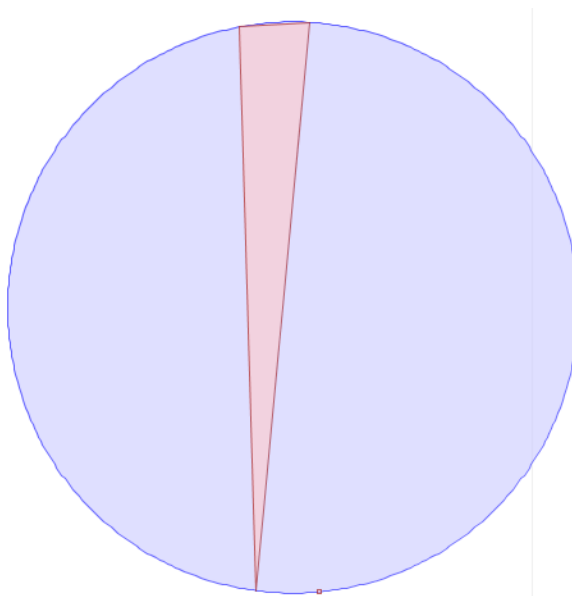


$$\begin{vmatrix} a_x & a_y & a_x^2 + a_y^2 & 1 \\ b_x & b_y & b_x^2 + b_y^2 & 1 \\ c_x & c_y & c_x^2 + c_y^2 & 1 \\ d_x & d_y & d_x^2 + d_y^2 & 1 \end{vmatrix} = \begin{vmatrix} a_x - d_x & a_y - d_y & (a_x - d_x)^2 + (a_y - d_y)^2 \\ b_x - d_x & b_y - d_y & (b_x - d_x)^2 + (b_y - d_y)^2 \\ c_x - d_x & c_y - d_y & (c_x - d_x)^2 + (c_y - d_y)^2 \end{vmatrix}$$

Obr. 3 Výpočet polohy bodu d voči kružnici určenej bodmi abc [19]

Tento výpočet je náchylný k zlyhaniu. Zaokrúhľovaním vzniknú chyby spojené s použitím zvolenej nízkej presnosti použitých dátových typov. To môže spôsobiť že determinant nemusí byť vypočítaný správne [18] [19].

Ako konkrétny príklad by sa dal uviesť bod ležiaci v tesnej blízkosti kružnice.



Obr. 4 Bod voči kružnici [18]

Z tohto pohľadu sa nedá veľmi rozoznať či bod je vo vnútri alebo vonku kružnice. Z bližšieho pohľadu vidíme že bod nie je vo vnútri čiže výsledok `inCircle` by mal byť `FALSE` [18] [19].



Obr. 5 Bod voči kružnici, priblíženie [18]

Počítač však vráti ako výsledok testu funkcie `inCircle` testu (determinantu) hodnotu `TRUE`. Je to preto, že pozícia bodu je určená až na niekoľko desatinných miest a tento veľký rozsah v pomere k veľkosti trojuholníka spôsobí významnú stratu informácií pri zaokrúhľovaní [18] [19].

Riešenie tohto problému spočíva v použití dvojitej presnosti dátových typov. Toto však nanešťastie pri technológii OpenCL nie je až také jednoduché z dôvodu rôznej podpory tohto dátového typu u rôznych zariadení, čiže prenositeľnosti kódu medzi zariadeniami.

2.4 Algoritmy

Existuje mnoho rôznych algoritmov riešenia problému Delaunayovej triangulácie:

- Otáčanie trojuholníkov. Tento proces zahŕňa vytvorenie ľubovoľnej triangulácie bodov z P , a potom prechádzaním hrán trojuholníkov zisťuje či je trojuholník Delaunayeho, ak nie, spraví z neho taký trojuholník. Toto sa opakuje až kým nezostane ani jeden trojuholník, ktorý nie je Delaunayov trojuholník [15].
- Rozdeľ a panuj. Tento prístup zahŕňa rekurzívne kreslenie čiar, ktoré rozdelia body z množiny P do dvoch podmnožín. Každá podmnožina je potom Delaunayovo striangulovaná a nakoniec sú zlúčené pozdĺž deliacej čiary. Rozdeľ a panuj sa zvyčajne používa pre triangulácie v dvoch rozmeroch a bolo preukázané, že najrýchlejší spôsob triangulácie [9][10][11][15].

- Inkrementálne. Inkrementálna triangulácia zahŕňa opakované pridanie bodu z množiny P do množiny už strianguľovaných bodov, táto oblasť sa potom prepočíta a tak je ovplyvnená iba tá časť mriežky kam je pridávaný bod [12][15].
- Sweepline. Tento algoritmus používa priamku, ktorá sa pohybuje od jednej strany plochy k druhej, a pridá každý bod ktorým prejde [13][15].
- Sweep hull. Je to hybridný algoritmus, ktorý používa radiálne sa odvíjajúcu metódu sweepline, ten pridáva body radiálne od nejakého začiatočného bodu z množiny P. To v spojení s preklápaním trojuholníkov dáva ako výsledok Delaunayovu trianguláciu pre množinu bodov P [14][15].

2.5 Euler-Poincaré Formula

Euler-Poincaré rovnica popisuje vzťah medzi počtom vrcholov, počtom hrán a počtom rôznych polygónov. Vzorec je generalizovaný tak aby zahŕňal priehlbiny a diery modelu, prípadne útvary, ktoré prenikajú do telesa. Uvedený Euler-Poincaré vzorec, potrebuje tieto definície (podľa [16]):

- V: počet vrcholov
- E: počet hrán
- F: počet polygónov (trojuholníkov)
- G: počet dier, ktoré prenikajú do telesa, zvyčajne je to vada v topológii mriežky
- S: počet obálok. Pevné teleso samotne je počítané ako jedna obálka. Preto je hodnota S je najmenej 1
- L: počet slučiek, započítavajú sa všetky vonkajšie a vnútorné slučky polygónov. Väčšinou je to číslo rovné počtu polygónov.

Potom Euler-Poincaré vzorec je nasledujúci:

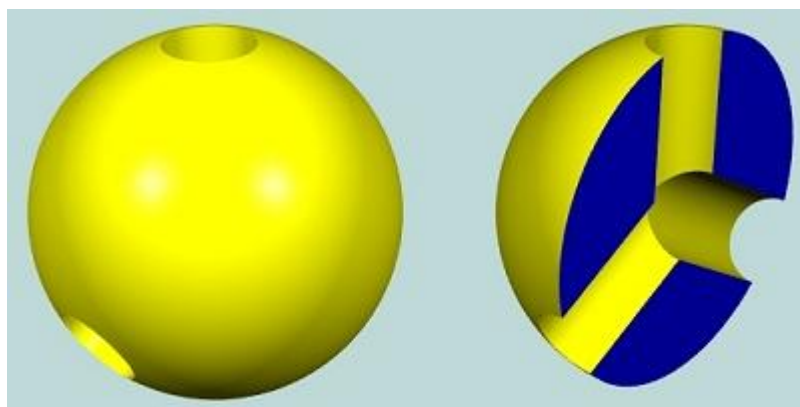
$$V - E + F - (L - F) - 2(S - G) = 0 \quad [16] \quad (4)$$

Napríklad kocka má osem vrcholov ($V = 8$), 12 hrán ($E = 12$) a šesť strán ($F = 6$), žiadne diery, alebo vady, a má jednu obálku ($S = 1$), ale $L = F$, pretože každá hrana (polygón) má jednu vonkajšiu slučku [16]. Preto:

$$V - E + F - (L - F) - 2(S - G) = 0$$

$$8 - 12 + 6 - (6 - 6) - 2(1 - 0) = 0 \quad [16] \quad (5)$$

Nie je vždy ľahké počítať diery (vady) G správne. Predpokladajme napríklad, že máme guľu prederavenú tromi dierami, ako je uvedené na obrázku nižšie. Objekt vľavo ukazuje vonkajší pohľad a napravo je jeho polovičný prierez. Aká je hodnota G ? Vieme, že G hodnota sa počíta ako množstvo vniknutých dier. Ale v tomto prípade, to je trochu dvojznačné. V skutočnosti tým, že kombinuje tri "tunely" budeme mať tri otvory v obálke. Avšak, toto je nesprávne [16].



Obr. 6 Komplikované teleso [16]

Problém sa dá rozložiť na menšie pod problémy, ale čo ak je mriežka komplikovaná, alebo je vopred neznáma? Zo vzorca vyplýva že pomer $V:F:E$ je približne 1:2:3. V práci je empiricky zistené že pre rôzne typy povrchov v 3D je tento pomer trochu odlišný. Pre objekt čajník s počtom vrcholov 2.081 je počet trojuholníkov 8.016 a počet hrán 5.680. Pre náhodne generované mračno bodov s počtom 4.000 bodov, je 23.865 trojuholníkov a 15.951 hrán. Na základe týchto údajov získaných z triangulácie, pri určovaní veľkosti pamäte pre dáta algoritmu, je najvhodnejšie zvoliť pomer tak, aby mierne presahoval teoretické požiadavky. Pre hrany je to 9 a pre trojuholníky je to 7 násobok počtu vstupných bodov do triangulácie. Toto platí ak používame statické údajové pole v OpenCL, naopak v Jave sa dá použiť ArrayList, čo je pole s variabilnou dĺžkou.

3 Paralelná architektúra

3.1 Úvod do paralelných výpočtov na GPU

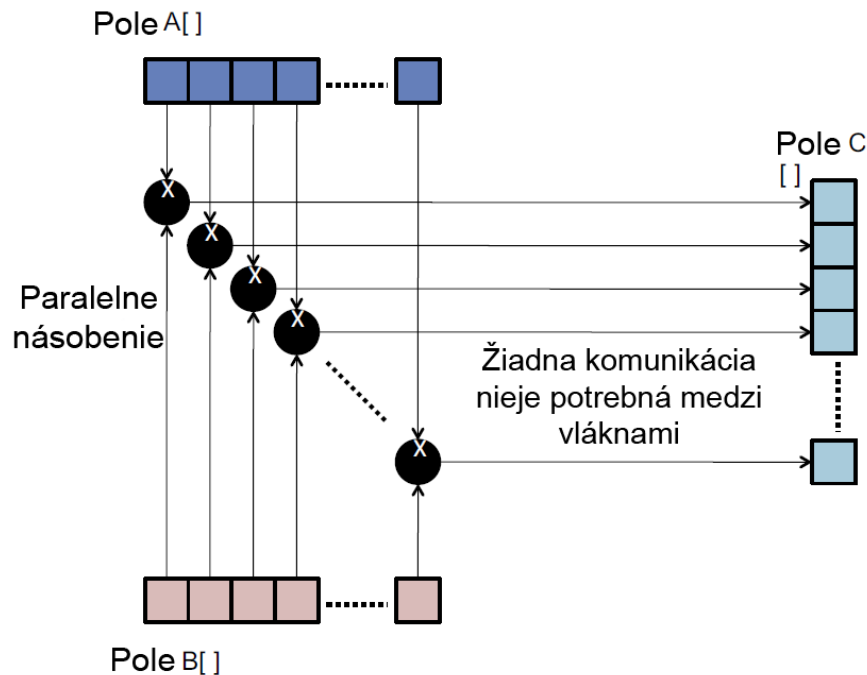
GPGPU (General-Purpose computation on Graphics Processing Units) je skratka pre univerzálne výpočty na grafických výpočtových jednotkách. Grafické výpočtové jednotky GPU, sú vysoko výkonné mnoho-jadrové procesory (v porovnaní s CPU kde je počet jadier 1,2,4 dnes max. 8) so schopnosťou veľmi vysokej výpočtovej a dátovej priepustnosti. Niekedy to boli špeciálne navrhnuté jednotky pre počítačovú grafiku so zložitým spôsobom programovania, dnes sú GPU univerzálne paralelné procesory s podporou prístupných programovacích rozhraní a štandardných jazykov, ako je C [29].

Paralelný výpočet je forma výpočtu, kde je mnoho výpočtov vykonávaných súčasne, založená na princípe, že veľké problémy sa často môžu rozdeliť na menšie, ktoré sú potom riešené súčasne (paralelne). Napríklad výpočet sčítania alebo násobenia dvoch vektorových polí sa môže spustiť paralelne. Násobenie prvkov dvoch polí A a B, kde každé má N prvkov, bude výsledok zapísaný do poľa C na pozíciu zodpovedajúceho prvku [30].

Kód bude vyzeráť takto:

$$C[i] = A[i] * B[i] \quad [30] (6)$$

kde miesto i sa dosadí číslo aktuálneho procesu (identifikátor). Výpočet každého prvku v poli C je nezávislý od všetkých ostatných prvkov tohto poľa. Je znázornený na nasledujúcom obrázku. Paralelizáciou tohto problému, môžeme spustiť jednu inštanciu programu pre jednu výpočtovú jednotku a tá vykoná výpočet jedného prvku C. Tento kód sa dá veľmi ľahko paralelizovať, pretože rovnaká operácia je aplikovaná na všetky prvky poľa A a B potrebných na výpočet C [30].



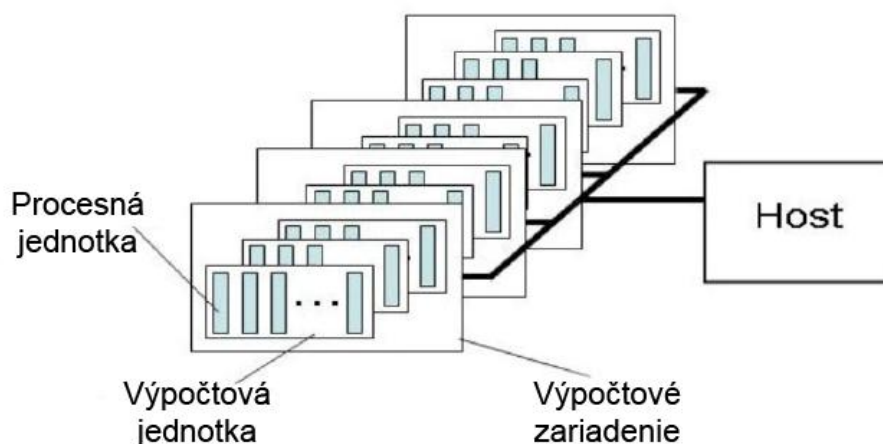
Obr. 7 Paralelne násobenie polí [30]

Asi najznámejšou paralelnou architektúrou umožňujúcou GPGPU výpočty je CUDA (Compute Unified Device Architecture). Je to hardvérová a softvérová architektúra, ktorá umožňuje na GPU spúšťať programy napísané v jazykoch C / C++, FORTRAN alebo programy postavené na technológiách OpenCL, DirectCompute a iných. Táto architektúra je dostupná iba na grafických akceleratoroch spoločnosti NVIDIA, ktorá ju vyvinula. Konkurenčná technológia spoločnosti AMD sa nazýva ATI Stream (po novom AMD Accelerated Parallel Processing, skratka APP). Obe spoločnosti sú tiež členmi Khronos Group, ktorá zaisťuje vývoj OpenCL [20][21].

3.1.1 Architektúra GPU

Drvivú väčšinu plochy čipu grafického akceleratora zaberá veľké množstvo relatívne jednoduchých skalárnych procesorov, ktoré sú organizované do väčších celkov nazývaných stream multiprocessory (procesor na báze SIMD modelu, jedna inštrukcia - viac dát, ktorý umožňuje aplikáciám využiť formu paralelných výpočtov). Vzhľadom k tomu, že sa jedná o SIMT (jedna inštrukcia, viac vlákien) architektúru, riadenie jednotiek a plánovanie inštrukcií je jednoduché a spolu s veľmi malou vyrovnávacou pamäťou zaberá malé percento plochy na GPU čipe. To má bohužiaľ za následok obmedzené predikcie skokov a časté výpadky cache pamäte. Poslednou významnou časťou, ktorá je veľmi podobná CPU je RAM radič [20][21] [22].

Kopírovanie medzi pamäťou host'a a pamäťou GPU zariadenia môže byť zdĺhavé v dôsledku zahltenia prenosovej zbernice medzi nimi.



Obr. 8 Architektúra GPU [25]

Všeobecne sa multiprocesor skladá z niekoľkých stream procesorov, poľa registrov, zdieľanej pamäte, niekoľkých čítacích/zapisovacích jednotiek a Special Function Unit - jednotky pre výpočet zložitejších funkcií ako napríklad sin, cos, ln [20][21].

