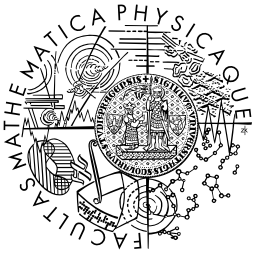


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Peter Polák

Decimace trojúhelníkových sítí

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce:	RNDr. Josef Pelikán
Studijní program:	Informatika
Studijní obor:	Programování a softwarové systémy

Praha 2018

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Chcel by som sa poďakovať RNDr. Josefovi Pelikánovi za jeho ochotu, neoceniteľné rady pri vedení práce a za poskytnutie štartovacieho programu.
Ďalej by som sa chcel poďakovať mojim rodičom za ich podporu pri štúdiu.

Název práce: Decimace trojúhelníkových sítí

Autor: Peter Polák

Katedra softwaru a výuky informatiky: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán, Katedra softwaru a výuky informatiky

Abstrakt: Dnešné 3D skenery produkujú veľmi detailné a rozsiahle scény pozostávajúce z obrovského počtu trojuholníkov. Takéto veľké siete nie sú vhodné na ďalšie spracovanie v analytických metódach. Práca sa venuje decimácii takýchto trojuholníkových sietí s atribútmi. Redukcia počtu trojuholníkov je skúmaná ako optimalizačný problém. Predstavené sú niekoľké algoritmy a postupy pri riešení vonkajšej a vnútornej optimalizácie. Navrhnuté sú tri kompletné decimácie. Dôraz je kladený najmä na zachovanie geometrie siete, ale aj iných atribútov, ako napríklad textúr a normál. Výsledky navrhnutých decimácii boli porovnané s existujúcimi riešeniami.
Výsledkom práce je ucelený program s grafickým užívateľským rozhraním. Ten dokáže načítať a zobraziť 3D scénu, zdecimovať ju pomocou užívateľom zvoleného algoritmu a parametrov. Na koniec je možné scénu exportovať.

Klíčová slova: level of detail decimácia sietí kvadratická chybová funkcia nespojitosť atribútov

Title: Triangle mesh decimation

Author: Peter Polák

Department of Software and Computer Science Education: Department of Software and Computer Science Education

Supervisor: RNDr. Josef Pelikán, Department of Software and Computer Science Education

Abstract: Modern 3D scenners produce detailed and vast scenes consisting of huge number of triangles. This thesis is dedicated to simplification of such triangular meshes with attributes. The problem of reducing the number of triangles is studied as an optimization problem. Couple algorithms and approaches are introduced as solution of inner and outer optimization process. Finally, three simplification algorithms are introduced. The goal is not only to preserve the geometry of the mesh, but also its attributes such as textures and normals. Results are compared with existing solutions.
The result of thesis is a complete software with graphical user interface. This software is able to load and display mesh, decimate it with user defined algorithm and parameters. Eventually, it's possible to export the scene.

Keywords: level of detail mesh decimation quadratic error function attribute discontinuities

Obsah

Úvod

V posledných desaťročiach sme svedkami obrovského pokroku v oblasti vedy, techniky a informatiky. Nové technológie nám dnes umožňujú robiť veci, o ktorých sa nám včera ani nesnívalo.

Tento pokrok sa nevyhol ani oblasti 3D scannerov — zariadení, ktoré zaznamenávajú tvar, prípadne dokážu zaznamenať aj iné atribúty ako napríklad farbu. Prvé technológie schopné zaznamenávať 3D scénu sa objavili v 60. rokoch 20. storočia. Na svoju činnosť využívali svetlá, fotoaparáty a projektory [1]. Dnes existujú rozličné scannery pracujúce na rôznych princípoch (napríklad laser alebo fotogrammetria), ktoré sú schopné zachytiť ten najjemnejší detail.

Napriek tomu, že obrovské pokroky neobišli ani oblasť hardvéru, zložitosť a veľkosť dnešných 3D scén prekračuje schopnosti tohto hardvéru zobrazovať a spracovávať takéto scény. Príkladom takéhoto modelu môže slúžiť Michelangelov Dávid z projektu „The Digital Michelangelo Project“ [2], naskenovaný s presnosťou 0,25 mm a ktorý je tvorený z 2 miliárd mnohouholníkov.

Obr. 1: Michelangelov Dávid z projektu „The Digital Michelangelo Project“ [2]. Tvorený 2 miliardami mnohouholníkov.

Aby bolo možné pracovať s komplexnými scénami, boli vyvinuté postupy, ktoré dokážu regulovať úroveň detailov. Táto disciplína sa zvykne označovať anglickým *level of detail*, alebo skrátene *LOD*. Pomocou techník *LOD* dokážeme redukovať modely virtuálneho sveta tak, aby ich bolo možné či už zobrazovať, alebo využiť skúmaní vlastností daných objektov.

Jedným z pracovísk využívajúce 3D skenovacie zariadenia je aj Laboratoř 3D zobrazovacích a analytických metod na Prírodovedeckej fakulte UK. Laboratórium pri svojom výskume využíva skenery napríklad na skenovanie tvárí ľudí. Nespracované dáta, ktoré poskytuje skener tvári Vectra 3D, majú rádovo 400 000 trojuholníkov. Pre využitie týchto skenov napríklad pre analytické metódy v geometrickej morfometrii nie je možné použitie takto detailných scén. A práve tu prichádza na rad *LOD*.

Skener tvárí Vectra 3D

Skener tvárí Vectra 3D využíva pri rekonštrukcii 3D scény stereofotogrametriu. Jedná sa o techniku, ktorá sa na základe štyroch snímkov snaží určiť trojdimenzionálne súradnice povrchu snímaného telesa. Konkrétny skener vyhotovuje šesť snímok. Štyri snímky, zachytávajúce tvár z bokov, sú čiernobiele a práve tie sa využívajú na rekonštrukciu 3D scény. Ďalšie dva, farebné snímky, sa využívajú na určenie farby bodov, resp. textúry. Výstupom je trojuholníková sieť (mesh) s textúrou.

Obr. 2: 3D sken tváre zosnímaný pomocou Vectra 3D. 476 592 trojuholníkov.

Surový, neupravený sken tváre obsahuje rôzne chyby. Príkladom môžu byť chyby geometrické — zle štrukturované okraje (viď obrázok č. 2), chyby v oblasti nosných dierok, očí, obočia a čela. Ďalším problémom je nespojitosť atribútov. Textúra sa na 3D sken rekonštruuje z dvoch farebných snímok zhotovených z bokov tváre (obrázok 3.4). Pre každý vrchol tvoriaci 3D scénu sa vyberie miesto (súradnica na obrázku) z jednej snímky. Takto vzniknú na tvári miesta, kde sú susedné vrcholy namapované na odlišné snímky. Nespojitosť atribútov nemusí byť nevyhnutne viditeľná, avšak pri ďalšom spracovávaní môže spôsobiť ťažkosti.

Ciele práce

Laboratórium 3D zobrazovacích a analytických metód spolu so Skupinou počítačovej grafiky UK na Matematicko-fyzikálnej fakulte vyvíjajú nástroj Morphome3cs. Projekt sa zaoberá analýzou biologických objektov. Skeny tvárí sú práve jedným z príkladov spracovávaných dát. Modely tvárí využívajú napríklad na štúdie vývoja lebky, či rekonštrukcie mäkkých tkanív. Pri svojej činnosti spracová-

vajú množstvo skenov. Ako už bolo spomenuté, surový výstup zo skenera nie je vhodný pre použitie (napríklad v analytických metódach a to najmä z dôvodu veľkého počtu vrcholov).

Cieľom tejto bakalárskej práce je navrhnúť algoritmy na decimáciu trojuholníkových sietí tak, aby dokázali zachovať atribúty vrcholov (najmä textúr). Trojuholníkové siete budú predovšetkým 3D snímky tvárí ľudí. Jednou z dôležitých požiadavok na tieto algoritmy teda je, aby dokázali spracovávať nevyčistené surové dáta zo skeneru. Špecifickým cieľom je optimalizovať decimáciu na meshe snímok tvárí. Na ďalšie spracovanie sa meshe typicky decimujú na približne 26 000 trojuholníkov.

Výsledkom práce bude samostatný GUI program pre Windows, schopný načítať sieť a po zadaní parametrov decimácie vykoná decimáciu a výsledok zobrazí. Načítanie 3D scény bude možné zo súborov formátu OBJ a export bude možný do formátov OBJ a PLY.

Štruktúra práce

Práca je delená na šesť kapitol. Prvá kapitola sa venuje analýze algoritmov a techník pri decimácii sietí. Priblížia sa základné prístupy — decimácia na základe vernosti a decimácia na základe budgetu (napríklad maximálneho počtu trojuholníkov). Predstavia sa niektoré decimačné operátory a nakoniec sa rozoberú prístupy k radeniu operácii decimácie.

Druhá kapitola diskutuje chybové metriky. Budú predstavené niektoré jednoduché a nakoniec aj komplexnejšie prístupy k meraniu chyby.