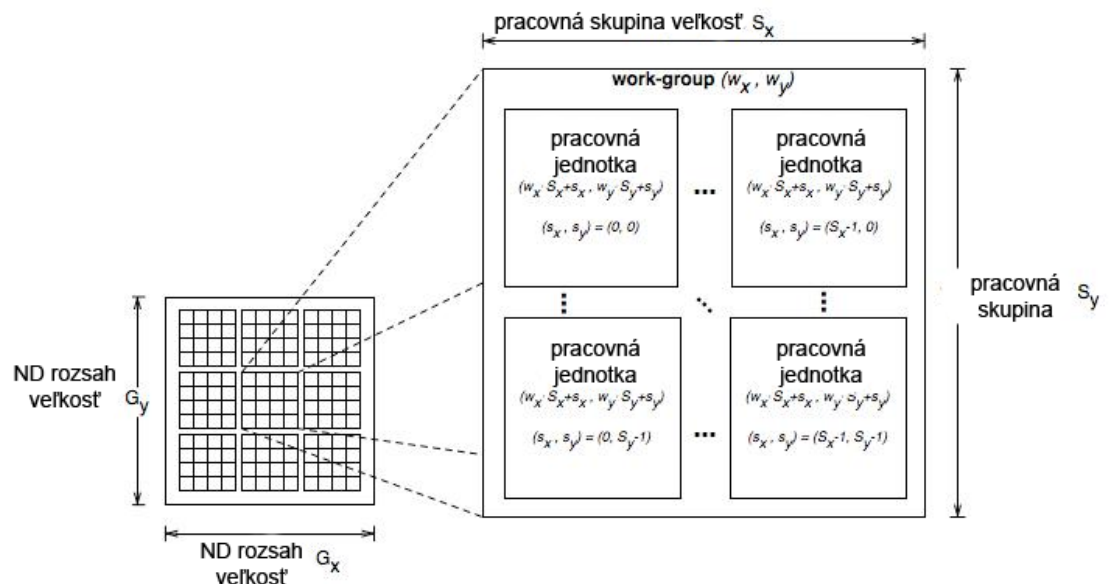
3.1.2 Programovací model GPU

Jadro (kernel - je funkcia vykonávaná jedným vláknom) je základnou výpočtovou jednotkou spustiteľného kódu. Jadrá sa môžu spúšťať v poradí za sebou alebo náhodne, záleží od toho ako je to nastavené v kontexte kódu. Pokiaľ ide o organizáciu, výpočtová doména jadra je definovaná v N-rozmernom výpočtovom priestore. To umožňuje systému vedieť, akú veľkú úlohu ide užívateľ riešiť na jadre a vie rozhodnúť ako má byť použité. Každý prvok v doméne je pracovný prvok GPU a poskytuje možnosť zoskupovať sa do skupín, pre synchronizačné a komunikačné účely [23].

GPGPU aplikácia je zložená z častí, ktoré bežia buď na host'ovi (CPU) alebo na zariadení (GPU). Časti aplikácie bežiace na zariadení sú spúšťané host'om zavolaním kernelu [20].

Organizácia jadier v GPGPU (podľa [20][24]):

- Pracovná jednotka: je základnou výpočtovou jednotkou. Zakaždým, keď jadro je spustené, spustených je mnoho jednotiek (určené programátorom) spustených, každá jedna vykonáva rovnaký kód definovaný kernelom. Každá jednotka má unikátne identifikačné číslo (ID), ktoré je prístupné z kernelu a podľa ktorého môžeme určiť ako budeme dáta spracovávať.
- Pracovná skupina (blok): existujú pre komunikáciu a spoluprácu medzi jednotkami. Je to N-rozmerná mriežka zložená z pracovných jednotiek, teda jednotlivých vykonávaných vlákien, s rozmerom $N = 1, 2$ alebo 3 . Tak ako jednotky, aj skupiny majú jedinečný identifikátor, ktorý môže byť postúpený z jadra. Každý blok vlákien musí byť schopný pracovať nezávisle na ostatných, aby bola umožnená škálovateľnosť systému. Na GPU s viac jadrami pôjde spustiť viac blokov paralelne oproti GPU s menej jadrami kde bloky pobežia v sérii.
- ND-Rozsah: (ND-Range v OpenCL, v CUDE nazvané warp, sú si podobné) je ďalšia úroveň organizácie, špecifikuje ako sú organizované pracovné skupiny (opäť, ako pri N-rozmernej mriežke pracovných skupín, $N = 1, 2$ alebo 3) [20][24].



Obr. 9 Organizácia jadier GPU [24]

Počet a organizácie spustených vlákien v jednom bloku a počet a organizácie blokov v mriežke sa určujú pri volaní kernelu [20].

Typický priebeh GPGPU výpočtu (podľa [20]):

- Vyhradenie pamäte na GPU
- Presun dát z hlavnej pamäte RAM do pamäte grafického GPGPU akcelerátora
- Spustenie výpočtu na grafickej karte
- Presun výsledkov z pamäte grafickej karty do hlavnej RAM pamäte

3.1.3 Pamäťový model GPU

Na grafickej karte je 6 druhov pamäte, s ktorými môže programátor pracovať (podľa [23][20]):

- pole registrov: je umiestnené na jednotlivých prúdových multiprocessoroch a ich rozdelenie medzi jednotlivé procesory plánuje prekladač. Každé vlákno môže pristupovať iba k svojim registrom.
- lokálna pamäť: je používaná v prípade, že dôjde k vyčerpaniu registrov. Táto pamäť je prístupná len jednému vláknu, lenže je fyzicky umiestnená v globálnej pamäti karty a tak je prístup k nej paradoxne pomalší než napríklad k zdieľanej pamäti.
- zdieľaná pamäť: je jedinou pamäťou, okrem registrov, ktorá je umiestnená priamo na čipe stream multiprocessoru (na grafickej karte). Môžu k nej pristupovať všetky vlákna v danom bloku. K zdieľanej pamäti sa pristupuje cez brány nazývané banky. Každá banka môže sprístupniť iba jednu adresu v jednom takte a v prípade, že viacero procesorov požaduje prístup cez tú istú banku, dochádza k pamäťovým konfliktom, ktoré sa riešia jednoduchým čakaním. Špeciálnym prípadom je situácia, keď všetky vlákna čítajú z rovnakej adresy a tak banka hodnotu sprístupní v jednom takte všetkým stream procesorom (tak zvaný broadcast).
- globálna pamäť: je zdieľaná medzi všetkými stream multiprocessormi a neukladá sa do vyrovnávacej pamäte.
- pamäť konštánt: je pamäť len na čítanie, je zdieľaná rovnako ako globálna pamäť s tým rozdielom, že je pre ňu na čipe multiprocessoru vyhradená L1 cache. Podobným spôsobom ako zdieľaná pamäť umožňuje rozosielenie výsledku broadcastom.

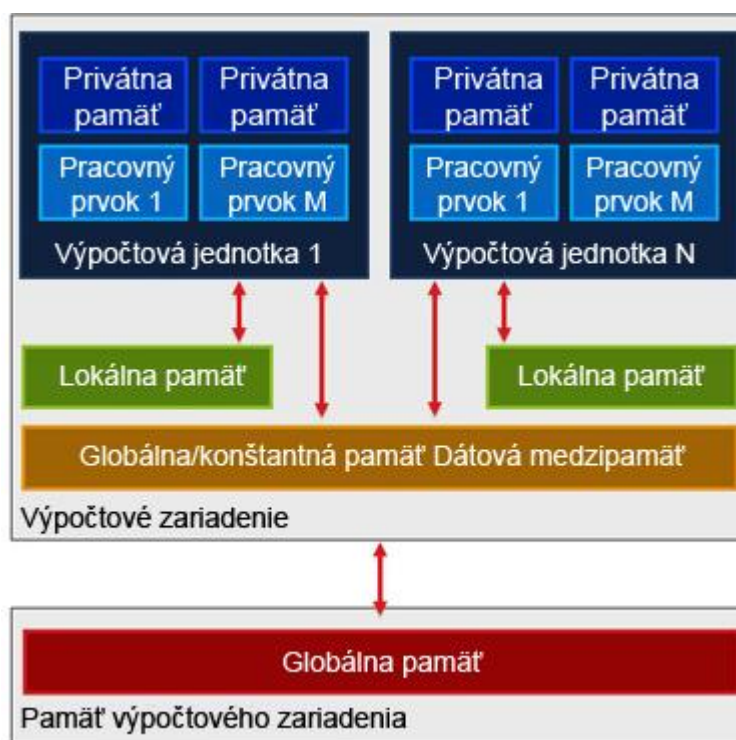
- pamäť textúr: je taktiež zdieľaná, je určená pre čítanie a má vyrovnávaciu pamäť. Je optimalizovaná pre 2D priestorovú lokalitu, takže vlákna sú v rovnakej skupine, a tak čítaním z blízkych textúrovacích súradníc dosahujú najlepšie výkony [23][20].

Hierarchia pamäte pre technológiu CUDA a OpenCL je založená na tomto modeli. Programátor si vie do istej miery určiť v kontexte ako a s ktorou pamäťou bude narábať. Presný popis pamäťovej architektúry OpenCL je presnejšie popísaný nižšie.

3.2 OpenCL

OpenCL je štandardný priemyselný framework pre programovanie počítačov zložených z kombinácie CPU, GPU a ďalších procesorov. Tieto takzvané heterogénne systémy sa stali dôležitou skupinou platforiem, a OpenCL je prvý priemyselný štandard, ktorý sa priamo zaoberá ich využitím. OpenCL je relatívne nová technológia. Prvý krát bola vydaná v decembri 2008. V OpenCL sa môže napísať jeden program, ktorý bude spustiteľný na širokom spektre systémov, od mobilných telefónov, cez notebooky až po masívne superpočítače. Žiadny iný paralelný programovací Standard nemá taký široký dosah. To je jeden z dôvodov, prečo je OpenCL tak dôležité, a má potenciál zmeniť softvérový priemysel [31].

Špecifikácia jazyka OpenCL 1.1 (pozri [26]) je akousi prvou vrstvou anatómie OpenCL systému. Popisuje syntax a programovacie rozhranie pre písanie výpočtových jadier, bežiacich na podporovaných GPGPU, ako napríklad AMD GPU, CUDA GPU a na viacjadrových procesoroch. Použitý jazyk je založený na podmnožine jazyka ISO C99. Jazyk C bol vybraný ako základ pre prvé OpenCL jadro vzhľadom k jeho častému použitiu a dobrej znalosti v komunite vývojárov. Pre dosiahnutie rovnakých výsledkov na rôznych platformách, je definovaná norma numerickej presnosti pre všetky operácie s pohyblivou desatinnou čiarkou a tiež veľká množina funkcií. Autor má možnosť vopred skompilovať OpenCL jadro (kernel), alebo nechať OpenCL aby kernel skompilovalo až pri behu aplikácie. Druhá, platformová medzivrstva, dáva vývojárovi prístup k počtu a druhom zariadení v systéme. Vývojár potom môže vybrať a inicializovať potrebné výpočtové zariadenie a tak rozložiť záťaž medzi ne. Práve v tejto vrstve sa vytvára kontext prostredia OpenCL aplikácie a vytvárajú sa požiadavky pre vykonanie kernelu a prenos dát do GPU. Napokon, tretia vrstva umožňuje vývojárom zaradiť kernely do fronty na výpočet a je zodpovedná za správu pamäťových zdrojov v OpenCL systéme [23].



Obr. 10 Hierarchia pamäte OpenCL [24]

OpenCL definuje viacúrovňový pamäťový model s pamäťou od privátnej pamäte viditeľnej iba jednotlivým výpočtovým jednotkám v zariadení až po globálnu pamäť, ktorá je viditeľná pre všetky výpočtové jednotky v zariadení. V závislosti na aktuálnom pamäťovom subsystéme, rôzne pamäťové priestory môžu byť zlúčené dokopy. OpenCL definuje 4 pamäťové priestory: privátne, lokálne, pamäť konštant a globálne. Privátna pamäť je pamäť, ktorá môže byť použitá iba jednou výpočtovou jednotkou. Je to podobné registrom ako pri CPU. Lokálna pamäť je pamäť, ktorá môže byť použitá pracovnými jednotkami z pracovnej skupiny. Pamäť konštant je pamäť, ktorá môže byť použitá pre ukladanie dát konštant pre čisto čítací prístup, všetkým výpočtovým jednotkám počas vykonávania jadra. Procesor host'a (CPU) je zodpovedný za pridelovanie a inicializáciu pamäťových objektov, ktoré sa nachádzajú v tejto pamäti. Globálna pamäť je pamäť, ktorá môže byť použitá vo všetkých výpočtových jednotkách na zariadení GPU [23][20].

3.3 Zrýchlenie a efektívnosť

Nech $T(n, 1)$ je sekvenčný čas vykonávania na jednom procesore a $T(n, p)$ je čas vykonávania na p procesoroch, pri riešení toho istého problému veľkosti n . Zrýchlenie $S(n, p)$ a efektívnosť $E(n, p)$ sú definované nasledujúcimi vzťahmi:

$$S(n, p) = \frac{T(n, 1)}{T(n, p)} \quad [38] \quad (6)$$

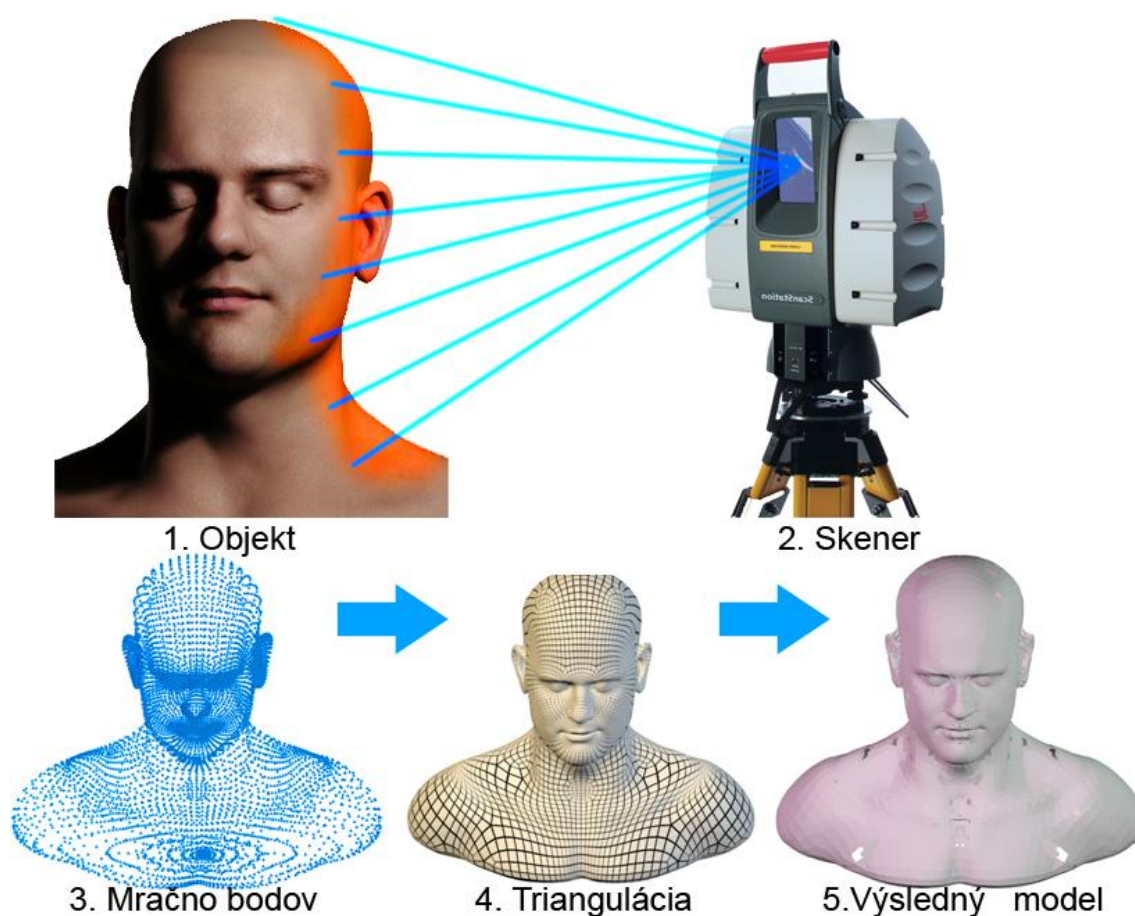
$$E(n, p) = \frac{S(n, p)}{p} \quad [38] \quad (7)$$

V ideálnom prípade očakávame $S(n, p) = p$ a $E(n, p) = 100 \%$. V praxi však platí $S(n, p) < p$, t.j. $E(n, p) < 100 \%$. Preto ak je problém nedostatočne škálovateľný, zvýšenie počtu procesorov už nevedie k zvýšeniu zrýchlenia. Preto pri výbere paralelnej architektúry je potrebné zvážiť, pre aké druhy paralelných problémov bude určená, jej naddimenzovanie neznamena automaticky efektívne riešenie paralelných problémov v budúcnosti [38].

4 Návrh systému

4.1 Motivácia

Pokroky v hardvéri počítača dnes umožňujú zobraziť 3D model skoro na každom dostupnom počítači. Toto bolo možné pred pár rokmi iba na super počítačoch. Táto práca sa popisuje proces ako dostať z informácie získanej 3D skenerom (mračno bodov) výsledný model za pomoci takmer každého dostupného počítača s prípadným využitím GPU karty.



Obr. 11 Proces skenovania a rekonštrukcie povrchu modelu

4.2 Požiadavky

Cieľom tejto práce je navrhnuť a implementovať programové vybavenie pre proces triangulácie mračna bodov s využitím technológie GPGPU.

Požiadavky na navrhovaný program:

- program musí vhodne spracovať načítané údaje zo vstupného súboru. Súborom bude zoznam bodov s ich koordinátami v 3D. Tieto údaje bodov sú získané z existujúceho modelu alebo zo skenovacieho zariadenia.
- na základe načítaných dát zo súboru musí program generovať výslednú mriežku povrchu modelu. Tento objekt je potrebné uložiť do niektorého zo štandardných formátov pre ukladanie 3D modelov.
- program musí mať implementovanú voliteľnosť medzi vykonaním algoritmov na CPU alebo na GPU.
- program musí obsahovať možnosť pre nastavenie konštánt, ktoré sú použité pri triangulácii a tak podľa požiadaviek ovplyvniť výsledný súbor.
- program musí mať okrem možnosti načítať existujúci súbor, aj možnosť generovať náhodné body v priestore.

Vstupné dáta budú vo formáte Obj (pozri Formát Obj, strana 48). Každý súbor bude obsahovať sadu bodov s tromi súradnicami x , y a z vo forme reálneho čísla. Uložené a spracovávané budú v dátových typoch s pohyblivou rádovou čiarkou dvojitej presnosti.

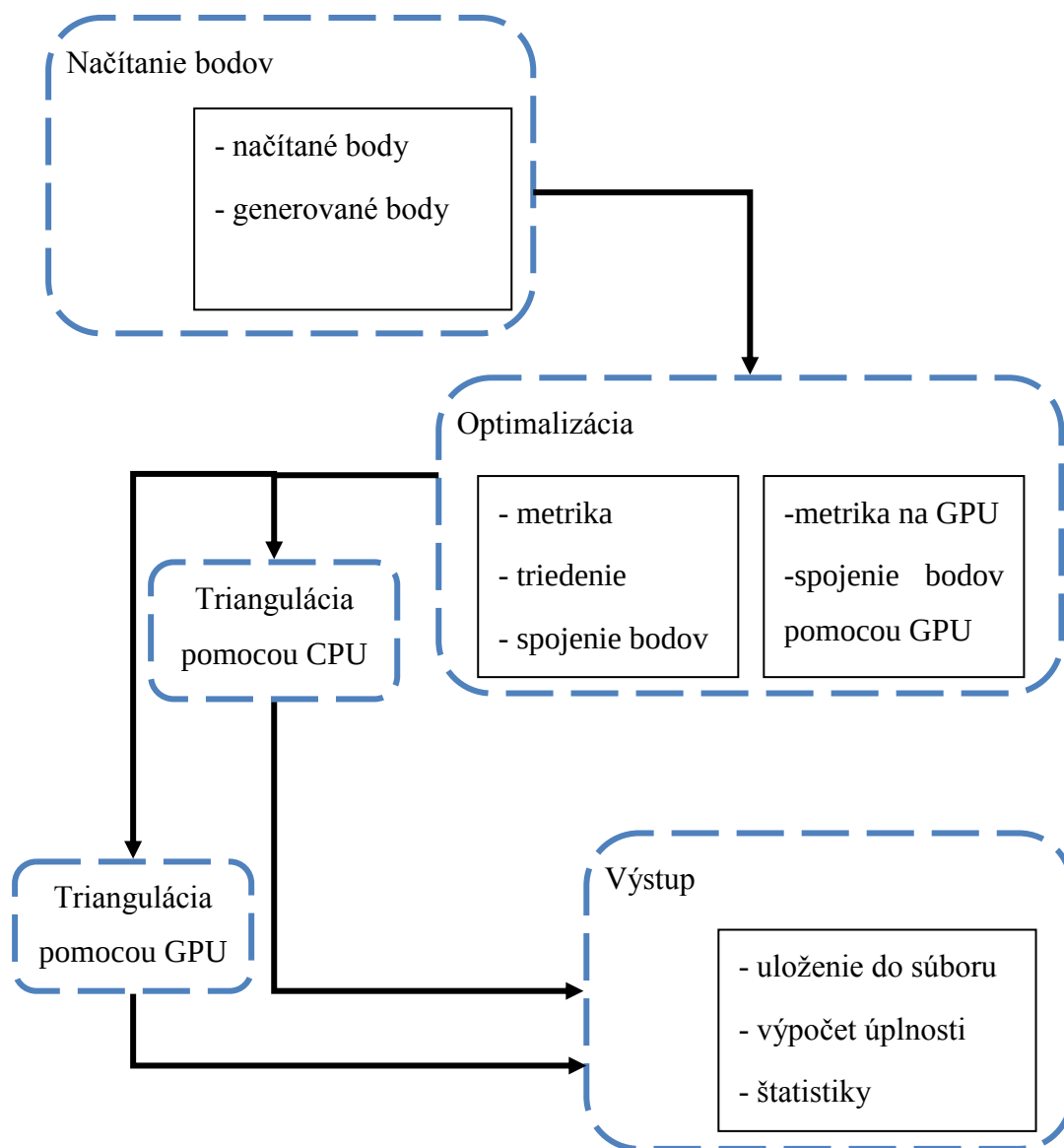
Export dát bude možný do štandardných formátov súborov pre ukladanie 3D mriežok. Tieto formáty sú Obj, ASE, WRL.

Jazyk aplikácie je Java, je to objektovo orientovaný programovací jazyk vychádzajúci zo syntaxe C (resp. C++). Jednou z najväčších výhod Javy je jej nezávislosť na platforme. Program napísaný v jazyku Java sa nekompile priamo do strojového kódu ale do tzv. bajt kódov, ktoré sú následne interpretované virtuálnym strojom (Java Virtual Maschine). Vďaka tomuto prístupu sú programy napísané v Jave, resp. v niektorej z jej verzií (súbor týchto verzií tvorí ucelenú platformu Java), možné spúšťať na širokej množine tried zariadení a počítačových platform. Java API poskytuje rozsiahlu kolekciu tried, ktoré je možné okamžite využívať pri tvorbe programov [34].

Aplikácia využíva ako GPGPU technológiu OpenCL. OpenCL je štandardný priemyselný framework pre programovanie heterogénnych systémov.

Prepojovacia medzivrstvá medzi platformami Java a OpenCL je JOCL previazanie. Táto knižnica ponúka Java väzby pre OpenCL, ktoré sú veľmi podobné pôvodnému OpenCL API. Funkcie sú poskytované ako statické metódy, a sémantika a

podpisy týchto metód boli vedené v súlade s pôvodnými funkciami knižníc, s výnimkou zvláštnych obmedzení jazyka Javy [35].



Obr. 12 Prehľadová schéma implementácie

4.3 Definovanie pojmov pri reprezentácii údajov

Bod je definovaný súradnicami osi x , y a z . Každý bod má svoje jedinečné identifikačné číslo (budeme ho označovať bod $p(i)$ z množiny bodov P), dve hodnoty metriky a príznak použitia bodu. Hrana je definovaná ľavým a pravým bodom. Ďalej sa bude označovať ako hrana $e(i)$ z množiny E . Každá hrana má definované práve dva

body, určené ich identifikátorom. Polygón, ďalej nazývaný trojuholník, obsahuje informáciu o troch bodoch ktoré ho tvoria. Sú to vrcholy $v(1)$, $v(2)$ a $v(3)$.

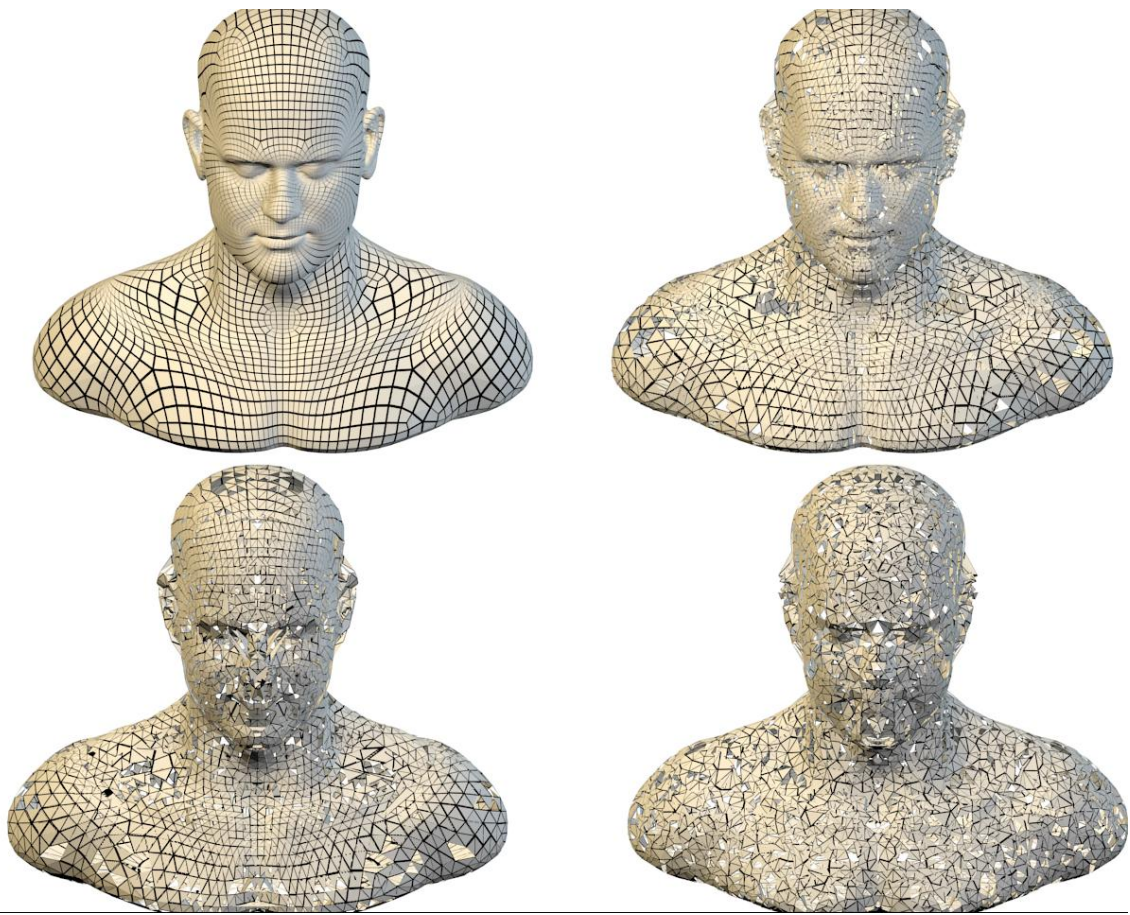
4.4 Optimalizácia

Fáza optimalizácie je dôležitým časťou pred výkonom samotnej triangulácie. Sú tu zahrnuté rôzne procedúry optimalizácie, ako spojenie blízkych a totožných bodov, metrika bodov a nakoniec aj spôsob určenia vhodného prvého bodu triangulácie.

4.4.1 Blízke body

Ak sa v množine bodov P získaných zo skenovacieho zariadenia nachádzajú body ležiace na tom istom mieste alebo na miestach k sebe veľmi blízkych, z pohľadu výslednej triangulácie je možné takéto body spojiť, bez toho aby to výrazne degradovalo kvalitu výsledného povrchu. V aplikácii užívateľ môže zadať toleranciu spájania blízkych bodov. Toto číslo vyjadruje maximálnu vzdialenosť medzi dvoma susednými bodmi, ktoré budú spojené. Novo vytvorený bod, z takto spojených blízkych bodov, leží medzi nimi v priestore. Staré body sú odobraté z množiny P a nový bod je pridaný do tejto množiny. Porovnáva sa každý bod s každým. Tento proces sa musí vykonať pred trianguláciou, aby bral do úvahy spojené body, a musí sa vykonať ešte aj pred výpočtom metriky, pretože metrika vypočíta parametre bodov použitých v triangulácií (ak by sa spojili body potom, novo vzniknuté body by nemali vypočítanú a priradenú metriku a zároveň by bola zbytočne počítaná metrika pre body ktoré sa z množiny odstránia). Spojenie blízkych bodov priamou úmerou urýchľuje výpočet triangulácie v závislosti od počtu spojených bodov.

Spojenie blízkych bodov sa dá využiť aj pre značné urýchlenie výpočtu, čo má za následok veľkú degradáciu kvality povrchu. Hodnota tolerancie je závislá od vzájomnej pozície bodov naskenovaného modelu. Pre každú množinu môže byť veľmi odlišná. Nasledujúci obrázok ilustruje ako je kvalita a čas (za ktorý dosiahneme výsledok) závislá od tolerancie. Model dostupný z [37].



Obr. 13 Spájanie blízkych bodov, model "hlava" (horný rad zľava: pôvodný model, tolerancia 0,02 s časom 44 minút, dolný rad: tolerancia 0,04 s časom 9 minút, tolerancia 0,1 s časom 5 minút)

4.4.2 Metrika

Algoritmus triangulácie závisí na vstupnom poli prvkov. Toto pole je množina neusporiadaných bodov P v 3D. Práve pre neusporiadanosť bodov je algoritmus nútený zakaždým hľadaním, napríklad najbližšieho prvku, prechádzať toto celé neutriedené pole. Pole môže mať rádovo niekoľko desiat tisíc bodov. Ak by mal napríklad proces hľadania najbližšieho prvku v každom cykle prechádzať celé pole odznova, výpočet celej triangulácie by bol neefektívny. Práve z toho dôvodu sa prideli každému bodu akási jeho hodnota, alebo jeho kvalitatívne ohodnotenie v rámci množiny P aby už potom nemusel tento výpočet opakovane vykonávať. Toto ohodnotenie je nazvané metrika. Metrika má niekoľko parametrov:

- minimálna vzdialenosť,
- priemerná vzdialenosť,

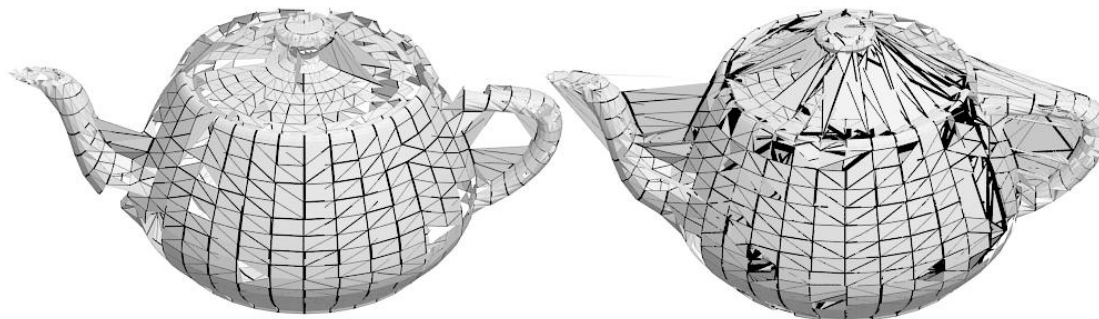
- príznak použitia bodu.

Každý z týchto parametrov sa vypočítava na začiatku triangulácie, ešte pred jej samotným spustením. Minimálna vzdialenosť je vzdialenosť konkrétneho bodu k bodu jemu najbližšiemu z celej množiny bodov P. Túto vlastnosť, v kombinácii s konštantami zadanými užívateľom aplikácie, vieme využiť pri triangulácii a hľadanie vhodnejšieho bodu - kandidáta na striangulovanie s vybratou hranou. Takýmto výberom, alebo obmedzením, vieme dosiahnuť kontrolu nad algoritmom a máme tak istú mieru voľnosti pri tvorbe výsledného objektu. Algoritmus ktorý ku každej hrane hľadá bod s ktorým by vytvoril trojuholník musí najprv prejsť takto vytvorenou podmienkou a až potom sa dostáva k vyhľadaniu vhodnejšieho kandidáta. Táto podmienka bude nazvaná, *prvou podmienkou maximálnej vzdialenosti*. Ak vyšetovaný bod s hľadajúcou hranou by tvorili Delaunayov trojuholník, ale bod by nespadal do tejto podmienky, algoritmus ho jednoducho preskočí. Body v oblasti s vyššou hustotou výskytu majú túto hodnotu menšiu. Táto podmienka zabraňuje vytvoreniu akýchsi anomálií v trojrozmernej triangulácii, napríklad pri tvorbe objektu akým je rúra, nedovolí spraviť trojuholník tak aby sa nachádzal v priestore dutiny, aj keď je to správny Delaunayov trojuholník. V aplikácii je prednastavená, empiricky zistená, konštanta najvhodnejšia pre väčšinu povrchov (napríklad typu čajník, hlava, alebo aj skenovaná budova). Konštanta vynásobená s parametrom minimálnej vzdialenosti bodu je maximálna vzdialenosť bodu od hrany s ktorou strianguluje. Ak sa konštanta zväčší bude toto číslo väčšie a tak aj viacero vzdialenejších bodov spadne do tejto podmienky. Pri veľmi detailných a komplikovaných tvaroch objektov odporúčam dávať nižšie konštanty, avšak nie menšie ako 1 (pretože by nestriangulovalo ani s najbližším susedom). Konštantu je potrebné obmeňovať, pri ďalšom spustení aplikácie, podľa spokojnosti a splnenia očakávaní od výsledku triangulácie.

Algoritmus nemá vedomosť o tvare modelu, hustote bodov alebo ich pravidelnosti vzdialeností medzi nimi, no napriek tomu je postavený tak aby v čo najvyššej miere tvoril plochy na seba nadväzujúce a aby keď sa trianguluje po povrchu objektu nestalo, že skočí na nejakú odľahlú pozíciu (ktorej vzdialenosť je väčšia ako vzdialenosť stanovená *prvou podmienkou maximálnej vzdialenosti*) v priestore a tak zničí kontinuitu výsledného povrchu.