Tretia kapitola spája a dopĺňa poznatky z predchádzajúcich kapitol. Predstavuje implementované decimácie a okrem toho sa venuje aj oprave chýb textúry, ktoré vznikajú počas procesu decimácie.

Štvrtá kapitola vyhodnocuje výsledky. Decimované siete sa porovnávajú s originálnymi z pohľadu geometrickej chyby ale aj vizuálne. Okrem toho, demonštruje dva existujúce nástroje schopné decimovať siete.

Piata kapitola obsahuje užívateľskú príručku. Dokumentuje užívateľské prostredie programu a okrem toho aj diskutuje vhodnú voľbu parametrov jednotlivých implementovaných decimácii tak, aby sa dosiahli optimálne výsledky.

Posledná, šiesta kapitola je programátorská príručka. Predstaví základný projekt, ktorý bol v implementačnej časti bakalárskej práce rozšírený. Ďalej sa venuje implementácii algoritmov z tretej kapitoly.

Dôležité pojmy

Mesh

Mesh, *sieť* alebo *scéna* je označenie pre množinu vrcholov, hrán a stien, ktoré definujú teleso v 3D. Informácie o meshi sú zakódované v topológii meshe a v jej geometrii.

Dôležitým pojmom je aj *2-manifold*. Hovoríme, že mesh je 2-manifold, ak:

(i) každá hrana má jeden alebo dva susedné trojuholníky,

(ii) hrany majúce spoločný vrchol tvoria vejár (môže byť uzavrený).

Vrchol

Geometria meshe je uložená vo vrcholoch/vertexoch — je to pozícia v priestore. Vrcholy môžu mať aj atribúty. To môžu byť napríklad farby, normály (väčšinou normované na jednotkové vektory) alebo textúrové súradnice. Niektoré vertexy môžu mať rôzne atribúty, teda pre rôzne susedné steny môže mať rôzne atribúty.

Stena

Steny sú množiny vrcholov. Je v nich zakódovaná topológia meshe. V tejto práci sa pracuje len s trojuholníkovými sieťami, preto pojmom stena sa bude označovať ďalej trojuholník.

1. Analýza techník decimácie meshí

Táto kapitola sa venuje prehľadu techník decimácie. Najprv sa porovnajú dva základné prístupy — decimácia s uvedenou maximálnou chybou a decimácia s určeným „budgetom“ (typicky počet trojuholíkov). Ďalej budú predstavené niekoľké simplifikačné operátory. Na záver kapitoly budú pojednávané decimačné frameworky, teda spôsoby ako radiť jednotlivé kroky decimácie.

1.1 Základné prístupy k decimácii meshí

Na základe účelu decimácie sa dá pristupovať k decimácii meshe dvoma spôsobmi. Pokiaľ je prvoradá výsledná kvalita zdecimovanej meshe, dá sa zvoliť fidelity-based decimácia. V opačnom prípade, keď je priorita napríklad maximálny počet trojuholníkov, jedná sa o budget-based decimáciu [3, str. 20].

1.1.1 Fidelity-based decimácia

V situáciach, keď je zachovanie vzhľadu meshe dôležitejšie ako výsldný počet trojuholníkov, je výhodná fidelity-based decimácia [3, str. 20]. Decimačnému algoritmu sa uvedie maximálna dovolená chyba (error) vzhľadom na pôvodnú mesh (chybovým metrikám sa bude venovať kapitola 2). Decimačný algoritmus sa potom snaží minimalizovať počet trojuholníkov, tak aby nebola uvedená maximálna chyba presiahnutá. Optimálna minimalizácia počtu trojuholníkov sa považuje za NP-úplný problém. Bežne sa teda používajú suboptimálne algoritmy.

1.1.2 Budget-based decimácia

Opačne k decimácii pristupuje budget-based decimácia [3, str. 20]. Algoritmus má zadaný maximálny počet trojuholníkov a úlohou algoritmu je minimalizácia chyby novej meshe vzhľadom k tej pôvodnej. Pretože je veľkosť siete kritériom, nie je garantovaná kvalita výstupu.

Výhodou tohto prístupu je možnosť riadiť veľkosť zdecimovanej siete. Táto vlastnosť sa preto často využíva v časovo náročných aplikáciach.

1.2 Decimačné operátory

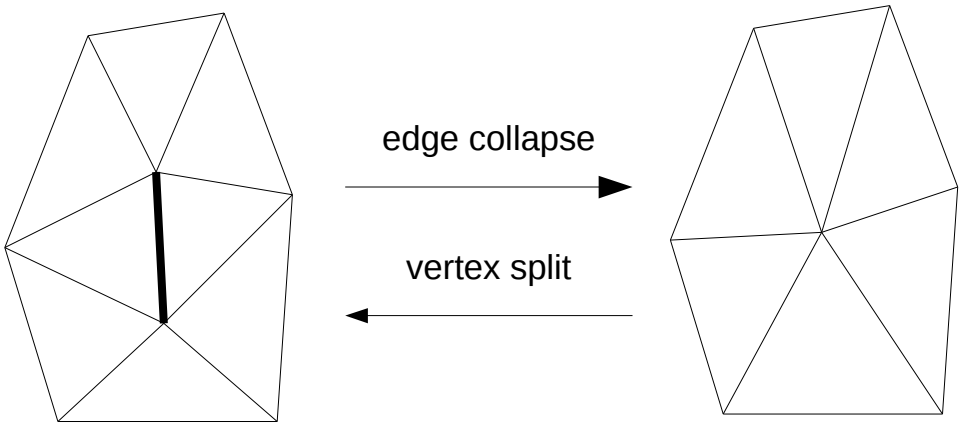
Nasledujúca podkapitola sa venuje niekoľkým vybraným operátorom využívaných pri decimáciach. Decimácia je optimalizačný proces. Tento pozostáva z vnútornej a vonkajšej optimalizácie. Pri vnútornej optimalizácii sa využívajú lokálne decimačné operátory. V tejto práci sa využívajú len lokálne simplifikačné operátory. Dôvodom tejto voľby je fakt, že pri decimácii skenov tvári nie sú potrebné zmeny v topológii meshe [3, str. 28] (globálne operátory sú často využívané práve na zmeny v topológii).

Lokálne simplifikačné operátory predstavujú triedu nízkoúrovňových operácii,

ktoré menia decimovanú sieť po malých kúskoch [3, str. 21]. V podkapitole 1.3 sú preberané algoritmy, ktoré využívajú tieto lokálne operátory.

1.2.1 Edge collapse operátor

Edge collapse operátor zlúči vrcholy v_a a v_b, ktoré sú spojené hranou, do nového vrcholu v_n. Okrem odstránenia hrany, ktorá pôvodné dva vrcholy spájala sa odstránia aj trojuholníky, ktoré obsahovali skolabovanú hranu. Opačným operátorom je vertex split [3].



Obr. 1.1: Edge collapse a vertex split.

Edge collapse operátor má dve varianty — half-edge a full-edge. Half-edge collapse operátor kolabuje hranu do jedného z pôvodných vrcholov. Motiváciou pre half-edge collapse operátor je napríklad zamedzenie zápisov do vertex-bufferu, čo môže byť výhodné pri zmenách *LOD* počas behu [4].

Mesh Foldover

Jedným z nepríjemných úkazov pri edge-collapse operátore môže byť mesh foldover, teda keď sa vytvorí záhyb alebo preklad. Tento jav sa dá detegovať meraním zmien v normálach dotknutých trojuholníkov. Mesh foldover je charakteristický veľkými zmenami v uhloch normál, typicky väčšími než 90° [3, str. 22].

Zmeny v topológii

Okrem vyššie spomenutého mesh foldover-u môže dôjsť k topologickým zmenám meshe. Z manifold meshe sa môže stať non-manifold mesh, teda keď hrana má jeden, tri alebo viac susedných trojuholníkov. Môže sa porušiť podmienka 2-manifold.

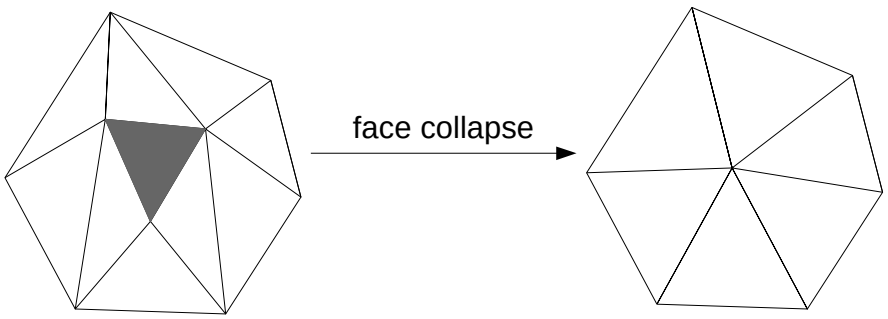
1.2.2 Vertex pair collapse

Vertex pair collapse operátor spája dva žiadnou hranou nespojené vrcholy. Na rozdiel od edge collapse, žiadne trojuholníky sa nevymazávajú. Tento operátor sa dá použiť na zmenu topológie meshe. Pretože spája doposiaľ nespojené vrcholy,

dokáže takto spojiť dve komponenty meshe, prípadne vie uzavrieť dieru alebo tunel.