Nasledujúci obrázok ilustruje rozdiel medzi nízkymi hodnotami zadaných konštánt (vľavo) a vysokými hodnotami (vpravo). Z obrázku ukážkovo vidno ako konštanty

vplývajú na výsledný povrch modelu, vytvárajú primálo, respektíve priveľa trojuholníkov.

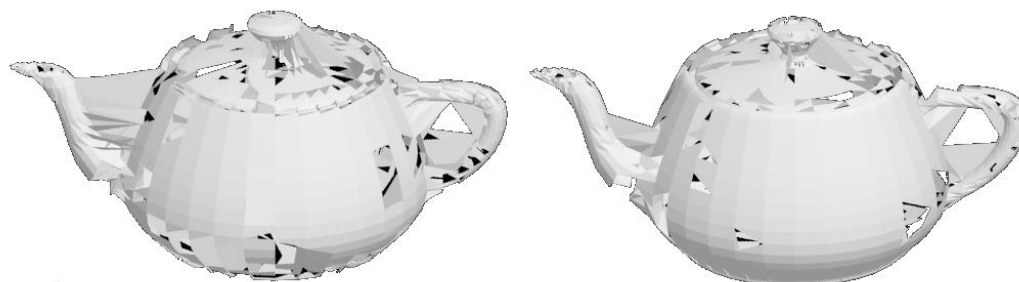


Obr. 14 Porovnanie povrchov s rôznymi vstupnými konštantami

Modely môžu mať rôznu hustotu bodov, napríklad v kritických oblastiach detailnejšieho spracovania modelov. Naopak pri rovinných častiach toho istého modelu, časti kde je nedostatok detailov je nepotrebná vysoká hustota. Z hľadiska optimalizácie modelovania, manipulácie s nimi a ukladania takýchto modelov je to priaznivý fakt. Z hľadiska rekonštrukcie povrchu je tento fakt negatívny, pretože nedokážeme povedať kde je hustota vyššia. Riešením je použitie metriky. Body v oblasti s vyššou hustotou majú parameter minimálnej vzdialenosti nižší ako ostatné body mimo tejto oblasti. To dáva parametru najlepšie vyhliadky pre použitie v algoritme pre kontrolu plynulosti povrchu.

Ďalším parametrom je priemerná vzdialenosť. Ako už z názvu vyplýva, je to priemerná vzdialenosť od všetkých bodov množiny P . Je to jednoduchý súčet všetkých vzdialenosti bodu od bodov množiny vydelený ich počtom. Taktiež tento parameter sa využíva v kombinácii s konštantou zadanou užívateľom a má také isté využitie ako prvý parameter. Avšak na rozdiel od minimálnej vzdialenosti má takto určená podmienka na vyhľadávanie vhodnejšieho kandidáta horšie výsledky. Svoje uplatnenie však našiel pri triedení bodov mračna. Podľa neho sa porovnávajú body a tak sa triedi mračno. Bod ktorý má najmenšiu priemernú vzdialenosť je najvhodnejší ako bod začiatku triangulácie. Je to tak preto, lebo teoreticky má tento bod najbližšie ku všetkým bodom, to znamená že je niekde uprostred modelu. Naopak bod s najvyššou hodnotou parametra je najvzdialenejší bod modelu, odľahlý, okrajový bod.

Ako vidno z nasledujúceho obrázka nedokončenej triangulácie, čajník napravo s použitím minimálnej vzdialenosti ako metriky určujúcej vhodnejší prvok, je povrch modelu celistvejší a už na prvý pohľad obsahuje menšie množstvo artefaktov.



Obr. 15 Porovnanie parametrov výberu bodu do triangulácie (priemerná vzdialenosť naľavo, minimálna vzdialenosť napravo)

Posledný parameter, príznak použitia bodu, je veľmi dôležitý. Rozdeľuje skupinu prvkov P na polovicu, na časť použitých bodov a na časť bodov nepoužitých. Použitý bod znamená, že bod je už obsiahnutý v mriežke, čiže je časťou trojuholníka. Bod nepoužitý je ak neje časťou grafu, to je ak neexistuje taká hrana ktorej jedným z bodov hrany by bol tento bod. Algoritmus triangulácie sa preto tiež delí na polovice. Prvá polovica musí prebehnúť pred druhou. Prvá hľadá body už použité, preto aby ich mohla obsiahnuť v tejto časti triangulácie skôr ako sa budú vyhľadávať nové, druhá hľadá body nepoužité. Čiže ku hrane sa najprv hľadá použitý bod z množiny použitých a ak sa nenájde, tak sa následné hľadá z množiny bodov nepoužitých.

4.4.3 Výber štartovacieho bodu

Výber štartovacieho bodu je dôležitou procedúrou pred začatím algoritmu triangulácie. Tento výber je voliteľnou položkou pri behu programu. Výber vyberá z množiny vstupných bodov, bod najvhodnejší pre začiatok triangulácie. Teoreticky by to mal byť bod uprostred poľa bodov modelu, pretože má zo všadiaľ najbližšie. Znamená to že ak sa triangulácia začne odvíjať od neho, tak algoritmus skôr nájde vhodnejší bod z poľa. To platí hlavne z pohľadu uloženia bodov v pamäti a nie bodov v 3D, pretože body sú prechádzané postupne, ak neboli utriedené, v poradí v akom boli načítané. Teda algoritmus bude rýchlejšie vykonaný a tým pádom efektívnejší. Tento štartovací bod sa

dá nájsť jednoducho. Ako bolo už spomenuté pri metrike, je to prvý bod z utriedenej množiny P , teda bod s najnižšou priemernou vzdialenosťou od ostatných bodov.

Niekedy nie je vhodné zvoliť si ako štartovací bod, bod najvýhodnejší podľa jeho metriky, a to hlavne pri utriedenom poli prvkov. Paradoxne by sa mohlo zdať že je to najvhodnejšia možnosť ako začať trianguláciu, ale v praxi sa osvedčilo pravý opak. Ak začneme triangulovať od najvhodnejšieho bodu a začneme hľadať bod s ktorým strianguluje, bude to bod jemu najbližší v 3D priestore, ale aj v poradí vo fronte bodov (z pohľadu uloženia dát v pamäti). Preto program sám pri zapnutí možnosti triedenia poľa zapne zároveň možnosť náhodného štartovacieho bodu.

Pri paralelnej triangulácií sa neoplatí začínať z jedného bodu, respektíve trojuholníka a preto algoritmus v tomto prípade hľadá body tri. Tri optimálne body by teoreticky mali byť body od seba navzájom najviac vzdialené ale zároveň by mali byť blízko strednej časti množiny bodov modelu. Ak nepoznáme štruktúru vstupných bodov, v zmysle ich usporiadania, je ťažko predvídať ako zvoliť tieto body. Preto prvý bod je bod najvýhodnejší podľa jeho metriky. Bod druhý, je bod z polovice poľa. Je to v podstate náhodný bod vybraný zo stredu fronty načítavaných bodov. Bod tretí je bod z konca poľa. To znamená že je to bod ktorý bol načítaný ako posledný. Taktiež je to bod náhodný. Ako bolo spomenuté vyššie, náhodné zvolené body sa paradoxne osvedčili a majú svoje opodstatnenie v tomto algoritme.

4.5 Algoritmus triangulácie

Triangulačný algoritmus je založený na inkrementálnej metóde. Môže sa zdať že vyzerá akoby používal radiálne sa odvíjajúcu metódu, podobne ako pri vyššie uvedenej Sweep Hull metóde. Podstatou algoritmu je na nájsť trojuholník k hrane. Ten sa hľadá nasledovne. K prvému vybranému bodu $p(1)$ predchádzajúcou metódou sa nájde najbližší bod $p(2)$ s ktorým sa vytvorí hrana $e(1)$ a zapíše sa do množiny hrán E . Body sa označia ako použité. V nasledujúcom kroku sa k tejto hrane hľadá taký bod $p(3)$ aby vyhovoval podmienke Delaunayovho trojuholníka. Vytvorením hrán z bodov $p(1)$ - $p(3)$ a $p(2)$ - $p(3)$ sa vytvorí trojuholník $t(1)$, ktorý sa pridá do množiny trojuholníkov T . Body sa označia ako použité. Po vytvorení prvého trojuholníka nasleduje rekurzívna časť. Tá prechádza pole všetkých doposiaľ vytvorených hrán E a ku každej hrane hľadá bod s ktorým strianguluje. Vyhľadáva sa najprv z množiny bodov už použitých (takých, ktoré sú už obsiahnuté v mriežke, čiže sú časťou trojuholníka). Každý bod $p(i)$ vyhovujúci Delaunayovej podmienke sa otestuje najprv na *prvú podmienku maximálnej vzdialenosti*

a potom sa porovná s ostatnými bodmi ktoré takisto prešli predchádzajúcimi dvoma podmienkami (toto porovnanie je bližšie opísané v časti 4.7 Výber kandidátov pre trianguláciu na strane 46).

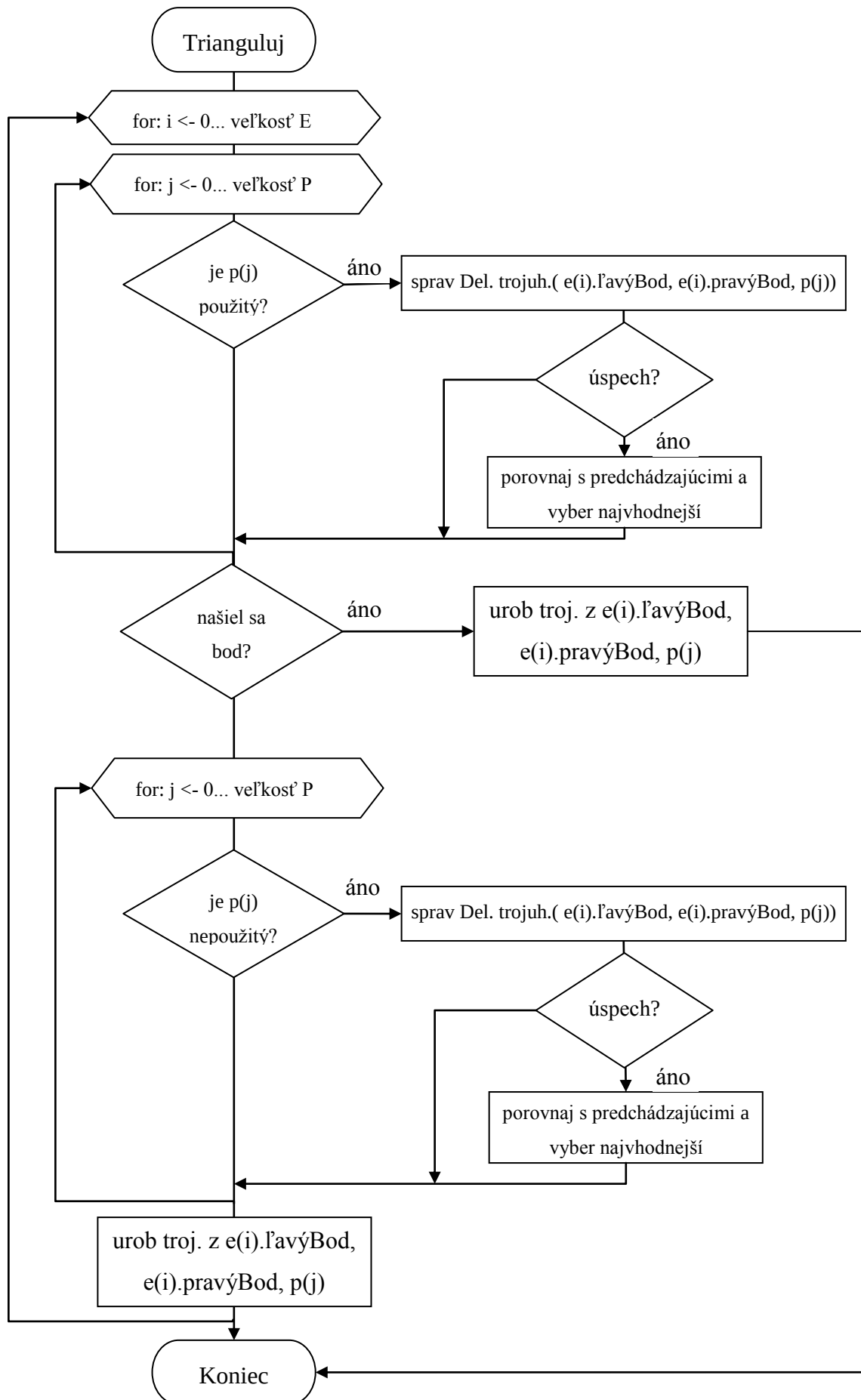
Parameter minimálnej vzdialenosti bodu a druhá užívateľom zadaná konštanta vytvárajú *druhá podmienku maximálnej vzdialenosti*, ktorá je použitá tvorbe trojuholníka. Ak prejde bod podmienkou, tak v tom prípade sa z vybraného bodu $p(i)$ a bodov hrany $e(n)$ vytvorí trojuholník t (za predpokladu že neexistuje). Bod a hrana sa označia ako použité.

Ako bolo spomenuté v prvej časti algoritmu, algoritmus nevie aký je tvar povrchu modelu, no zároveň je postavený tak aby v čo najvyššej miere tvoril plochy na seba nadväzujúce. Tak isto tu platí nejaké obmedzenie, presnejšie obmedzenie *druhou podmienkou maximálnej vzdialenosti*. Táto podmienka odfiltruje všetky vadne vytvorené trojuholníky (z procesu triangulácie, ešte počas jej behu) aby sa nestalo že nejaká takto vadne vzniknutá hrana nestriangulovala s bodom a nezničila by plynulosť povrchu.

Ak sa nenájde bod v prvej časti, tak sa následné hľadá z množiny bodov nepoužitých. Postup je rovnaký.

Body sú prechádzané postupne, v poradí v akom boli načítané, alebo utriedené. Hrany sa prechádzajú postupne v poradí v akom boli vytvorené. Na začiatku sú to tri hrany ku ktorým sa postupne hľadá nový trojuholník. Keďže sa hrany prechádzajú postupne za sebou, nevznikne prípad že sa použije viackrát jedna hrana.

Algoritmus končí ak už sa nenájde bod s ktorým by strianguloval, z čoho vyplýva že už sa nevytvorí nová hrana a tak sa terminuje hlavný cyklus.

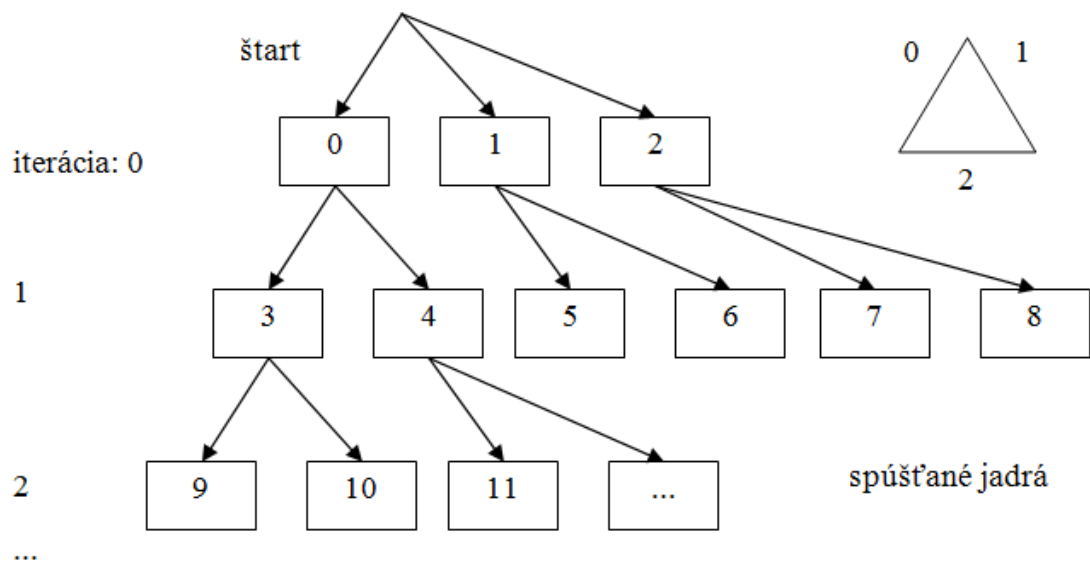


Obr. 16 Algoritmus Delaunayovej triangulácie

4.5.1 Paralelný algoritmus triangulácie

Problém dekompozície je prvý krok k paralelizácii algoritmu. Daný algoritmus je potrebné rozdeliť na menšie úlohy. Riešenie je nasledovné. Pri triangulácii na CPU sa hľadá trojuholník k hrane a prechádza pole hrán postupne po jednom. Pri paralelnej triangulácii každý prvok poľa prečíta 1 jadro a tak sa prejde pole naraz. Nanešťastie je to obmedzené počtom jadier na procesore GPU. Ak sa vyčerpá počet voľných jadier na výpočet, manažment karty jednoducho čaká až sa nejaké uvoľní, a potom spúšťa úlohy sériovo.