1.2.3 Face/triangle collapse

Nahradením troch vrcholov v_a, v_b a v_c trojuholníka do nového vrchola v_n je operátor triangle collapse. Ilustrácia face collapsu je na obrázku 1.2. Nový trojuholník v_n môže byť podobne ako pri edge collapse buď úplne nový vrchol, alebo jeden z pôvodných. Tento operátor je ekvivalentom dvoch edge collapsov. Pretože pri face collapse je možné v jednom kroku odstrániť až štyri trojuholníky naraz (jeden kolabujeme, trojuholníky so spoločnými hranami by sa stali degenerovanými — tie treba odstrániť), je tento operátor rýchlejší ako edge collapse. Toto však môže byť aj nevýhoda, pretože edge collapse by mohol byť pružnejší a vedel by dosiahnuť menšiu globálnu chybu.



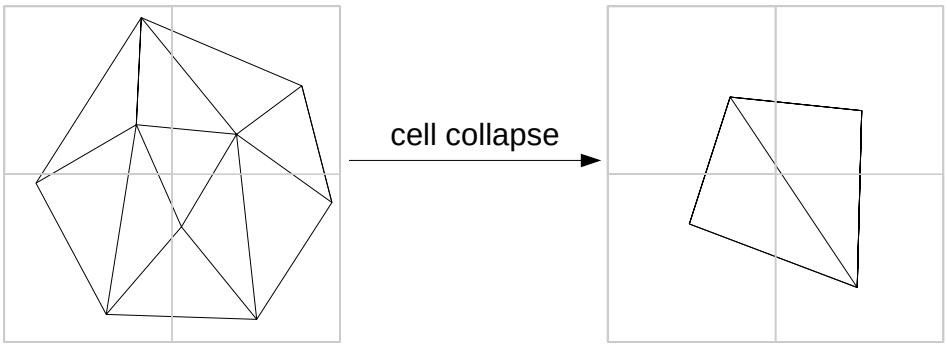
Obr. 1.2: Face collapse operátor.

1.2.4 Cell collapse

Cell collapse operátor spája vrcholy z nejakej oblasti do jedného vrcholu. Hrany vedúce do pôvodných vrcholov sa napoja na nový vrchol bunky. Ilustráciu cell collapse operátoru je vidieť na obrázku 1.3. Odstrániť sa musia degenerované trojuholníky a násobné hrany. Existujú rôzne prístupy k rozdeleniu meshe do oblastí. Príkladom môže byť pravidelná mriežka alebo iné rozdelenie priestoru (napríklad implementovaný Octree). Vrchol, ktorým budú nahrádzané vrcholy bunky môže byť vypočítaný (minimalizujúci chybu), alebo jeden z pôvodných vrcholov siete.

1.3 Decimačné frameworky

Doposiaľ boli predstavené operátory zjednodušujúce sieť. Táto sekcia sa venuje popisu vyšších algoritmov, ktoré využívajú prezentované operátory. Cieľom týchto frameworkov je vybrať správny operátor (ak sa využívajú viaceré) a správne operácie (miesta aplikovania operátoru) tak, aby decimácia dola čo najefektívnejšia (vonkajšia optimalizácia). Decimačné operátory väčšinou riešia vnútornú optimalizáciu (napr. výber/výpočet nového vrcholu pri face collapse) a framworky sú zodpovedné za vonkajšiu optimalizáciu — v akom poradí a ktoré operácie sa



Obr. 1.3: Cell collapse operátor. Vrcholy vo vnútri každej bunky sú nahradené jedným vrcholom.

budú vykonávať.

Ako bude popísané neskôr, výber a nastavenie algoritmu nie je len otázkou kvality výstupu, ale môže mať aj zásadný dopad na výkon, resp. rýchlosť celej decimácie.

1.3.1 Neoptimalizujúci

Neoptimalizujúci decimačný framework vykonáva operácie v ľubovoľnom poradí, teda nezaoberá sa vonkajšou optimalizáciou. Tento prístup môže byť použitý napríklad pri decimácii s cell collapse operátorom s rozdelením priestoru na uniformnú mriežku, kde každá operácia vytvára približne rovnakú chybu.

1.3.2 Greedy

Greedy framework vykonáva operácie v poradí podľa váhy/chyby, ktorá vznikne po vykonaní operácie. Po prevedení operácie znovu vyhodnotí operácie susediace s vykonanou.

```
Pre každú operáciu op
        VypočítajCenu(op)
        Q->Zaraď(op)

Pokiaľ Q neprázdna
        op = Q->Min()
        Vykonaj(op)
        Pre každú susednú operáciu so
                VypočítajCenu(so)
                Q->ZmeňCenu(so)
```

Väčšina času bude pravdepodobne strávená vo funkciách VypočítajCenu() a Vykonaj().

1.3.3 Lazy

Alternatívou ku greedy môže byť lazy framework. Je možné si všimnúť, že v prípade predchádzajúceho algoritmu môže dochádzať k veľkému počtu volaní

funkcie VypočítajCenu() na nejakej operácii predtým než je v skutočnosti vykonaná. Tento algoritmus sa preto snaží minimalizovať počet volaní VypočítajCenu() tak, že namiesto okamžitého prepočítania váh okolitých operácii ich označí ako neaktuálne. V prípade, že je z fronty Q vybraná operácia, ktorá má neaktuálnu váhu, nie je vykonaná, ale je prepočítaná a opäť zaradená do fronty.

```
Pre každú operáciu op
        VypočítajCenu(op)
        op->aktuálna = true
        Q->Zaraď(op)

Pokiaľ Q neprázdna
        op = Q->Min()
        Ak op->aktuálna == true
                Vykonaj(op)
                Pre každú susednú operáciu so
                        so->aktuálna = false
        Inak
                VypočítajCenu(op)
                op->aktuálna = true
                Q->ZmeňCenu(op)
```

Cohen [5, str. 90] vo svojej dizertačnej práci porovnával počet volaní funkcie počítajúcej cenu operácie v prípade greedy a lazy algoritmu pri používaní edge collapse operátoru. Porovnanie prebehlo na známom modeli „Stanford Bunny“ [6] a počet volaní bol 1 372 122 pre greedy a 436 817 pre lazy algoritmus. V prípade jeho implementácie sa jednalo o viac než 2,5-násobné zrýchlenie. Zároveň však podľa Cohena [5, str. 89] chyba pri úplnom a lazy vyhodnocovaní rastie rovnako rýchlo počas procesu decimácie.

1.3.4 Estimating

Estimating algoritmus poskytuje ďalšiu alternatívu ku greedy algoritmu. Namiesto výpočtu skutočnej ceny operácie sa použije oveľa rýchlejší odhad ceny. Prípadne je možné ešte skombinovať prístup s lazy vyhodnocovaním. Práve kombinácia oboch prístupov (lazy a estimating) je použitá v implementácii jednej z decimácii tejto práce (viď 3.2).

Zhrnutie

Kapitola uviedla základné prístupy k decimácii sieti na základe cieľa, teda či je dôležitá veľkosť výslednej siete alebo je kladený dôraz na kvalitu výstupu. Decimáciu predstavila ako optimalizačný problém a rozčlenila ho na vnútornú — lokálne operátory — a vonkajšiu — radenie operátorov — optimalizáciu. Navrhnuté boli niekoľké operátory využívané vo vnútornej optimalizácii. Spomenuté bolo napríklad optimálne umiestňovanie vrcholov, ale nebol uvedený spôsob. Tomu sa bude venovať nasledujúca kapitola.

2. Prehľad chybových metrík

V predchádzajúcej kapitole boli predstavené algoritmy decimujúce trojuholníkové siete. Veľa krát boli spomenuté pojmy ako cena alebo chyba. Táto kapitola sa venuje práve meraním chyby (v anglickej literatúre error). Väčšina decimačných algoritmov je priamo závislá na využívaní nejakej chybovej metriky — či už pri radení operácii alebo pri prevádzaní operácie (napríklad výber nového vrcholu). Meranie chyby/rozdielu však môže byť využité na porovnanie viacerých decimačných algoritmov. Porovnaniu implementovaných decimácii sa venuje posledná kapitola (viď Výsledky).

Totálna a inkrementálna chyba

Dôležitou otázkou je nie len ako merať chybu, ale aj vzhľadom na čo merať chybu. Viaceré algoritmy využívajú totálnu chybu, teda vyhodnocujú aktuálnu chybu vzhľadom na pôvodnú sieť. Iný prístup môže byť využitie inkrementálnej chyby, teda chyba sa vyhodnotí vzhľadom na sieť pred vykonaním jednej operácie. Hoppe vo svojom článku prezentuje inkrementálnu chybu pod pojmom memoryless simplification a tvrdí, že jej využitím je možné dosiahnúť lepšie výsledky [7]. Totálna chyba je výhodná na porovnanie celkového výsledku decimácie s pôvodnou sieťou.