Spúšťa sa toľko jadier, koľko je hrán v mriežke. Keďže nepoznáme dopredu počet hrán, potrebujeme teoreticky vypočítať koľko ich bude, aký priestor im vymedziť a koľko jadier spustiť. Na to nám posluží rovnica (4). Každému jadru je priradená jedna hrana ku ktorej bude hľadať bod s ktorým strianguluje. Jadru je priradené identifikačné číslo v poradí v ktorom bolo spustené. Podľa tohto čísla identifikuje ktorú hranu bude to ktoré jadro riešiť. Jadro na začiatku čaká, až jemu priradená hrana sa stane voľnou pre výpočet, to znamená vytvorenou z predchádzajúceho výpočtu. Pri prvých troch trojuholníkoch to je deväť hrán (očíslované 0-8) a tak sa spustí prvých deväť jadier s výpočtom. Ostatné jadrá ostávajú čakať na "svoju" hranu, alebo až do času kým nevyprší v jadre implementovaný časovač. Tieto jadrá vytvoria nové hrany a tak sa paralelne spustia ďalšie nové jadrá.



Obr. 17 Paralelne vytváranie nových hrán a zároveň spúšťanie procesov

Paralelný algoritmus je rovnaký algoritmus ako sériový, s pár rozdielmi. Ako bolo spomenuté pri hľadaní štartovacieho bodu pre paralelný algoritmus, nájdú sa tri začiatkové body, z ktorých sa sekvenčne na CPU vypočítajú tri štartovacie trojuholníky. Tieto trojuholníky sa zapíšu do pola v pamäti a pošlú sa na výpočet do GPU. Pre paralelný výpočet je potrebné GPU poslať údaje o:

- bodoch: pozícia v 3D, parameter použitia, parameter minimálnej vzdialenosti
- hranách: ľavý bod, pravý bod, parameter existencie hrany (t.j. informácia o stave vytvorenia hrany),
- trojuholníkoch: tri body.

Tieto údaje musia byť, z hľadiska pamäte, uložené v globálnej pamäti. Akákoľvek iná možnosť, v tomto prípade nemá význam. Je tomu tak z dôvodu rýchlosti tejto pamäte a potreby viditeľnosti dát v pamäti, konkrétne pri čakaní jadier na vytvorenie hrany. Všeobecne (nezávislé od platformy) zdieľaná a globálna pamäť sú rýchlejšie ako lokálna (spomenuté na strane 29 a podľa [20] [23]), preto sa ju nevyplatí využiť.

Pri tvorení nových hrán alebo trojuholníkov je potrebné dať pozor na ich očíslovanie. Keďže každé jadro beží paralelne a naraz sa vytvorí niekoľko hrán súčasne, nedokážu ich uložiť na správne miesto v pamäti a správne ich označiť. Z toho dôvodu je implementované počítadlo hrán, trojuholníkov (identifikátor ID) a zámok pre ich čítanie. Jadro čaká do chvíle až zámok nie je otvorený. Keď sa tak stane, zámok sa zavrie a prečíta sa aktuálna hodnota identifikátora (iba jedno jadro v čase ho číta), inkrementuje sa, zapíše a následné odomkne zámok.

Takto sa vykonáva paralelná časť triangulácie, ktorá je voliteľnou možnosťou aplikácie.

4.6 Paralelný algoritmus optimalizácie

4.6.1 Blízke body

Problém dekompozície je prvý krok k paralelizácii algoritmu. Daný algoritmus je potrebné rozdeliť na menšie úlohy. Riešenie je nasledovné. Pri hľadaní blízkych bodov na CPU sa prechádza každý bod postupne a porovnáva sa navzájom s každým bodom

množiny P (pole sa celkovo prejde n^2 -krát, z čoho teoreticky vyplýva algoritmická zložitosť $O(n^2)$), pričom paralelný výpočet môže pole P rozdeliť tak, že každé jadro má priradený jeden bod (nazvime ho *bod aktuálny*), ku ktorému hľadá bod splňujúci podmienku tolerancie. Tento algoritmus spustí n (n je počet bodov v P) jadier na n procesných jednotkách. Čím by sa mala algoritmická zložitosť znížiť na $O(n)$, za predpokladu že všetky jadrá sa vykonajú naraz. Na začiatku algoritmu sa zistí či skúmané dva body už existujú a ak nie, proces na danom jadre sa preruší. Ak existujú a splňajú podmienku tolerancie, vypočíta sa nový bod, ktorý sa uloží na aktuálne miesto bodu priradeného procesu a druhý bod sa invaliduje (označí sa ako neexistujúci). Algoritmus prejde všetky prvky s ktorými porovná aktuálny bod a skončí. Množina bodov P je uložená v globálnej pamäti.

Spojenie blízkych bodov v spojení so silou paralelných výpočtov sa dá využiť pre značné urýchlenie výpočtu.

4.6.2 Metrika

Pri výpočte metriky na CPU sa prechádza každý bod postupne a porovnáva sa navzájom s každým bodom množiny P , okrem samého seba. Pole sa celkovo prejde n^2 krát, z čoho teoreticky vyplýva algoritmická zložitosť $O(n^2)$. Paralelný výpočet sa rozdelí na menšie úlohy výpočtu metriky pre jeden bod. Keďže jadrá sú rovnaké a zmena je len v ich identifikátore, identifikátor určí bod ktorý bude jadrom vyšetrovaný. Bod i sa vykonáva na i -tej procesnej jednotke. Tak sa dosiahne že n procesov spustených naraz, prejde pole iba n -krát. Tým sa algoritmická zložitosť teoreticky znížiť na $O(n)$, za predpokladu že všetky jadrá sa vykonajú naraz.

4.7 Výber kandidátov pre trianguláciu

V časti výberu štartovacieho bodu sa využíva funkcia *inCircle test* (pozri Nejednoznačnosť určenia polohy bodu na strane 20). Ak máme zvolený trojuholník ktorý chceme testovať, prechádzame pole bodov a hľadáme bod ktorý je vo vnútri gule, ak sa tam bod nenachádza, trojuholník testom prázdneho kruhu prejde. Ak sa tam bod p nachádza, vieme ho zvoliť ako ďalšieho možného nástupcu, miesto aktuálneho. Čiže v procese hľadania, sa vytvorí trojuholník z hrany a bodu $p(i)$, a následne sa bude testovať či sa v ňom nenachádza nejaký bod z poľa. Ak takto vytvorený trojuholník obsahuje bod $p(j)$ dosadíme ho v ďalšom cykle na miesto pôvodného. Takto sa vyberie nový vhodnejší kandidát bodu pre trianguláciu. Cyklus sa opakuje dokým sa nenájde taký bod

s ktorým hrana tvorí Delaunayov trojuholník. Ak body ležia na opísanej guli trojuholníka, cyklus sa po prvom prejení opakujúceho bodu preruší a posledný bod zostane ako jediný vhodný kandidát.

V hlavnom algoritme pri výbere bodu vhodného pre vytvorenie trojuholníka s hranou, sa testuje najprv *prvá podmienka maximálnej vzdialenosti* a potom sa bod porovnáva s ostatnými bodmi. Toto porovnanie môže byť trojaké. Porovnanie na základe:

- menšej vzdialenosti od hrany - berie sa do úvahy vzdialenosť hrany $e(n)$ a bodu $p(i)$. Pri porovnaní menšej vzdialenosti od hrany sa vypočíta stred hrany medzi jej dvoma krajnými bodmi a ten sa porovnáva s bodom. Táto metóda sa experimentálne ukázala ako najlepšia vzhľadom na kvalitu povrchu a času za ktorý ho dosiahne. Je to posledný čajník na nasledujúcom obrázku.
- menšieho polomeru opísanej gule - uvažuje sa guľa tvorená bodmi hrany $e(n)$ a bodom $p(i)$. Tato metóda sa ukázala ako najmenej výhodná.
- väčšieho uhlu - uhol je vypočítaný ako funkcia $\arccos$ skalárneho súčinu troch bodov. Prvým bodom je stredný bod predchádzajúcej hrany $e(n-1)$, druhým je aktuálna vyšetřovaná hrana $e(n)$ a tretím je bod $p(i)$. Vychádza sa z predpokladu že predchádzajúca hrana je v blízkosti hrany $e(n)$ a že ležia v rovine, tak vytvára toto spojenie bodov vektor kvázi rovnobežný s plochou. Tento výpočet je zjednodušený z dôvodu urýchlenia výpočtu celkovej triangulácie a v experimentoch sa osvedčil ako účinný.

Na obrázku Obr. 17 vidno zrejmé rozdiely vo výslednom povrchu. Celkovo najmenej mriežkou je pokrytý v poradí druhý čajník, čo je výsledkom porovňovania podľa menšej gule. Mriežka je veľmi deravá a detaily sú zničené. Rozdiel medzi poslednými dvomi je v detailoch ako sú uško kanvice a konkávna oblasť držiaku pokrievky čajníka. Pri prvom (porovnanie podľa väčšieho uhlu) je uško z väčšej časti pokryté, avšak na druhej strane je lievnik pri koreni nedokončený. Jeho výhoda je v tom že sú zachované drobné detaily medzi nádobou čajníka a pokrievkou, ako aj držiak pokrievky. Artefakty vo vnútornej časti ucha, na pokrievke a pri lievniku sú spôsobené toleranciou, ktorú spôsobuje *prvá a druhá podmienka maximálnej vzdialenosti*. Týmto vadám sa nedá predísť a sú akceptovateľné, pri väčších typoch povrchov sú zanedbateľné.



Obr. 18 Porovnanie výberu kandidátov (horný rad zľava: pôvodný model, podľa menšej gule, spodný rad: podľa väčšieho uhlu, menšej vzdialenosti od hrany)

4.8 Čítanie a zápis 3D dát

4.8.1 Formát Obj

Formát súboru .obj je formát pre ukladanie popisu povrchu 3D objektov, zložený z trojuholníkov (alebo z polygónov vyššieho stupňa). Obj súbor je textový súbor, čiže sa dá otvoriť v textovom editore a je čitateľný aj ľuďom. Súbor obsahuje okrem informácií o modeli aj informácie o materiáli modelu. Ukladajú sa v ňom dáta len v troch rozmeroch. Je to nekomprimovaný súbor. Môže obsahovať definíciu povrchu jedného alebo viac 3D objektov. Formát podporuje popis línií, polygónov, kriviek a plôch. Línie a polygóny sú popísané z hľadiska ich bodov, zatiaľ čo krivky a plochy sú definované cez kontrolné body. Obj súbory nevyžadujú žiadny druh hlavičky, hoci je vždy vhodné začať s komentárom obsahujúcim informácie o súbore. Riadky s komentárom začínajú znakom #. Každý neprázdny riadok začína kľúčovým slovom a pokračuje na rovnakom

riadku s údajmi pre dané kľúčové slovo. Riadky sú čítané a spracovávané postupne až do konca súboru [27] [28].

Najdôležitejšími kľúčovými slovami ktoré sú použité vo vstupných/výstupných súboroch v práci sú:

- v - dáta vrcholov,
- f - polygóny, trojuholníky,
- # - komentár.

Najčastejšie Obj súbory obsahujú len polygóny. K popisu mnohouholníka, súbor najprv opisuje každý bod s kľúčovým slovom "v", potom popisuje polygón označený kľúčovým slovom "f". Riadok pre polygón obsahuje číselné označenia bodov tvoriacich polygón. Číslovanie je poradie bodu definovaného vyššie v súbore a je číslované od 1 [27] [28].

```
1
2 # Simple Wavefront OBJ file
3 # Triangulation algorithm of Boris Nikolaevich Delaunay, code by Dominik Januvka
4
5
6 v 8.5498 5.6467 -6.6665
7 v 8.5567 5.8533 -6.7237
8 v 8.5521 5.6933 -6.4243
9
10 f 1 2 3
11
```

Obr. 19 Príklad výstupného súboru Obj

4.8.2 Formát WRL

Subor .wrl je objektom jazyka VRML (Virtual Reality Modeling Language, v preklade jazyk pre modelovanie virtuálnej reality). Je to jazyk pre popis interaktívnej simulácie mnohých účastníkov vo virtuálnom svete prepojeného prostredníctvom globálnej siete internet, a spojenej s celosvetovou počítačovou sieťou. Všetky aspekty zobrazenia virtuálneho sveta, ich interakcia a spojenie je špecifikované pomocou VRML. Zámerom jeho návrhárov bolo aby sa VRML stal štandardným jazykom pre interaktívnu simuláciu v rámci celosvetovej počítačovej siete. VRML súbory sa bežne nazývajú "svety". VRML svety sú popísané pomocou textového formátu. Mnoho 3D modelovacích programov dokáže importovať a exportovať formáty tohto typu [33].

Kľúčovými slovami použitými vo výstupných súboroch práce sú:

- # - tento znak označuje komentár,
- geometry - definuje blok údajov definície mriežky,
- coord - blok koordinátou,
- point - medzi hranatými zátvorkami definuje súradnice bodu v 3D, body sú navzájom oddelené čiarkou,
- coordIndex - blok trojuholníkov, sú to indexy vrcholov počítaných od 1, navzájom oddelené sú číslom -1.

```

1
2 # VRML V2.0 utf8
3 # Triangulation algorithm of Boris Nikolaevich Delaunay, code by Dominik Januvka
4
5 Shape {
6   }
7   geometry IndexedFaceSet {
8     coord Coordinate {
9       point [
10        43.224654019010366 15.98433296819569 52.81666567862353,
11      ]
12    }
13    coordIndex [
14      1, 2, 3, -1,
15    ]
16  }
17 }
```

Obr. 20 Príklad výstupného súboru WRL

4.8.3 Formát ASE

ASE (Ascii Scene Export) je skratka pre exportovane Ascii scény. Je to textový formát. ASE formát je preferovaný formát pre import statických mriežok modelov pre UnrealEd editor (editor Unreal jadra - počítačová hra) z takmer každého 3D editora. Formát je založený na množine identifikátorov (kľúčových slov) vo forme *MENO_IDENTIFIKATORU nasledované niekoľkými parametrami. Bloky viac identifikátorov sú uzavreté v zložených zátvorkách. Keďže neexistuje žiadna oficiálna špecifikácia, mal by každý identifikátor začať na samostatnom riadku, oddelený od parametrov len medzerami, tabulátormi a zátvorkami. Mriežka modelu je definovaná najprv zoznamom všetkých vrcholov, a potom zoznamom polygónov, v ktorých sú použité [32].

Najdôležitejšími kľúčovými slovami, ktoré sú použité vo výstupných súboroch práce sú (podľa [32]):

- *COMMENT "*komentár*" - komentár,
- *GEOMOBJECT - blok obsahujúci informácie o geometrii objektov v súbore,
- *NODE_NAME "*meno*" - meno objektu,
- *MESH - blok obsahujúci mriežku a informácie o textúrach,
- *MESH_NUMVERTEX *n* - počet všetkých vrcholov v mriežke,
- *MESH_VERTEX_LIST - blok údajov obsahujúci pozície bodov v 3D,
- *MESH_VERTEX - obsahuje 4 číselné parametre kde prvý je index vrcholu (začínajúci od 0), následne 3 koordináty x, y a z,
- *MESH_NUMFACES *n* - počet trojuholníkov,
- *MESH_FACE_LIST - blok obsahujúci definície trojuholníkov,
- *MESH_FACE - prvé číslo je index polygónu počítaný od 0, nasledujúce tri sú indexy v ňom použité vrcholov, ďalšia je orientácia hrán, skupina vyhladenia mriežky a napokon označenie materiálu pre mriežku [32].

```

1
2 *COMMENT "AsciiExport Version "
3 *COMMENT Triangulation algorithm of Boris Nikolaevich Delaunay, code by Dominik Ja
4
5 *GEOMOBJECT {
6     *NODE_NAME "mesh"
7     *MESH {
8
9         *MESH_NUMVERTEX 3
10        *MESH_VERTEX_LIST {
11            *MESH_VERTEX 43.224654019010366 15.98433296819569 52.81666567862353
12            *MESH_VERTEX 1 26.1801545923551 17.348905189861057 35.14706350997606
13            *MESH_VERTEX 2 16.63283193438877 49.739171446349516 44.99274644646699
14        }
15        *MESH_NUMFACES 1
16        *MESH_FACE_LIST {
17            *MESH_FACE 0: A: 1 B: 6 C: 4 AB: 1 BC: 1 CA: 0
18            *MESH_SMOOTHING 1 *MESH_MTLID 0
19        }
20    }
21 }
```

Obr. 21 Príklad výstupného súboru ASE

4.8.4 Vstup

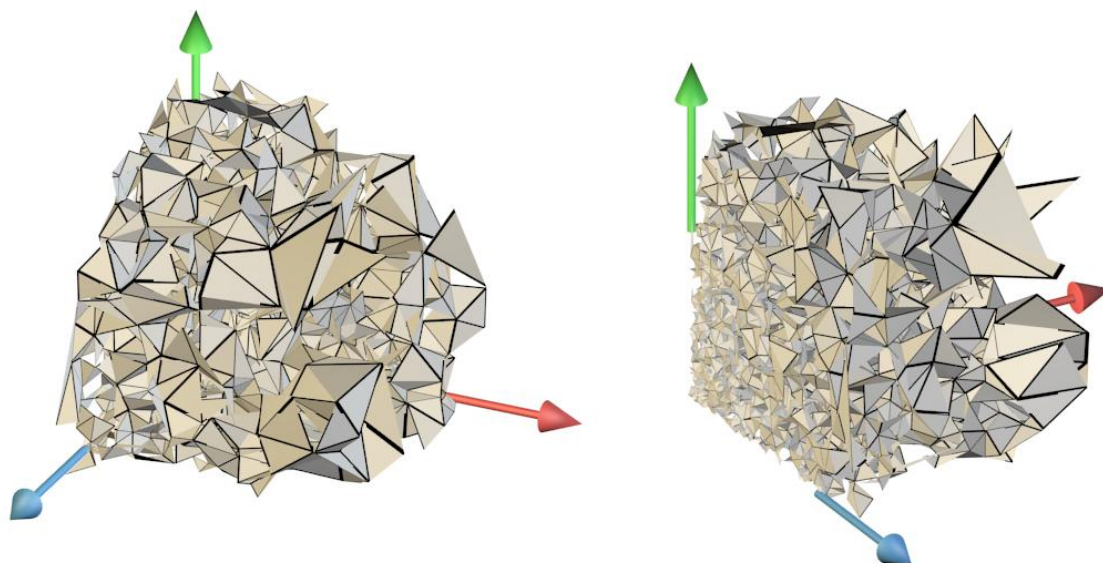
Vstupnými dátami môže byť akýkoľvek model alebo sken vo forme mračna bodov. Vstupný súbor musí obsahovať sadu bodov s tromi súradnicami x , y a z vo forme reálneho čísla. Program pracuje s dátovými typmi s pohyblivou rádovou čiarkou dvojitej presnosti. Ako bolo spomenuté, paralelný výpočet na GPU musí orezať túto presnosť na jednoduchú presnosť kvôli prenositeľnosti kódu. Dôležitou podmienkou je aby vstupná množina bola uložená vo formáte Obj.