2.1 Geometrická chyba

Geometrická chyba vzniká zmenou tvaru povrchu decimovanej meshe. Decimačné algoritmy sa vo všeobecnosti snažia pri prevádzaní operácii túto chybu minimalizovať, čím sa najlepšie zachováva pôvodný tvar. Vrcholy trojuholníkových sietí sú body v trojrozmernom priestore. V kartézskom priestore $\mathbb{R}^n$ je definovaný ako

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{2.1}$$

2.1.1 Hausdorfova vzdialenosť

V praxi je však zaujímavejšia vzdialenosť medzi dvoma povrchmi. Vzdialenosť bodu p od povrchu $\mathcal{S}'$ je definovaná v [8] ako

$$d(p, \mathcal{S}') = \min_{p' \in \mathcal{S}'} \|p - p'\|_2. \tag{2.2}$$

Hausdorffova vzdialenosť medzi povrchmi $\mathcal{S}$ a $\mathcal{S}'$ potom je

$$d(\mathcal{S}, \mathcal{S}') = \max_{x \in \mathcal{S}} d(p, \mathcal{S}'). \tag{2.3}$$

Dôležité pozorovanie je, že Hausdorffova vzdialenosť medzi povrchmi nie je symetrická [8]. To znamená, že vo všeobecnosti platí $d(\mathcal{S}, \mathcal{S}') \neq d(\mathcal{S}', \mathcal{S})$. Nevýhodou Hausdorffovej vzdialenosti je, že udáva maximálnu chybu. Maximálna chyba

teda môže byť pomerne vysoká, zatiaľ čo priemerná môže byť podstatne menšia. V praxi sa preto skôr využíva priemer namiesto maxima [3]. V tejto práci sa Hausdorffova vzdialenosť bude využívať v kapitole venujúcej sa porovnaniu výsledkov (viď Výsledky).

2.2 Chyba atribútov

Okrem geometrie je pre niektoré aplikácie dôležité zachovanie atribútov siete. Medzi časté atribúty siete patria textúry a normály. Tie bývajú uložené vo vrcholoch. Niektoré siete (a to je aj prípad skenov tvárí, ktorým sa venuje táto práca) môžu byť niektoré atribúty rôzne pre jeden vrchol. Správne zachovanie atribútov môže mať zásadný dopad na výslednú kvalitu meshe.

2.2.1 Textúry

Medzi dôležité atribúty patria textúry. Textúrový atribút je tvorený dvojicou (u, v) koordinátov fotografie. Pri decimácii sa vrcholy môžu presúvať a vtedy je nevyhnutné správne prepočítať aj súradnice textúr. Inak by sa textúra na jednotlivých trojuholníkoch namapovala s rôznymi deformáciami.

2.2.2 Normály

Normály sú atribútom využívaným pri výpočte osvetľovania. Ich zachovanie (resp. správna úprava) sa tiež môžu zásadným spôsobom podpísať pod vizuálnu kvalitu zdecimovanej meshe. Normály sú väčšinou vektory s veľkosťou jedna. V prípade, že by bola len požiadavka na kvalitné tieňovanie, namiesto zachovania pôvodných normál (resp. ich úpravou), stačí normály po decimácii prepočítať na základe s vrcholom susediacich trojuholníkov. Okrem toho sa dajú normály využiť na detekciu „zlých“ operácii. Medzi takéto operácie patrí napríklad edge collapse, pri ktorom vznikne mesh fold-over, teda operácia, kedy sa nový vrchol umiestni tak, že prekryje inú časť meshe. V tomto prípade je operácia sprevádzaná veľkými zmenami v smere normál typicky viac ako 90° [3].

2.3 Chybové kvadriky

Chybové kvadriky sú metriky často využívané v decimačných algoritmoch. Jedná sa o chybovú funkciu zakódovanú do maticovej podoby. Výhodou tejto reprezentácie je pomerne malá priestorová zložitosť, ale aj ako bude ukázané neskôr, rýchle vyhodnocovanie. V literatúre sa používa skratka *QEF* z anglického quadratic error function, prípadne *QEM* z quadratic error metric.

2.3.1 Jednoduché kvadriky

S prvou kvadrikou zachytávajúcou len geometrickú chybu prišli v [9] Garland a Heckbert. Vyjadruje súčet vzdialeností bodu v od rovín susediacich s pôvodnými

vrcholmi:

$$\begin{aligned}\Delta(v) &= \sum_{p \in roviny(v)} (v^\top p)(p^\top v) \\ &= \sum_{p \in roviny(v)} v^\top (pp^\top) v \\ &= v^\top \left(\sum_{p \in roviny(v)} (pp^\top) \right) v,\end{aligned} \tag{2.4}$$

kde $p = (a \quad b \quad c \quad d)^\top$ reprezentujúce rovinu definovanú rovnicou $ax + by + cz + d = 0$, kde $a^2 + b^2 + c^2 = 1$. Výhodou je, že si treba pamätať iba 16 čísel. Pri vyhodnocovaní edge collapsu je potrebné spraviť zjednotenie rovín. Tu však stačí sčítať QEF. V prípade, že zjednocované množiny sú disjunktné, súčet je správny. Ak však nie sú množiny disjunktné, tak bude rovina započítaná viackrát, ale celkovo nie viac ako 3 krát [9] počas celej decimácie.

Reprezentáciu neskôr upravil Hoppe [7]:

$$Q^v(\mathbf{v}) = \sum_{f \ni v} plocha(f) \cdot Q^f(\mathbf{v}). \tag{2.5}$$

Funkcia $Q^f(\mathbf{v})$ je definovaná ako:

$$Q^f(\mathbf{v}) = (n^\top \mathbf{v} + d)^2 = \mathbf{v}^\top (nn^\top)\mathbf{v} + 2dn^\top \mathbf{v} + d^2, \tag{2.6}$$

kde n je normálový vektor plochy a $d = n^\top p$ (p je jeden z určujúcich vrcholov roviny/trojuholníka). Táto funkcia sa reprezentuje aj ako:

$$Q^f(\mathbf{v}) = \mathbf{v}^\top A\mathbf{v} + 2b^\top \mathbf{v} + c, \tag{2.7}$$

kde $A = nn^\top$, $b = dn$ a $c = d^2$.

2.3.2 Rozšírené kvadriky s atribútmi

Merať a optimalizovať iba geometrickú chybu nemusí vždy stačiť. Meshe veľakrát prichádzajú s atribútmi, ktoré je treba zachovať, resp. tiež optimalizovať. Garland a Heckbert [10] navrhli rozšíriť QEF do n dimenzii, teda napríklad vrchol farebnej meshe $p = (p_x \quad p_y \quad p_z \quad p_r \quad p_g \quad p_b)^\top$. Hoppe [7] však argumentuje, že táto metrika môže podceniť geometrickú chybu a že navyše je nutné pamätať si až $(4 + m)(5 + m)/2$ koeficientov. Preto navrhol novú kvadratickú chybovú metriku. Hoppe-ova metrika interpoluje jednotlivé zložky atribútov v 3D a na základe toho počíta geometrickú a atribútovú chybu.

Hoppe-ová chybová funkcia pre tvár f je definovaná ako

$$Q^f(\mathbf{v} = \binom{\mathbf{p}}{\mathbf{s}}) = Q_p^f(\mathbf{v}) + \sum_{j=1}^{m} Q_{s_j}^f(\mathbf{v}), \tag{2.8}$$

kde $Q_p^f(\mathbf{v})$ odpovedá druhej mocnine vzdialenosti $\mathbf{p}$ od priemetu $\mathbf{p}'$ na rovinu $P \subset \mathbb{R}^3$ a $Q_{s_j}^f(\mathbf{v})$ je druhá mocnina rozdielu hodnoty atribútu $\mathbf{s}$ a interpolovanej hodnoty $\mathbf{s}'$.