4.8.5 Výstup

Ako výstupný súbor môže byť zvolený jeden z vyššie definovaných formátov Obj, WRL alebo ASE. Ak nebol použitý optimalizačný algoritmus spojenia blízkych bodov, tak do výsledného súboru je potrebné zapísať pôvodnú množinu bodov P , ak je zmenená, je potrebné zapísať novo vzniknutú množinu bodov. Ďalšou dôležitou položkou vo výstupe je zoznam polygónov, v tomto prípade trojuholníkov. Podľa užívateľom vopred zvoleného formátu sa uloží táto štruktúra do toho ktorého súboru. Všetky formáty sú v textovej podobe, takže je možné nahliadnuť do zdrojových kódov uloženého 3D súboru obsahujúceho striangulovanú mriežku.

4.9 Náhodné vstupné body

Keďže algoritmus pri základných nastaveniach sa snaží do triangulácie zahrnúť všetky body z množiny P , výsledný povrch z náhodne vygenerovaných bodov vyzerá nasledovne.



Obr. 22 Generované náhodné body v 3D

Náhodné body si môže užívateľ zvoliť na začiatku programu a sú využiteľné napríklad na testy funkčnosti programu na systéme, alebo na meranie výkonu hardvéru. Počet náhodne generovaných bodov v 3D priestore je variabilný, môže ho zadať užívateľ. Minimálny počet je obmedzený na hodnotu 9 bodov, keďže algoritmus začína s tromi trojuholníkmi a z nich každý obsahuje práve tri vrcholy. Existuje aj možnosť že trojuholníky majú spoločnú hranu, čiže aj dva vrcholy, ale potom by triangulácia s takto nízkym počtom bodov nemala význam. Maximálny počet bodov nie je obmedzený. Triangulácia takto náhodne vygenerovaného a neutriedeného povrchu je o čosi komplikovanejšia pre zariadenia na výpočet a preto trvá výpočet v porovnaní s modelmi a skenmi dlhšie.

4.10 Modely a skeny

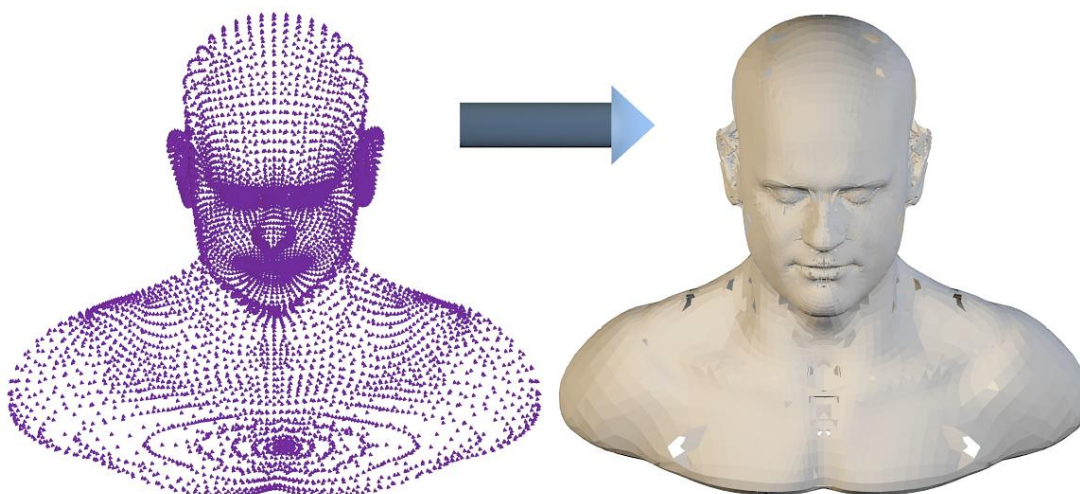
Množinu bodov získaných zo skenovacieho zariadenia nazývame skeny. Množinu bodov získaných z existujúceho 3D modelu vytvoreného v 3D modelovacom softvéri nazývame vo všeobecnosti model. V princípe ide z pohľadu uložených informácií o vrcholoch o ten istý pojem, rozdiel na prvý pohľad je v ten, že model uchováva navyše informácie o mriežke (program neberie tieto údaje do úvahy). Hlavný rozdiel je pozícií bodov v priestore a ich množstve. Zatiaľ čo model pozostáva z niekoľko tisíc, maximálne miliónov bodov, sken obsahuje bodov podstatne viac. Model je väčšinou reprezentácia nejakého povrchu telesa, ktoré chce jeho tvorca stvárniť v 3D. Model tvorí človek, preto nie je až taký komplikovaný a v istej miere nezmyselný ako sken.

Sken sú surové, neusporiadané dáta zozbierané skenovacím zariadením zo skutočného sveta. Práve z tohto dôvodu môže proces triangulácie trvať o čosi dlhšie ako spracovanie modelu.

O dátach zo skenovacích zariadení popísaných v kapitole 3D skenery na strane 14 sa uvádza, že výstupné súbory zo skenovacích zariadení obsahujú iba informácie o polohe bodov, ktoré sú postačujúcou podmienkou pre bežanie triangulačného algoritmu. Ak by sme sa rozhodli použiť ako vstupný súbor existujúci model, je to možné, ale program bude brať do úvahy iba informácie o polohe bodov. V závere triangulačného procesu si užívateľ môže porovnať pôvodný model a model vygenerovaný algoritmom triangulácie.

5 Generovanie povrchu

Výsledkom implementácie je aplikácia, ktorá generuje výsledný povrch zo vstupných údajov, na základe princípu Delaunayovej triangulácie. Program umožňuje užívateľovi istou mierou podieľať sa na tvorbe výslednej mriežky a to tak, že užívateľ môže zadať alebo modifikovať prednastavené konštanty použité v algoritme. Na začiatku algoritmu je na vstupe množina bodov P zobrazená na nasledujúcom obrázku vľavo. Aplikáciou vygenerované polygóny, v tomto prípade trojuholníky tvoria model, ktorý je reprezentáciou alebo skôr rekonštrukciou naskenovaného povrchu. Zrekonštruovaný model je zobrazený v pravej časti nasledujúceho obrázka.

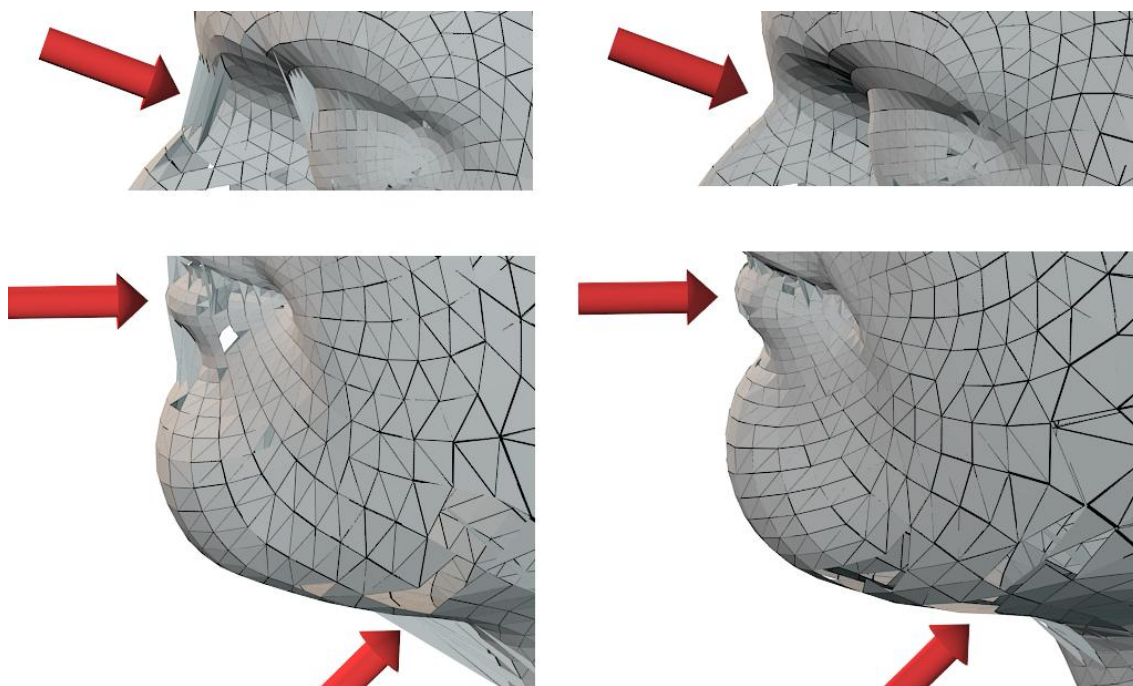


Obr. 23 Proces triangulácie, vstup, výstup

5.1 Chyby a úspešnosť

Pri procese triangulácie sa žiaľ môžu a vyskytnúť aj neželané artefakty. Je potrebné si uvedomiť, že algoritmus je určený pre všeobecné povrchy a nie špecifické ako napríklad iba guľové plochy alebo hranaté a deravé objekty. Program sa snaží vyhovieť čo najširšiemu spektru vstupných modelov. Algoritmus nemá vedomosť o tvare modelu, hustote bodov alebo ich pravidelnosti vzdialeností medzi nimi, no zároveň je postavený tak aby v čo najvyššej miere tvoril plochy na seba nadväzujúce a aby keď sa trianguluje po povrchu objektu nestalo, že skočí na nejakú odľahlú pozíciu v priestore a tak zničí kontinuitu výsledného povrchu.

Preto je tu prítomná istá miera chybovosti povrchu, ktorá je do značnej miery prípustná a ovplyvniteľná konštantami. Z nasledujúceho obrázku vidno (v ľavej časti) kritické oblasti, kde vytvorené polygóny síce prejdú oboma *podmienkami maximálnej vzdialenosti* ale sú neželané vo výsledku. Tie vznikajú hlavne pri veľmi zahnutých častiach modelov a pri vyššej hustote bodov.



Obr. 24 Kritické oblasti modelov

V pravej časti obrázku vidno už odstránené vadné polygóny v 3D editore. Jedným riešením je zmena užívateľom modifikovateľných konštánt s ktorými program pracuje pri tvorbe podmienok tvorby polygónov. Ak by sa zmenšili konštanty pod nejakú kritickú úroveň ktorá by nevytvárala tieto artefakty, celkom určite by sa stalo že v modeli by zostali obrovské diery a povrch by sa nedal považovať za dokončený. Naopak ak by sme chceli dosiahnuť najúplnejší povrch a konštanty by sme zväčšili na privysokú mieru, stalo by sa že algoritmus by bol pri tvorbe trojuholníkov benevolentnejší a tvoril by veľké množstvo chybných polygónov. Konštanty prednastavené v programe sú empiricky zistené ako najvhodnejšie pre väčšinu typov povrchov.

Modely môžu mať rôznu hustotu bodov v oblastiach detailnejšieho spracovania. Naopak pri rovinných častiach toho istého modelu, časti kde je nedostatok detailov je

nepotrebná vysoká hustota. Použitím metriky v algoritme dokážeme kontrolovať plynulosť povrchu.

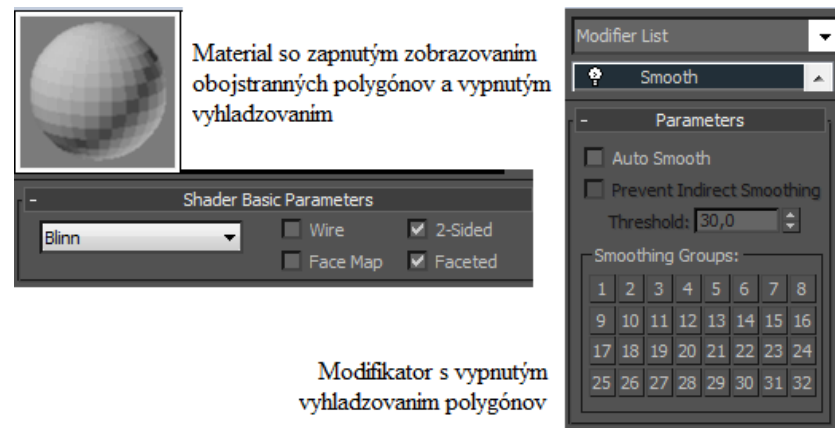
Diery v modeli vznikajú z triangulácie vadne vytvorených polygónov. Prítomná miera chybovosti povrchu dovoľí v niektorých prípadoch vytvoriť polygón v kritickej oblasti tak, že polygón nenadväzuje na povrch a vytvorí sa v zlom smere. Tento polygón má dve nové hrany, ku ktorým sa hľadajú body pre trianguláciu, a tieto hrany si môžu nájsť bod v nesúlade doposiaľ tvoreným povrchom. Táto následnosť udalostí spôsobí vytvorenie dier v modeli.

Výpočet úplnosti počíta koľko percent bodov zo vstupnej množiny bolo striangulovaných. Slúži na informačné účely určené pre užívateľa, ktorý sa podľa tejto informácie môže rozhodnúť či proces triangulácie spustí odznova, alebo nie. Tento program dosahuje úspešnosť tvorby povrchu v priemere 90%.

5.1.1 Post produkcia modelov

Hotový model je možné upraviť v ľubovoľnom 3D editore tak aby vyhovoval požiadavkám užívateľa. Ďalšími postupmi post produkcie môžeme len zvýšiť kvalitu výstupného súboru. Najčastejšie je to zmazanie nadbytočných trojuholníkov a neželaných artefaktov. Niekedy je vhodné spustiť trianguláciu odznova s inými nastaveniami, napríklad s náhodným štartovacím bodom. Tento postup zabezpečí zakaždým trochu iné výsledky od výsledkov predchádzajúcich a užívateľ tak môže byť uspokojený.

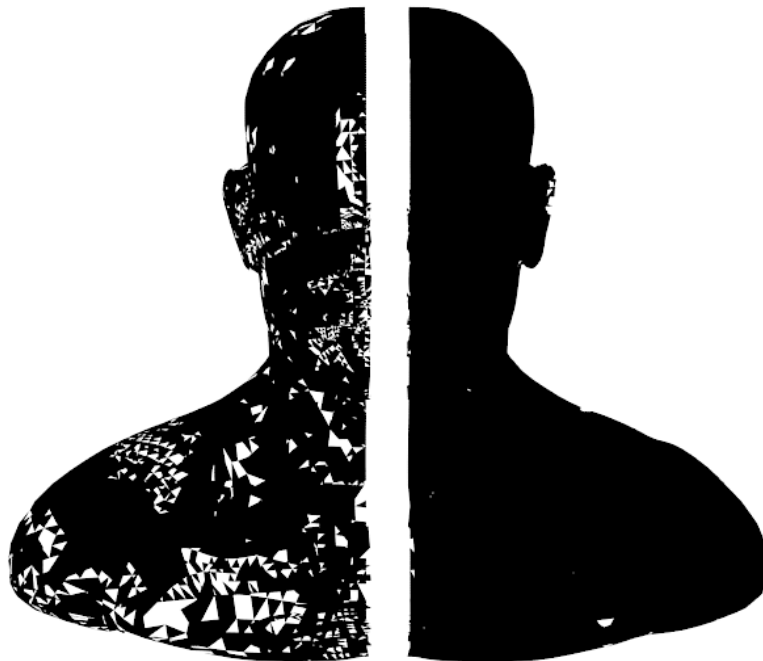
Výstupná generovaná mriežka nemusí byť vždy dopočítaná so sto percentnou úspešnosťou. Ak je model príliš veľký (má mnoho bodov) a výpočet trvá prídlho na to aby bola triangulácia spustená odznova, je možné model modifikovať v 3D editore. Táto núdzová oprava alebo povedzme post produkcia modelu je v celku jednoduchá. V prípade ak je model symetrický, stačí vytvoriť jeho zrkadlovú kópiu okolo osi modelu a modely prekryť. Takto sa s vysokou úspešnosťou zaplnia problémové miesta. Je vhodné spojiť prekrývajúce sa vrcholy. V neposlednom rade treba spomenúť že triangulačný proces nedefinuje normály a ani vyhladzovanie polygónov a preto sa pri zobrazovaní modelu môže stať, že 3D prehliadač si prečíta niektorý polygón ako otočený a vykreslí ho nevhodným spôsobom. Riešením je zmena materiálu objektu. Vo väčšine editorov sa dá zapnúť zobrazovanie obojstranných polygónov. Takisto sa dá vypnúť vyhladzovanie medzi polygónmi, buď zmenou preferencií materiálu alebo modifikátorom.



Modifikátor s vypnutým
vyhladzovaním polygónov

Obr. 25 Ukážka post produkcie v prostredí 3D studio Max. Zmena preferencií materiálu (vľavo), alebo modifikátorom (vpravo)

Na nasledujúcom obrázku je zobrazený vygenerovaný model v čiernej farbe, aby bolo vidno kontrast medzi modelom a pozadím. Takto je možné ľahko si všimnúť kvalitu výslednej mriežky. V ľavej časti je model bez post produkcie. V pravej časti je model modifikovaný pomocou všetkých spomenutých postupov. Už na prvý pohľad vidno značné zlepšenie dosiahnuté za krátky čas.



Obr. 26 Porovnanie "surového" modelu a post produkcie

5.2 Spracovanie dát z 3D skenera

Pomocou triangulačného procesu bol spracovaný výstupný súbor z 3D skenera. Súbor bol transformovaný do formátu Obj. Na nasledujúcom obrázku je porovnaný výstup z aplikácie pridávanej k 3D skenovaciemu zariadeniu a výstup z aplikácie ktorú opisuje táto diplomová práca. Model bol optimalizovaný spojením bodov s toleranciou 0,6. Model je prekrytý a spojený z dvoch triangulácií s náhodnými štartovacími bodmi. Nakoniec na model boli aplikované len niektoré zo spomenutých postupov post produkcie. Konkrétne jeho zrkadlová kópia.



Obr. 27 Porovnanie výstupov z triangulácie 3D skenera (hore) a triangulácie tejto práce (dole)

V porovnaní so štandardným algoritmom od 3D skenera, tento algoritmus dokáže zaplniť aj väčšie prázdne plochy v modeli (okná, dvere) a tak sa povrch modelu zdá byť úplnejší. Objavujú sa tu však isté artefakty a náhodné nevyplnené plochy, tak ako sa predpokladalo. Pri bližšom preskúmaní vidno, že nedostatkami trpí aj štandardný algoritmus.



Obr. 28 Bližší pohľad na porovnanie výstupov z triangulácie 3D skenera (vľavo) a triangulácie tejto práce (vpravo)

Mriežka modelu vyzerá po optimalizácii chaoticky, ale povrch modelu je plne popísaný s minimálnym počtom chýb. Program vyhodnotil úplnosť povrchu na 99% (výpočet úplnosti počíta koľko percent bodov zo vstupnej množiny bolo striangulovaných). Triangulácia mriežky spustená na CPU s použitou toleranciou spojenia bodov 0,6 trvala programu celkovo 96 minút a 59 sekúnd.

5.3 Výkonnosť

5.3.1 Merané zariadenia

Meranými zariadeniami bol procesor (ďalej označovaný ako CPU) a tri grafické karty od Nvidie, konkrétne karta GeForce GTX 275 (ďalej označovaná ako GPU1), karta GeForce 8500 GT (ďalej označovaná ako GPU2) a karta určená na výpočty Tesla C2075. Procesor Intel® Core™2 Quad CPU Q9400 má 64 bitovú architektúru, 4 výpočtové jadrá a frekvenciu jadier 2,66 GHz. Rozšírené špecifikácie zariadení sú pre porovnanie popísané v nasledujúcej tabuľke.

Tabuľka 1 Prehľad špecifikácie testovaných zariadení

	Intel Core2 Quad Q9400 2.66GHz	GeForce GTX 275	GeForce 8500 GT	Tesla C2075
Jadrá/CUDA jadrá	4	240	16	448
Frekvencia 1 jadra	2 666 Mhz	1 404 MHz	450 MHz	1 150
Frekvencia pamäte	1 066 MHz	1 134 MHz	400 MHz	1 500
Pamäť	8 GB DDR2	896 MB GDDR3	256 MB DDR2	6 GB DDR5
Priepustnosť pamäte (GB/sek)	9,207	127,0	12,8	144,0

Ako z tabuľky vidno, grafické karty majú o mnoho väčší počet výpočtových jadier v porovnaní s procesorom. Sice ich frekvencie sú o polovicu nižšie, ale svojou paralelnou architektúrou pri riešení paralelných problémov, prekonávajú výkon procesora mnohonásobne. Frekvencia pamäti je porovnateľná s frekvenciou pamäte procesora (pamäť procesora je myslená pamäť RAM). Veľkosť pamäte procesora je mnohonásobne vyššia, no je o čosi pomalšia, nie len pomalšími frekvenciami ale aj v závislosti umiestnenia čipu pamäte CPU na matičnej doske. Posledným, no nie menej

dôležitým, faktom je šírka priepustnosti pamäte. Pričom u CPU je to len niekoľko GB za sekundu, grafické karty majú priepustnosť neporovnateľne vyššiu, čo výrazne dokáže urýchliť čítanie a zápis dát uložených v pamäti grafickej karty.

5.3.2 Zrýchlenie a efektívnosť

Z nameraných údajov pomocou vzorcov 6 a 7 možno vypočítať zrýchlenie $S(n, p)$ a efektívnosť $E(n, p)$ pre všetky procesy aplikácie.

Tabuľka 2 Zrýchlenie a efektívnosť

triang. náhodná	
gpu1	
S(n, p)	E(n, p)
0,2159	0,09 %

triang. modelov			
gpu1		tesla	
S(n, p)	E(n, p)	S(n, p)	E(n, p)
0,37868	0,157 %	0,3076	0,068 %

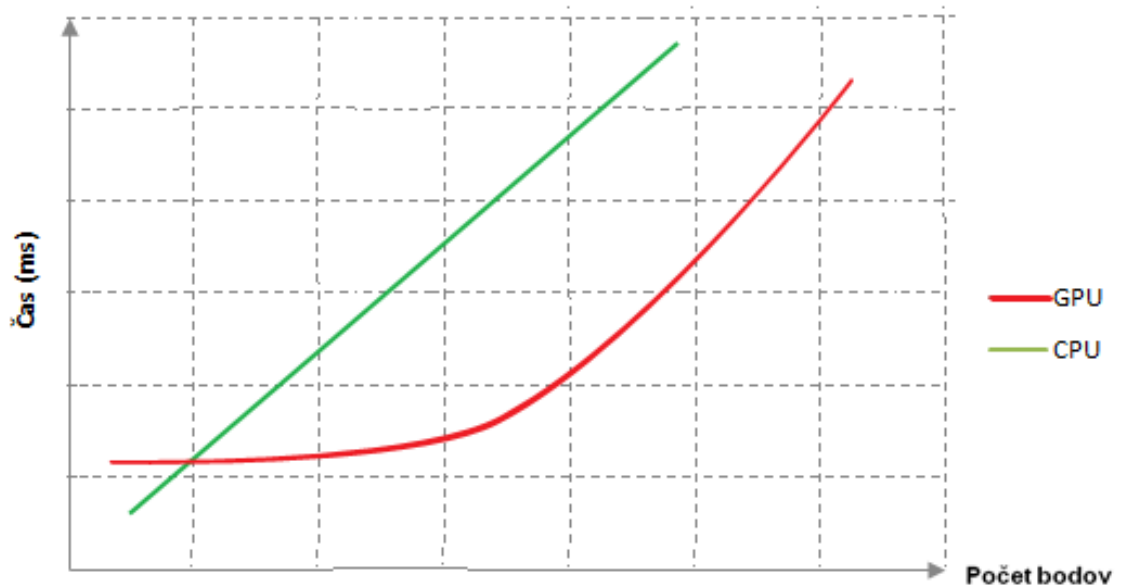
spoj. bodov	
gpu1	
S(n, p)	E(n, p)
1,0158	0,42 %

metrika					
gpu1		gpu2		tesla	
S(n, p)	E(n, p)	S(n, p)	E(n, p)	S(n, p)	E(n, p)
8,519	3,547 %	0,9512	0,39 %	10,861	2,424 %

Z tabuľky vyplýva že najvyššia efektívnosť paralelného výpočtu nastáva pri výpočte metriky. Naopak najnižšia je pri triangulácii náhodných bodov. Z týchto výsledkov môžeme vyvodiť teoretický záver, že optimalizačné procesy (metrika a spájanie bodov), ktoré i keď dosiahli len mierne zrýchlenie a efektívnosť, je vhodné spúšťať na GPU zariadeniach, zatiaľ čo proces triangulácie s nízkou efektívnosťou sa vyplatí spustiť na CPU. V poslednej časti tabuľky je porovnaný výkon medzi GPU1, GPU2 a Teslou. GPU1 a Tesla je výkonnejšia karta a preto bolo možné od začiatku predpokladať že bude mať väčšie zrýchlenie a efektívnosť. V nasledujúcej časti je porovnanie výkonov, dosiahnutých časov v jednotlivých procesoch na meraných zariadeniach.

5.3.3 Porovnanie

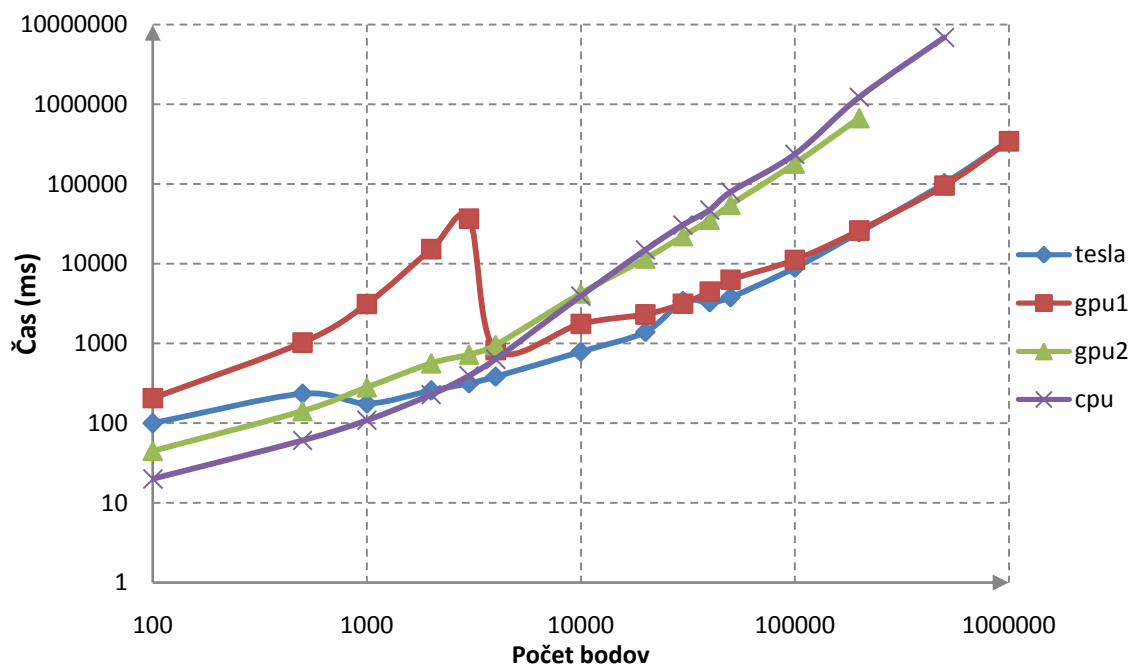
Teoretický, očakávaný výsledok GPGPU výpočtov v porovnaní s CPU, by mal mať lineárny priebeh pre CPU a GPU priebeh by mal byť na začiatku konštantný a neskôr exponenciálne rastúci, tak ako je znázornené na nasledujúcom obrázku.



Obr. 29 Teoretický výsledok GPGPU výpočtov v porovnaní s CPU

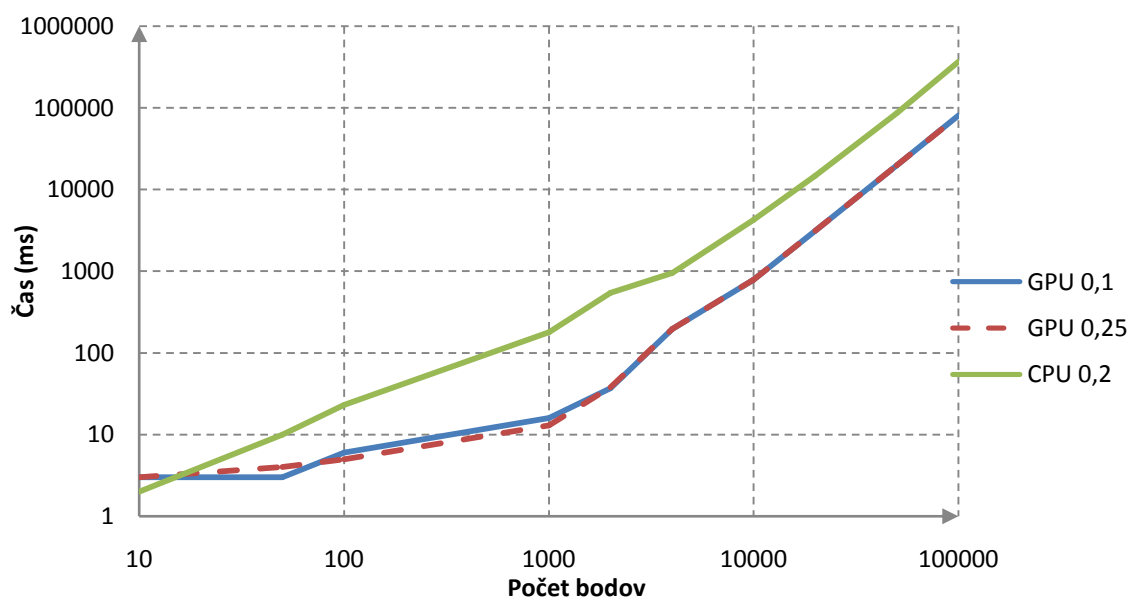
Je tomu tak z dôvodu, že paralelný výpočet pri narábaní s menším počtom prvkov je pomalší. Je pomalší z niekoľkých dôvodov. Obmedzený je rýchlosťou výpočtových procesorov GPU karty. Dáta je potrebné na začiatku presunúť z hlavnej pamäte do pamäte karty a vytvoriť prostredie pre spustenie paralelného výpočtu. Z tohto dôvodu je čas výkonu procesu pri nízkych hodnotách konštantný. Čím viac dát je potrebné preniesť, tým to môže trvať dlhšiu dobu. Práve preto je GPGPU výpočet s menším množstvom dát pre spracovanie pomalší v porovnaní s CPU. Aby sme využili plný potenciál GPGPU a dosiahli vysokú efektivitu paralelného algoritmu výhodné je pracovať s väčšími vstupnými dátami.

Na nasledujúcom grafe je zobrazená časová závislosť procesu metriky od počtu vstupných bodov. Z grafu vidno že výsledok CPU sa blíži ideálnemu tvaru ktorý sa predpokladal, a graf GPU zariadení sa s menšími odchýlkami tiež podobá očakávanému tvaru. Tu vidno rozdiel medzi GPU1, 2 a Teslou. Zatiaľ čo Tesla a GPU1 je silnejšia karta, jeho charakteristika sa oproti GPU2 blíži viac ideálnej línii paralelného výpočtu. Naopak GPU2 karta je slabšia a preto jej charakteristika je kdesi medzi týmito dvoma. Graf v podstate hovorí, čím viac bodov, tým viac potrebného času na výpočet. Preto krivka GPU1 je z prvej časti vyjadruje nízku efektivitu (z dôvodu už spomínaného nízkeho počtu bodov) ale v druhej časti sa jej priebeh ustáli smerom k ideálnej línii.