$Q_p^f(\mathbf{v})$ je rozšírením 2.7 o nuly:

$$Q_p^f = (A, b, c) = \left(\left(\begin{array}{c|c} nn^\top & \ddots\, 0\, \ddots \\ \hline \ddots\, 0\, \ddots & \ddots\, 0\, \ddots \end{array} \right), \begin{pmatrix} dn \\ \hline 0 \end{pmatrix}, d^2 \right). \tag{2.9}$$

Na odvodenie atribútovej chybovej funkcie $Q_{s_j}^f(\mathbf{v})$ sa zavedie lineárna funkcia

$$\hat{s}_j(\mathbf{p}) = g_j^\top \mathbf{p} + d_j,$$

ktorá reprezentuje očakávanú hodnotu atribútu j v bode $\mathbf{p} \in \mathbb{R}^3$. g_j je gradient skalárnej funkcie atribútu na rovine f danej trojuholníkom $((\mathbf{p_1}, \mathbf{s_1}), (\mathbf{p_2}, \mathbf{s_2}), (\mathbf{p_3}, \mathbf{s_3}))$. Parametre g_j a d_j sa získajú riešením lineárnej rovnice

$$\begin{pmatrix} \mathbf{p}_1^\top & 1 \\ \mathbf{p}_2^\top & 1 \\ \mathbf{p}_3^\top & 1 \\ n^\top & 0 \end{pmatrix} \begin{pmatrix} g_j \\ \\ d_j \end{pmatrix} = \begin{pmatrix} s_{1,j} \\ s_{2,j} \\ s_{3,j} \\ 0 \end{pmatrix}. \tag{2.10}$$

Potom $Q_{s_j}^f(\mathbf{v}) = (\hat{s}_j(\mathbf{p}) - s_j) = (g_j^\top \mathbf{p} + d_j - s_j)^2$ upravené do tvaru ako 2.9:

$$Q_{s_j}^f = \left(\left(\begin{array}{c|ccc} nn^\top & \ddots\, 0\, \ddots & \ddots\, 0\, \ddots & \ddots\, 0\, \ddots \\ \hline \\ \ddots\, 0\, \ddots & \ddots\, 0\, \ddots & 0 & \ddots\, 0\, \ddots \\ -g_j^\top & \cdots 0 \cdots & 1 & \cdots 0 \cdots \\ \ddots\, 0\, \ddots & \ddots\, 0\, \ddots & 0 & \ddots\, 0\, \ddots \end{array} \right), \begin{pmatrix} d_j g_j \\ \hline \\ 0 \\ -d_j \\ 0 \end{pmatrix}, d_j^2 \right), \tag{2.11}$$

kde 1 sa vyskytuje na pozícii $A_{3+j,3+j}$ a $-d_j$ na b_{3+j}. Spojením 2.9 a 2.11 dohromady dostávame:

$$Q^f = \left(\left(\begin{array}{c|ccc} nn^\top + \sum_j g_j g_j^\top & -g_1 & \cdots & -g_m \\ \hline \\ -g_1^\top & & & \\ \vdots & & I & \\ -g_m^\top & & & \end{array} \right), \begin{pmatrix} dn + \sum_j d_j g_j \\ \hline \\ -d_1 \\ \vdots \\ -d_m \end{pmatrix}, d^2 + \sum_j d_j^2 \right). \tag{2.12}$$

Na uloženie QEF v takomto tvare je potrebných $11 + 4m$ čísel, pretože iba prvé tri riadky a stĺpce matice A sú husté a veľkú časť matice tvorí jednotková matica I, ktorú je možné uložiť úsporne.

Váhy atribútov

QEF v tvare 2.12 umožňuje priradiť atribútom váhy, tak že $Q = Q_p + \sum_j \lambda_j^2 Q_{s_j}$. Priradenie váh atribútom nemá za následok potrebu zmeny reprezentácie. Hoppe vo svojom článku [7] uvádza, že ak sa nastavia váhy $\lambda_j \to 0$, tak kvalita výslednej siete je podstatne vyššia. Toto sa dá ilustrovať zavedením nadroviny $P' \subset \mathbb{R}^{3+m}$, kde $\mathbb{R}^{3+m}$ predstavuje spoločný priestor geometrie a atribútov. Nastavením váh $\lambda_j \to 0$ bude táto rovina viac rovnobežná s geometrickou rovinou $P \subset \mathbb{R}^3$. Celkovo je potom matica A reprezentujúca QEF numericky stabilnejšia.

2.3.3 Využitie kvadrík na vnútornú optimalizáciu

Minimalizovaním $Q^v(\mathbf{v})$ (minimum kvadratická funkcia dosahuje tam, kde sa jej gradient $\nabla Q^f(\mathbf{v}) = 2A\mathbf{v} + 2b$ rovná 0; teda po úpravách riešením lineárnej rovnice $A\mathbf{v}_{min} = -b$) dostaneme novú pozíciu $\mathbf{v}_{min}$. Tú je možné využiť pri vnútornej optimalizácii. V tejto práci je využitá na umiestenie nových vrcholov v octre decimačnom algoritme, ako aj pri face collapse.

2.3.4 Zlá podmienenosť sústavy

Ak okolie bodu v mesh-i má Gaussovú krivosť rovnú nule, teda je buď rovné alebo valcové, je lineárny systém na hľadanie minima zle podmienený [7]. Riešením je nastavenie bodu $\mathbf{p} = (\mathbf{p_1} + \mathbf{p_2})/2$ v prípade edge collapsu následné dopočítanie atribútov.

Zhrnutie

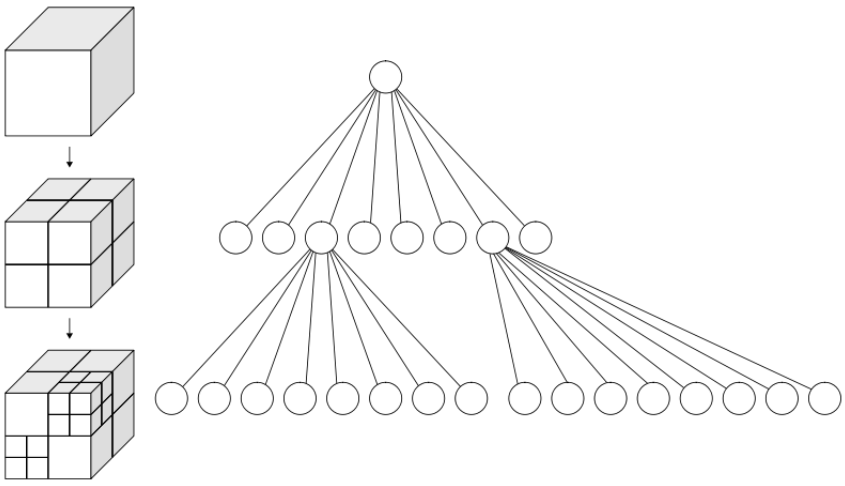
Kapitola sa venovala chybovým metrikám. Meranie chyby nie je dôležité iba na celkové vyhodnotenie výsledku decimácie, ale je potrebné už počas procesu redukovania siete. Využitie si metriky nájdu pri vonkajšej (radenie operácii), ale aj vnútornej optimalizácii (umiestňovanie vrcholov a výpočet atribútov). Nasledujúca kapitola bude integrovať poznatky z tejto a predchádzajúcej kapitoly do samostatných decimácii.

3. Implementované decimácie

V tejto kapitole budú predstavené konkrétne algoritmy decimácie, ktoré boli v implementačnej časti práce využité. Predstavené algoritmy spájajú dohromady poznatky spomenuté v prvých dvoch kapitolách. Najprv bude predstavená dátová štruktúra octree a dve konkrétne decimácie využívajúce túto štruktúru. Ďalej bude predstavená decimácia využívajúca face collapse operátor. Na záver kapitoly je uvedený postup opravy chýb v textúre.

3.1 Octree

Prvé dve decimácie využívajú dátovu štruktúru (DS) octree. Octree je stromová DS, ktorá rekurzívne rozdeľuje 3D priestor na osem oktantov. Každý vnútorný uzol má teda osem potomkov (v tejto práci je pripustená existencia len potomkov, ktorý obsahujú nejaký vrchol). Jednoduchá schéma stromu a odpovedajúceho priestoru je na obrázku 3.1.



Obr. 3.1: Schéma octree. V ľavo: Rekurzívne delenie kocky na oktanty. V pravo: Odpovedajúci octree. Prebrané z [11].

Každý vrchol scény sa uloží do jedného uzlu[1]. Uzol stromu octree udáva jeho pozíciu a veľkosť. Ak je to vnútorný uzol, obsahuje aj odkazy na potomkov. Pri konkrétnych decimáciach sú v uzloch uložené aj iné dáta: napríklad vrchol reprezentujúci uzol stromu, QEF, alebo aj nová textúra a normála. Octree má na rozdiel od rovnomerného rozdelenia priestoru výhodu, že dokáže prispôsobiť delenie s ohľadom na detaily meshe. Teda ak je nejaký vrchol ďaleko od ostatných vrcholov, bude reprezentovaný väčším uzlom (s väčším rozmerom a bližšie ku koreňu stromu). Naopak, ak nejaká oblasť meshe je veľmi hustá, vrcholy budú obsiahnuté vo vrcholoch s menšou veľkosťou. Táto vlastnosť umožňuje decimačným algoritmom lepšie zachovať geometrické detaily.

[1]Koordináty sú uložené v plávajúcej desatinnej čiarke; rovnosť nastáva, ak dva body sú v nejakom, typicky užívateľom nastavenom, intervale. Teda neskolabované uzly môžu obsahovať viac vrcholov (viď kapitolu 6).

Decimácia veľkých meshí

Scheffer a Warren navrhujú použitie octree DS na decimáciu meshí, ktoré sa nevojdú do pamäte [12]. Táto práca sa venuje sieťam, ktoré sa do pamäti vojdú, preto implementované algoritmy vynechávajú a zjednodušujú niektoré kroky. Autori navrhli spôsob pridávania nových vrcholov do octree tak, aby každý vrchol kocky i predchádzal vrcholom kocky $i + 1$. Kľúč pre vrchol $(0.x_1x_2..., 0.y_1y_2...,$ $0.z_1z_2...)$ (vrcholy sú normované do intervalu $[0,1]^3$) je $0.x_1y_1z_1x_2y_2z_2...$. Takéto usporiadanie potom umožňuje postupné orezávanie (decimáciu) stromu tak, že veľkosť novej meshe ostane nanajvýš taká veľká ako je povolené maximum a nebudú sa odstraňovať vetvy, do ktorých by sa mohol pridať ešte nejaký vrchol.

Stavba stromu

Stavba stromu prebieha postupným pridávaním vrcholov. Na začiatku je koreň — uzol s veľkosťou a polohou nastavenou tak, aby tento uzol obsahoval všetky vrcholy scény. Pre pridávaný vrchol sa nájde najhlbší uzol, ktorý ho obsahuje. Ak je to uzol vnútorný, tak to znamená, že nový potomok je vložený do tohto uzlu a do neho sa uloží pridávaný vrchol. Ak je najhlbší uzol list a vrchol v liste sa rovná pridávanému[2], vloží sa do tohto uzlu. Inak sa pridá najhlbšiemu uzlu nový potomok. Do potomka sa skopíruje pôvodný vrchol uzlu a nasleduje pokus o vloženie pridávaného vrcholu do nového potomka. Takýto proces štiepenia pokračuje pokým sa nepodarí pridávaný a pôvodný vrchol uzlu vložiť do samostatných uzlov.

Decimácia na octree

Decímacie implementujúce octree redukujú počet vrcholov (a teda aj trojuholníkov) pomocou orezávania stromu. Typicky má každý uzol priradenú nejakú chybovú metriku, ktorá určuje chybu po skolabovaní vetvy octree pod daným uzlom. Uzly stromu sú vložené do minimovej haldy. Decimačný algoritmus vyberá z haldy uzol s najmenšou chybou a vykoná operáciu — cell collapse. Cell collapse operátor spraví z daného uzlu stromu list. Okrem orezania sa vypočíta pozícia vrcholu reprezentujúceho orezanú vetvu a nastaví sa do nového listu a upraví sa chybová metrika (napríklad decimácia veľkých sietí navrhovaná Scheafferom a Warrenom [12] využíva agregovanú QEF, teda QEF z odrezaných potomkov sa sčíta a vloží do uzlu).

Generovanie meshe z octree

Výstupom algoritmu je orezaný octree. Z neho je potrebné vygenerovať zdecimovanú mesh. Vrcholy novej meshe odpovedajú listom výstupného octree. Algoritmus generujúci novú sieť postupne prejde zoznam pôvodných trojuholníkov. Pre každý vrchol pôvodného trojuholníka vyhľadá najhlbší uzol. Pretože niektoré

[2]Koordináty sú uložené v plávajúcej desatinnej čiarke; rovnosť nastáva, ak dva body sú v nejakom, typicky užívateľom nastavenom, intervale. Teda neskolabované uzly môžu obsahovať viac vrcholov (viď kapitolu 6).

trojuholníky budú degenerované, nebudeme ich pridávať do novej meshe. Degenerovaný trojuholník je možné detegovať tak, že sa porovnajú indexy uzlov obsahujúce jednotlivé vrcholy. Ak sa niektorá dvojica rovná, trojuholník je degenerovaný a nebude pridaný.

3.1.1 Decimácia s využitím octree a vertex-plane distance chybovej metriky

Jedna z implementovaných decimácii v tejto práci využíva kombináciu octree, lazy algoritmus (1.3.3) a chybovú metriku vertex-plane distance s jednoduchou atribútovou metrikou. Decimácia nemení polohu pôvodných vrcholov ale odstraňuje a upravuje trojuholníky meshe.

Chybová metrika

Každý uzol si udržuje zoznam vrcholov. Geometrická chyba vrcholu je definovaná ako

$$Chyba_{uzol}^{geom}(\mathbf{p}) = \sum_{v \in Vrcholy(uzol)} (\mathbf{p} \cdot \mathbf{n_t}(v) + \mathbf{D}(v))^2, \tag{3.1}$$

kde v je index vrcholu, $\mathbf{p}$ je umiestenie nového vrcholu, ďalej $\mathbf{D}(v)$ vzdialenosť roviny (trojuholníka) od počiatku a $\mathbf{n_t}(v)$ vráti normálu roviny susediaceho s vrcholom.

Decimácia využíva aj jednoduché meranie chyby textúry a normál:

$$Chyba_{uzol}^{textúra}(\mathbf{t}) = \sum_{v \in Vrcholy(uzol)} (\mathbf{t}(v) - \mathbf{t})^2, \tag{3.2}$$

$$Chyba_{uzol}^{normála}(\mathbf{n}) = \sum_{v \in Vrcholy(uzol)} (\mathbf{n}(v) - \mathbf{n})^2, \tag{3.3}$$

kde $\mathbf{n}$ a $\mathbf{t}$ je normála a textúra, $\mathbf{n}(v)$ a $\mathbf{t}(v)$ sú funkcie vracajúce textúru a normálu odpovedajúceho vrcholu. Celková chybová metrika je váženým súčtom 3.1, 3.3 a 3.2: $Chyba_{uzol}(\mathbf{p}, \mathbf{t}, \mathbf{n}) = \lambda_{geom} \cdot Chyba_{uzol}^{geom}(\mathbf{p}) + \lambda_{textúra} \cdot Chyba_{uzol}^{textúra}(\mathbf{t}) + \lambda_{normála} \cdot Chyba_{uzol}^{normála}(\mathbf{n})$, kde váhove koeficienty určuje užívateľ.

Cell collapse operátor

Cell collapse operátor, ktorý je využitý v tomto algoritme vyberie zo zoznamu vrcholov v kolabovanom podstrome vrchol s najmenšou chybou. Po kolapse sa do uzlu pridajú všetky vrcholy podstromu (duplicity sa vymažú). Po tejto oprácii vznikne nový list a tento je treba zaradiť do poradia na decimáciu (do minimovej haldy).

Generovanie meshe

Generovanie meshe prebieha tak, ako je popísané v časti Generovanie meshe z octree. Navyše je umožnené užívateľovi aj opravenie textúry (viď 3.3).

3.1.2 Decimácia s využitím octree a QEF s atribútmi

Ďalšia z implementovaných decimácii využívajúca DS octree využíva kvadratickú chybovú metriku (QEF) s atribútmi. Jedná sa o QEF navrhnutú Hoppe-om [7], ktorá je bližšie opísaná v 2.3.2.

Podobne ako v predchádzajúcej decimácii sa využíva lazy algoritmus na vonkajšiu optimalizáciu. Vnútorná optimalizácia — prevedenie cell collapse operátoru — minimalizuje QEF. To znamená, že vyrieši sústavu (opísané v 2.3.3), ktorá určí novú polohu a hodnoty atribútov (textúry a normály). Ak je sústava zle podmienená (viď 2.3.4), použije sa priemerná hodnota koordinátov a atribútov.

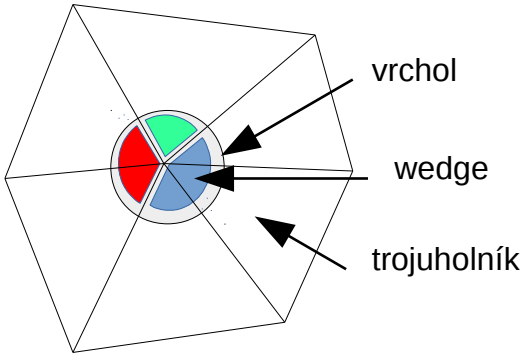
Opäť, generovanie meshe prebieha vyššie uvedeným spôsobom (Generovanie meshe z octree). Na miestach nespojitosti atribútov (najmä textúry) mohlo dôjsť k viditeľným chybám (pri skenoch tváre sa jedná najmä o švy vznikajúce v strede tváre), takže užívateľovi je umožnené vykonať opravu textúry (viď 2.3.2).

3.2 Face collapse decimácia s využitím rozšírenej QEF

Alternatívnou metódou k octree je face collapse. V tejto práci je implementovaná spolu s rozšírenou metrikou navrhnutou Hoppe-om [7]. Rozšírená metrika sa stará o lepšie zachovanie nespojitosti atribútov (najmä textúr).

Ako vonkajšia optimalizácia je použitý lazy algoritmus (viď 1.3.3), ktorý je spojený s estimating algoritmom (viď 1.3.4). Dôvodom na túto voľbu je pomerne veľké množstvo možných operácii, väčšie množstvo susedných operácii a časová náročnosť vyhodnocovania QEF. Odhad chyby je určený ako vážený priemer štvorcov rozdielov priemernej hodnoty vrcholov (pozície, textúry a normály) a hodnoty vrcholov. Rozšírená metrika sa využíva na výpočet nového vrcholu a atribútov po kolapse trojuholníka.