Obr. 30 Časová závislosť procesu metriky od počtu vstupných bodov

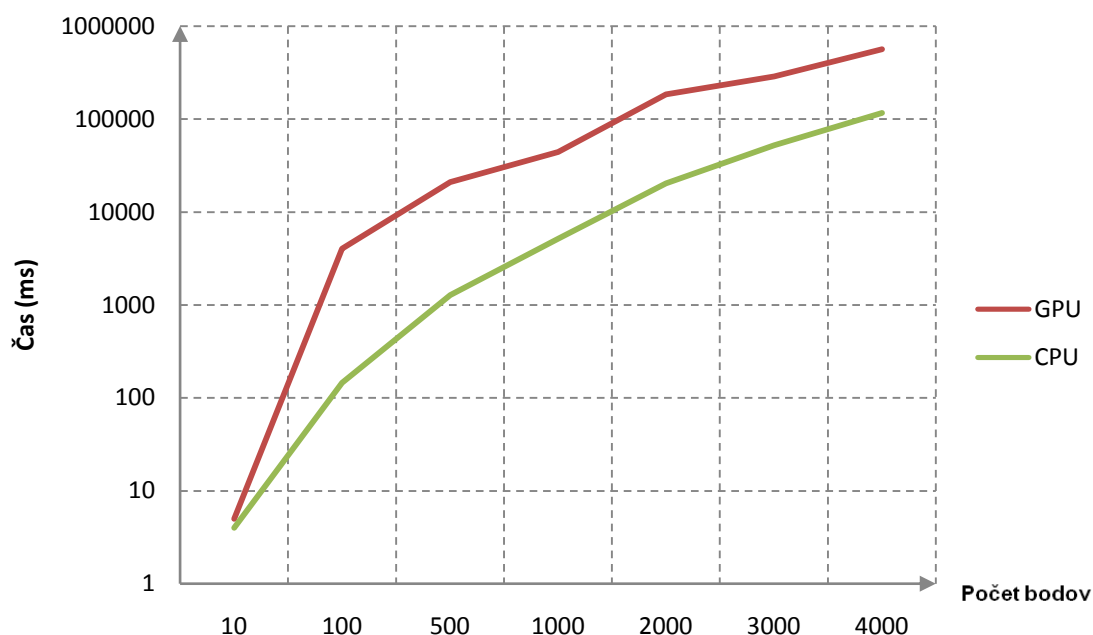
Druhý graf prehľadne zobrazuje vplyv optimalizácie spájania blízkych bodov. Z grafu vidno že hodnota tolerance nemá takmer žiaden vplyv na čas potrebný k výpočtu. Aj v tomto procese vidno že výsledná časová krivka meraných zariadení sa blíži k ideálnemu tvaru ktorý sme predpokladali.



Obr. 31 Časová závislosť procesu spájania bodov od počtu vstupných bodov

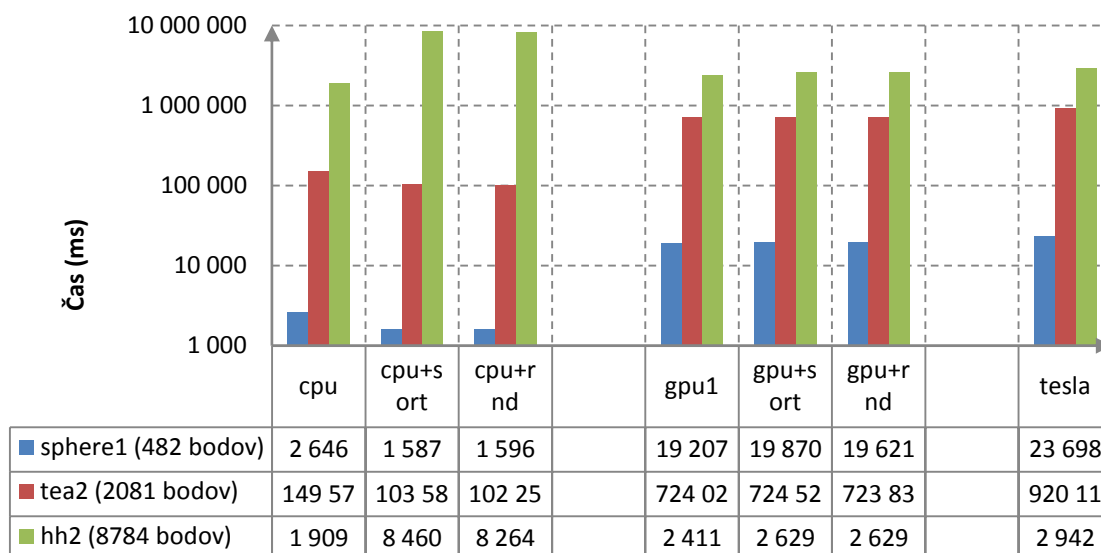
Z predchádzajúcich grafov je zrejmé, že GPU od istého počtu bodov modelu dosahuje nižšie časy výpočtu v porovnaní s CPU. Preto je výhodné využiť GPGPU na výpočet metriky bodov s vyšším počtom bodov.

Ďalší graf prehľadne zobrazuje nízku efektivitu, tak ako bolo teoreticky predpovedané. Pri triangulácii náhodných bodov sa ukázal teoretický predpoklad ako správny. Graf zobrazuje ako nízke zrýchlenie a efektivita vplývajú na výsledný čas výpočtu. Z toho vyplýva že paralelný proces triangulácie je pre náhodnú množinu akejkolvek (meranej) veľkosti bodov nevýhodný.



Obr. 32 Časová závislosť triangulačného procesu náhodne generovaných bodov

Posledný graf prehľadne zobrazuje vplyv optimalizácie a výberu náhodného štartovacieho bodu na čas vykonávania. Zatiaľ čo pri CPU triangulácii modelov s nízkym počtom bodov optimalizácia kladne ovplyvní dobu času triangulácie, GPU triangulácia nebude výrazne ovplyvnená. Z grafu veľmi jasne vidno že prvé dva modely triangulované na CPU potrebujú menej času ako na GPU. Tretí model však je tvorený väčším počtom bodov a GPGPU technológia výrazne zrýchli jeho výpočet.



Obr. 33 Graf závislosti triangulácie v závislosti od vstupných modelov

Z predchádzajúcich grafov vyplýva, že GPU od istého počtu bodov modelu dosahuje nižšie časy výpočtu v porovnaní s CPU. Preto je výhodné využiť GPGPU na výpočet triangulácie modelov s vyšším počtom bodov.

5.3.4 Štatistické súbory

Program po zápise 3D modelu do súboru, vytvorí štatistický súbor kde uloží všetky informácie o:

- zariadení,
- dĺžke trvania výpočtu triangulácie,
- dĺžke trvania výpočtu metriky,
- dĺžke trvania výpočtu spojenia bodov,
- špecifikuje model,
- informácie o modeli,
- nastavenia procesu: náhodný štart, triedenie, tolerancia spojenia bodov, konštanty.

Časové údaje sú ukladané v milisekundách.

```
1
2 device: GeForce GTX 275
3
4 triangulácia: (ms)
5 GPU čas : 2629808
6 CPU čas : -1
7 -----
8 metrika: (ms)
9 GPU čas : 921
10 CPU čas : -1
11 -----
12 spojenie bodov: (ms)
13 GPU čas : -1
14 CPU čas : -1
15 -----
16 model          : C:\Users\RADAS\Desktop\dist2\c
17 počet bodov    : 8784
18
19 náhodný štart: true
20 triedenie      : true
21 tolerancia     : -1.0
22 konštanta 1   : 12
23 konštanta 2   : 5
```

Obr. 34 Príklad štatistického súboru

Tieto súbory sa pomenúvajú formátom *statyyyMMdd_HHmms*, kde *stat* je označenie súboru, *yyyMMdd* je formát zápisu roku, mesiaca a dňa sekunda kedy bol vytvorený. *HHmms* je formát času hodina, minútá a sekunda kedy bol vytvorený. Z týchto súborov boli dáta zapísané do tabuliek a z nich boli vytvorené predchádzajúce grafy. Všetky štatistické súbory a tabuľky sú dostupné v prílohe na CD médiu.

6 Záver

Hlavným cieľom tejto diplomovej práce bolo navrhnuť a implementovať programové vybavenie pre proces triangulácie mračna bodov s využitím technológie GPGPU. Pre úspešné vypracovanie práce bol dodržaný postup podľa jednotlivých bodov a pokynov uvedených v zadaní. Všetky hlavné ciele boli úspešne splnené.

Ťažisko práce spočívalo v návrhu a implementovaní programového vybavenia pre proces triangulácie mračna bodov. Prínosom tejto práce bol vznik aplikácie, ktorá sa dá využiť ako nástroj pre spracovanie modelov vo forme neusporiadaného mračna bodov, teda ako výstupu z rôznych 3D skenovacích zariadení. Ten sa dá využiť v rôznych odvetviach každodenného života, či už v zdravotníctve, armáde, pri vývoji hier, projektovaní strojárenských riešení, kartografií, atď.

Aplikácia využíva multiplatformovú technológiu Java pre spúšťanie aplikácie na host'och a GPGPU technológiu pre zariadenia, konkrétne je to technológia OpenCL určená pre heterogénne systémy. Spojením Javy a OpenCL technológie vznikol silný a platformovo nezávislý nástroj, vykonávajúci proces triangulácie na heterogénnych systémoch. Výsledná aplikácia, generuje výsledný povrch zo vstupných údajov, na základe princípu Delaunayovej triangulácie. Program umožňuje užívateľovi istou mierou podieľať sa na tvorbe výslednej mriežky. Na začiatku algoritmu je na vstupe množina bodov a aplikáciou vygenerované polygóny vytvoria model, ktorý je reprezentáciou alebo rekonštrukciou naskenovaného povrchu.

Pri procese triangulácie sa žiaľ môžu a vyskytnúť aj neželané artefakty spôsobené nepredvídateľnou rôznorodosťou skenovaného povrchu.

Merania časov behu aplikácie na rôznych zariadeniach do istej miery splnili očakávaný teoretický predpoklad priebehu krivky závislosti, medzi počtom naskenovaných bodov a časovým úsekom potrebným na výpočet modelu. Bol zistený výkonnostný rozdiel medzi jednotlivými GPU kartami. Zatiaľ čo sa výkonová charakteristika silnejších kariet blížila ideálnej línii, karty slabšie dosahovali celkovo nižší výkon. Avšak aj slabšia karta pri efektívnom rozložení paralelného problému a väčšom počte bodov dosiahla nižšie časy výpočtu ako procesor. Testy potvrdili aj teoretické predpoklady nízkej efektivity paralelných výpočtov pri nízkom počte bodov.

Z meraní časov triangulácie modelov bolo zrejmé, že GPU od istého počtu bodov modelu dosahuje nižšie časy výpočtu v porovnaní s CPU. Preto je výhodné využiť

GPGPU na výpočet nielen metriky, ale aj skenovaných modelov s vyšším počtom bodov.

Najnižšie teoreticky vypočítané zrýchlenie a efektivita sa prejavila pri paralelnej triangulácii náhodne vygenerovaného mračna bodov. Z toho vyplýva že paralelný proces triangulácie je pre náhodnú množinu akejkol'vek (meranej) veľkosti bodov nevýhodný. Naopak paralelný výpočet metriky a triangulácia väčších modelov sa ukázali nie len teoreticky, ale aj prakticky ako efektívnejšie v porovnaní so sériovým výpočtom.

Triangulácia povrchov je využívaná v širokej oblasti odvetvi pre spracovanie naskenovaných povrchov. Naštudovanie a chápanie problematiky bolo zaujímavým a prínosným procesom. Z výsledkov práce vyplýva výhoda GPGPU technológie a jej vhodné využitie pri výpočte metriky a triangulácie modelov s vyšším počtom bodov. Vzniknutá aplikácia okrem urýchlenia výpočtu vďaka využitiu GPGPU technológii poskytuje aj svoju multiplatformovú výhodu.

Zoznam použitej literatúry

- [1] OKABE, A.- BOOTS, B.- SUGIHARA, K.: Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. New York: Wiley, 1992, s. 94, ISBN-10 0471986356
- [2] <http://en.wikipedia.org/wiki/Polygon> , Wikipédia
- [3] Gabriel on Graphics: An easy, comprehensive, and step by step approach to computer graphics, September 25, 2005. Dostupné na internete: <<http://gabrielongraphics.blogspot.com/2005/09/drawing-triangles.html>>
- [4] NAVARRO, Cristobal A.- HITSCHFELD-KAHLER, Nancy- SCHEIHING, Eliana: A Quasi-Parallel GPU-Based Algorithm for Delaunay Edge-Flips, February 28, 2011. Dostupné na internete: <http://swp.dcc.uchile.cl/TR/2011/TR_DCC-20110228-002.pdf>
- [5] <http://sk.wikipedia.org/wiki/Trojuholník> , Wikipédia - trojuholník
- [6] SHEWCHUK, Jonathan Richard: Lecture Notes on Geometric Robustness, October 4, 1999.
- [7] http://en.wikipedia.org/wiki/Delaunay_triangulation , Delaunay triangulation
- [8] VAVREK, Miloš: Triangulácia: Implementácia algoritmu pre výpočet Delaunayovej triangulácie, Diplomová práca, Košice, s. 8-9, 2007.
- [9] De BERG M.- VAN KREVELD M.- OVERMARS M.- SCHWARZKOPF O.: Computational Geometry: Algorithms and Applications 3rd ed, Springer-Verlag New York, 2008
- [10] CIGNONI P.- MONTANI C.- SCOPIGNO R.: DeWall: A fast divide and conquer Delaunay triangulation algorithm in Ed, Computer-Aided Design, s. 333– 341 , 1996
- [11] DWYER, R.A.: A faster divide-and-conquer algorithm for constructing Delaunay triangulations, Algorithmica, 1987
- [12] SU, P. - DRYSDALE, R.L. Scott: A Comparison of Sequential Delaunay Triangulation Algorithms, Computational Geometry: Theory and Applications, 1997
- [13] WATSON, D.F.: Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes, Computer Journal, 1981
- [14] <http://www.s-hull.org/> , S-hull: a fast sweep-hull routine for Delaunay triangulation
- [15] FINDT, Andras P.- CLARKE Willem A.: Delaunay triangulation using a parallel architecture, School of Electrical and Electronic Engineering, University of Johannesburg, Auckland Park, South Africa, 2008.
- [16] <http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/model/euler.html> , The Euler-Poincaré Formula

-
- [17] SHEWCHUK, Jonathan Richard: Delaunay Refinement Mesh Generation, Carnegie Mellon University, Pittsburgh, 1997
 - [18] DAVIS, Martin: Improvements to robustness in JTS Delaunay Triangulation, 2010. Dostupné na internete: <<http://lin-ear-thinking.blogspot.com/2010/06/improvements-to-robustness-in-jts.html>>
 - [19] SHEWCHUK, Jonathan Richard: Adaptive Precision Floating-Point Arithmetic and Fast Robust Predicates for Computational Geometry, California. Dostupné na internete: <<http://www.cs.cmu.edu/~quake/robust.html>>
 - [20] <http://cs.wikipedia.org/wiki/CUDA> , CUDA
 - [21] http://www.nvidia.com/object/cuda_home_new.html , Nvidia -CUDA architektúra
 - [22] NVIDIA CUDA Programming Guide Version 1.1 , Dostupné na internete: <<http://developer.nvidia.com/nvidia-gpu-computing-documentation>>
 - [23] An Introduction to OpenCL. Dostupné na internete: <<http://www.amd.com/us/products/technologies/stream-technology/opencl/pages/opencl-intro.aspx>>
 - [24] OpenCL Tutorials 1 - Quickstart , 2010. Dostupné na internete: <<http://opencl.codeplex.com/wikipage?title=OpenCL%20Tutorials%20-%201>>
 - [25] MATTSON Tim - BUCK Intellian - HOUSTON Michael - GASTER Ben: OpenCL - A Standard Platform for programming Heterogeneous parallel computers, 2009. Dostupné na internete: <<http://www.scribd.com/doc/78000156/43/Host-Set-kernel-arguments-2>>
 - [26] Editor: MUNSHI Aaftab: The OpenCL Specification, Version: 1.1, Khronos OpenCL Working Group, 2011. Dostupné na internete: <<http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf>>
 - [27] <http://www.fileformat.info/format/wavefrontobj/egff.htm> , Wavefront OBJ File Format Summary.
 - [28] <http://people.sc.fsu.edu/~jburkardt/data/obj/obj.html> , OBJ Files, A 3D Object Format.
 - [29] <http://gpgpu.org/> , General-Purpose Computation on Graphics Hardware.
 - [30] GASTER, Benedict - HOWES, Lee - KAELI, David R. - MISTRY, Perhaad - SCHAA, Dana : Heterogeneous Computing with OpenCL, USA, 2012. ISBN 978-0-12-387766-6
 - [31] MUNSHI - Aaftab, GASTER - Benedict R., MATTSON - Timothy G., FUNG - James , GINSBURG - Dan : OpenCL Programming Guide, Michigan, USA, 2012. ISBN-13: 978-0-321-74964-2
 - [32] http://wiki.beyondunreal.com/Legacy:ASE_File_Format , Legacy: ASE File Format
 - [33] <http://www.web3d.org/x3d/specifications/vrml/VRML1.0/> , The Virtual Reality Modeling Language, Version 1.0 Specification. 9. Nov. 1995.
-

-
- [34] KIŠ, Roman: 3D zobrazovanie mapových podkladov. Košice, 2010.
 - [35] <http://www.jocl.org/index.html> , jocl.org - Java bindings for OpenCL
 - [36] DOĽÁK Róbert: 3D skenovanie. Košice, 2010, str. 20-22
 - [37] <http://www.ten24.info/index.php/shop/3d-head-scans/free-3d-head-scan-model-52/>
 - [38] KOLLÁR, Ján: Metódy a prostriedky pre výkonné paralelné výpočty. TUKE, FEI, Košice, 2003, str. 27-28.

Prílohy

Príloha A: CD médium

Príloha B: Používateľská príručka

Príloha C: Systémová príručka