Hoppeom navrhované vylepšenie spočíva v zachovávaní nespojitých atribútov vo vrchole. Každý vrchol scény má zoznam wedge-í, ktoré sú na vrchole. Wedge [13] je dátová štruktúra udržujúca si informácie o vrchole, textúre a normále, ktoré sú spoločné pre niekoľko susediacich rohov vrcholu (viď obrázok 3.2). Trojuholník si namiesto pamätania vrcholov, ktoré ho definujú pamätá wedge-e. Wedge potom okrem geometrie zabezpečujú aj informácie o atribútoch. Ak viaceré trojuholníky využívajú ten istý vrchol a majú aj rovnaké atribúty, majú spoločnú wedge.



Obr. 3.2: Vrchol rozdelený na tri wedge-e. Každá wedge definuje pre rohy susediacich trojuholníkov atribúty.

Rozdelenie vrcholu na wedge-e je možné využiť na lepšie zachovanie nespojitosti atribútov. V predchádzajúcej decimácii (viď 3.1.2) bola použitá rozšírená kvadrika o atribúty. Chybová metrika pre vrchol bola určená týmto algoritmom ako súčet chybových metrík rohov trojuholníkov vychádzajúcich z daného bodu. Pre spojité hodnoty atribútov (teda 1 vrchol má 1 wedge) je toto riešenie v poriadku. Keď však sú v jednom vrchole rôzne atribúty, bude dochádzať k väčším chybám. Hoppe-om navrhované riešenie [7, sekcia č. 5] zachováva wedge. Namiesto sčítania všetkých rohov do jednej QEF s rozmerom $3+m$ (m je počet atribútov), postaví QEF s rozmerom $3+km$ ($k \geq 1$ je počet wedge-í vo vrchole). Zostavenie takejto QEF je priamočiaré z QEF pre wedge. QEF wedge-e je, obdobne ako pre vrchol v predchádzajúcej decimácii, súčtom kvadrík vypočítaných z trojuholníka (teda majú rozmer $3+m$). QEF pre vrchol s rozmerom $3+km$ je $Q^f = (A, b, c)$, kde

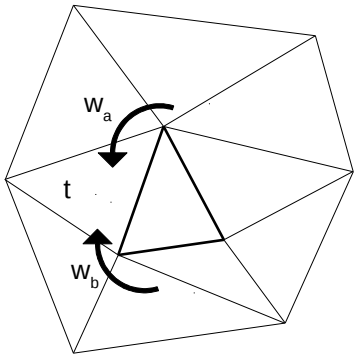
$$A = \left(\begin{array}{c|ccccccc} \sum_k (n_k n_k^\top + \sum_j g_{k,j} g_{k,j}^\top) & -g_{1,1} & \cdots & -g_{1,m} & \cdots & -g_{k,1} & \cdots & -g_{k,m} \\ \hline -g_{1,1}^\top & & & & & & & \\ \vdots & & & & & & & \\ -g_{1,m}^\top & & & & & & & \\ \vdots & & & & I & & & \\ -g_{k,1}^\top & & & & & & & \\ \vdots & & & & & & & \\ -g_{k,m}^\top & & & & & & & \end{array}\right), \tag{3.4}$$

$$b = \begin{pmatrix} \frac{\sum_k (d_k n_k + \sum_j d_{k,j} g_{k,j})}{} \\ -d_{1,1} \\ \vdots \\ -d_{1,m} \\ \vdots \\ -d_{k,1} \\ \vdots \\ -d_{k,m} \end{pmatrix} \tag{3.5}$$

a nakoniec

$$c = \sum_k (d_k^2 + \sum_j d_{k,j}^2). \tag{3.6}$$

Počas aplikovania face collapse operátoru sú zjednocované niektoré wedge-e. Ak wedge-e w_a a w_b presahujú do trojuholníka t, ktorý bude po face collapse-e odstránený, wedge-e sa zlúčia do novej (viď obrázok 3.3). QEF novej wedge-e w' sa bude rovnať súčtu pôvodných. Zlučovanie sa vykonáva na všetkých odstraňovaných trojuholníkoch.



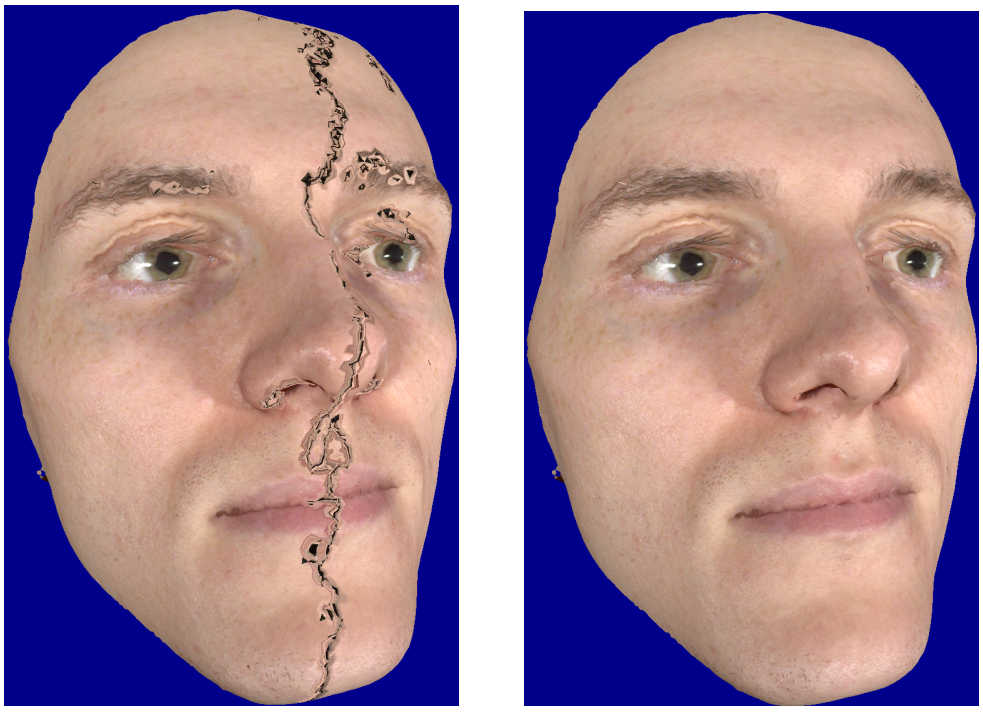
Obr. 3.3: Schéma zlučovania wedge-í počas face collapse-u.

Algoritmus si počas behu udržiava zoznam skolabovaných trojuholníkov. Tie je na konci treba z meshe odstrániť. Opäť mohli vzniknúť chyby textúry, užívateľovi je umožnené vykonať opravu.

3.3 Oprava chýb v textúre

Pri decimáciach vznikajú na miestach nespojitosti textúry chyby. Príklad takejto chyby je možné vidieť na obrázku 3.5. 3D skener vytvára dve snímky a tie spojí do jedného obrázka, ktorý slúži ako textúra scény (viď 3.4). Chyby vznikajú najmä v oblasti stredu tváre, okolí nosa a na čele, teda tam, kde textúra je namapovaná z rôznych snímok.

Obr. 3.4: Príklad textúry vytvorenej 3D skenerom.



(a) Chyba textúry po decimácii. (b) Implementovaná oprava chyby.

Obr. 3.5: Výsledok face collapse decimácie (26 000 trojuholníkov; váhy: $\lambda_{geom} = 1$, $\lambda_{textúra} = \lambda_{normály} = 10^{-17}$).

Chybové metriky nie sú poriadne schopné takýmto chybám predchádzať. Je preto potrebné na konci decimácie vykonať úpravu textúry. Implementovaná oprava vychádza z jednoduchej myšlienky: vrcholy nového trojuholníka sa zobrazia na rovinu určenú pôvodným trojuholníkom. Textúrové súradnice sú vrámci jedného trojuholníka vždy z jednej polovice textúry, takže je ich možné interpolo-

vať. Vypočítaním textúrových koordinátov v miestach priemetu nových vrcholov na pôvodnú rovinu sa získajú uspokojivé výsledky (viď obrázok 3.5).

Výpočet koordinátov textúry

Nech $\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3} \in \mathbb{R}^3$ sú vrcholy pôvodného trojuholníka a $\overrightarrow{\mathbf{u}} = \mathbf{p_2} - \mathbf{p_1}$, $\overrightarrow{\mathbf{v}} = \mathbf{p_3} - \mathbf{p_1}$ sú smerové vektory a $\overrightarrow{\mathbf{n}} = \overrightarrow{\mathbf{u}} \times \overrightarrow{\mathbf{v}}$ je normálový vektor. Pre danú opravu je zaujímavé zobrazenie na pôvodný trjuholník (resp. rovinu ním určenú). Riešením nasledujúcej sústavy sa získajú súradnice $\mathbf{s}^i_{\overrightarrow{\mathbf{u}}}, \mathbf{s}^i_{\overrightarrow{\mathbf{v}}}$ a $\mathbf{s}^i_{\overrightarrow{\mathbf{n}}}$ i-teho vrcholu vzhľadom na bázové vektory $\overrightarrow{\mathbf{u}}$, $\overrightarrow{\mathbf{v}}$, $\overrightarrow{\mathbf{n}}$:

$$\begin{pmatrix} \begin{pmatrix}\overrightarrow{\mathbf{u}}\end{pmatrix} \begin{pmatrix}\overrightarrow{\mathbf{v}}\end{pmatrix} \begin{pmatrix}\overrightarrow{\mathbf{n}}\end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix}\mathbf{s}^i_{\overrightarrow{\mathbf{u}}}\end{pmatrix} \begin{pmatrix}\mathbf{s}^i_{\overrightarrow{\mathbf{v}}}\end{pmatrix} \begin{pmatrix}\mathbf{s}^i_{\overrightarrow{\mathbf{n}}}\end{pmatrix} \end{pmatrix} = \begin{pmatrix}\mathbf{p'}_i - \mathbf{p}_1\end{pmatrix}. \tag{3.7}$$

Ďalej sú dôležité len súradnice $\mathbf{s}_{\overrightarrow{\mathbf{u}}}$ a $\mathbf{s}_{\overrightarrow{\mathbf{v}}}$ (priemet na rovinu). Ostáva určiť nové textúry. Nech $\overrightarrow{\mathbf{s}} = \mathbf{t_2} - \mathbf{t_1}$, $\overrightarrow{\mathbf{t}} = \mathbf{t_3} - \mathbf{t_1}$ sú smerové vektory textúry. Súradnice novej textúry pre i-ty vrchol nového trojuholníka potom sú:

$$\mathbf{t'}_i = \begin{pmatrix} \mathbf{t}_{1,x} + \mathbf{s}^i_{\overrightarrow{\mathbf{u}}} \cdot \overrightarrow{\mathbf{s}}_x + \mathbf{s}^i_{\overrightarrow{\mathbf{v}}} \cdot \overrightarrow{\mathbf{t}}_x \\ \mathbf{t}_{1,y} + \mathbf{s}^i_{\overrightarrow{\mathbf{u}}} \cdot \overrightarrow{\mathbf{s}}_y + \mathbf{s}^i_{\overrightarrow{\mathbf{v}}} \cdot \overrightarrow{\mathbf{t}}_y \end{pmatrix}. \tag{3.8}$$

Detekcia vrcholov na opravu

Nie je potrebné opravovať všetky trojuholníky siete, ale len tie, ktoré majú veľkú chybu. Na skenoch tváre sa ukazuje nasledujúca detekcia účinná: ak je vzdialenosť[3] nového a pôvodného vrcholu vynásobená koeficientom 10^{-2} menšia ako vzdialenosť novej a pôvodnej textúry, je daný vrchol a celý trojuholník označený na opravu.

Zhrnutie

Spojením a doplnením postupov z prvých dvoch kapitol boli vytvorené tri kompletné decimačné algoritmy, ktoré boli v práci implementované. Okrem nich bola predstavená aj oprava mapovania textúr, ktoré sa počas procesu decimácie môžu na niektorých miestach pokaziť. Kvalitu výsledkov týchto decimácii a kvalitu opravy preverí nasledujúca kapitola.

[3]Geometrická vzdialenosť je vynormovaná tak, aby sa celá scéna vmestila do $[-1,1]^3$

4. Výsledky

Táto kapitola je venovaná výsledkom práce a ich porovnaniu s existujúcimi nástrojmi.

4.1 Geometrické porovnanie

Táto sekcia sa venuje geometrickému porovnaniu výsledných sietí. Orezané siete boli decimované pomocou troch implementovaných algoritmov na približne 10% svojej pôvodnej veľkosti. Porovnávanie prebehlo v programe `MeshLab`. Program vypočítava Hausdorffovú vzdialenosť medzi dvoma sieťami. Vzorkuje zdecimovanú sieť a pre každú vzorku vyhľadá najbližší bod v pôvodnej sieti. Vzorkované boli v porovnaní vrcholy a trojuholníky. Výstupom sú pre každé porovnanie tri údaje: maximálna zaznamenaná vzdialenosť, priemerná vzdialenosť vzoriek a kvadratický priemer vzdialenosti. Výsledky pre tri vybrané skeny tvári je možné vidieť v tabuľke 4.1.

Sieť	$\#\triangle$[1] **BBD**[2] $[mm]$	**Decimácia**[3]	$\#\triangle$	**Max** $[mm]$	**Priemer** $[mm]$	**RMS**[4] $[mm]$
jana.obj	227 272 229,030	f	25 999	0,737	0,015	0,029
		v	27 065	1,961	0,017	0,033
		vo	25 603	0,460	0,017	0,037
pepca.obj	280 161 249,069	f	25 998	0,462	0,015	0,027
		v	27 207	0,781	0,018	0,029
		vo	26 134	0,581	0,017	0,035
vaclav.obj	274 267 246,891	f	25 999	0,622	0,016	0,030
		v	27 058	0,599	0,019	0,031
		vo	25 964	0,436	0,018	0,038

[1] počet trojuholníkov siete

[2] veľkosť diagonály ohraničujúceho boxu scény (bounding box)

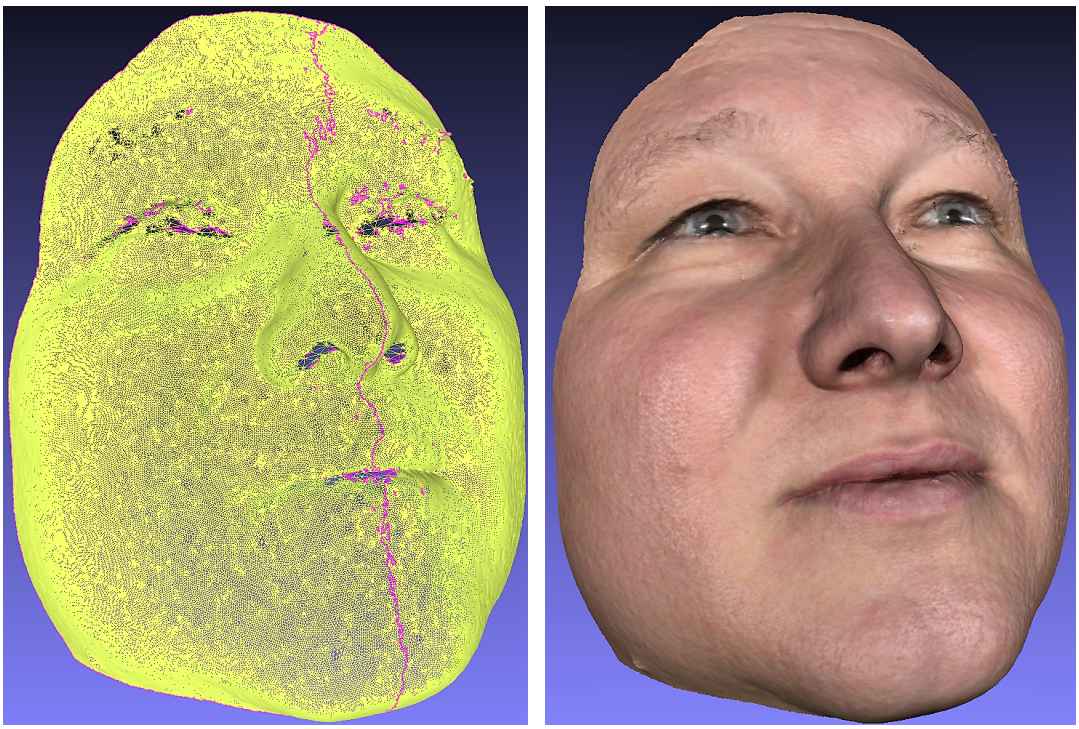
[3] f = face collapse, v = vertex clustering s použitím QEF, vo = vertex clustering využívajúci pôvodné vrcholy s metrikou vrchol-rovina

[4] kvadratický priemer

Tabuľka 4.1: Geometrické porovnanie decimovaných a pôvodných sietí.

Vo výsledkoch z tabuľky 4.1 je možné vidieť, že vzdialenosti sú v najhoršom maximálnom prípade menšie ako 1% vzhľadom na veľkosť diagonály obaľujúceho boxu (bounding box diagonal, BBD). Zaujímavejšia je však asi priemerná chyba, ktorá v prípade všetkých decimácii a decimovaných sieti vyšla ostro pod 10^{-2}% vzhľadom k BBD. Kvadratický priemer (RMS) ukazuje tiež odchýlky pod $2 \cdot 10^{-2}$%.

Najlepšie výsledky v priemeroch dosahuje face collapse decimačný algoritmus.



(a) Sieť, fialová farba zvýrazňuje nespojitosť textúry.

(b) Vytieňovaná sieť.

Obr. 4.1: Orezaný sken tváre *pepca.obj*. Scénu tvorí 280 161 trojuholníkov.

V maximálnych odchýlkach zaznamenáva najlepšie výsledky algoritmus využívajúci pôvodné vrcholy. Pod výrazne vyššiu maximálnu odchýlku v prípade vertex clustering algoritmu pri decimácii siete *jana.obj* sa moholo podpísať nedostatočné orezanie nepravidelných okrajov.

Celkovo však decimácie vykazujú v priemernom prípade porovnateľné výsledky.

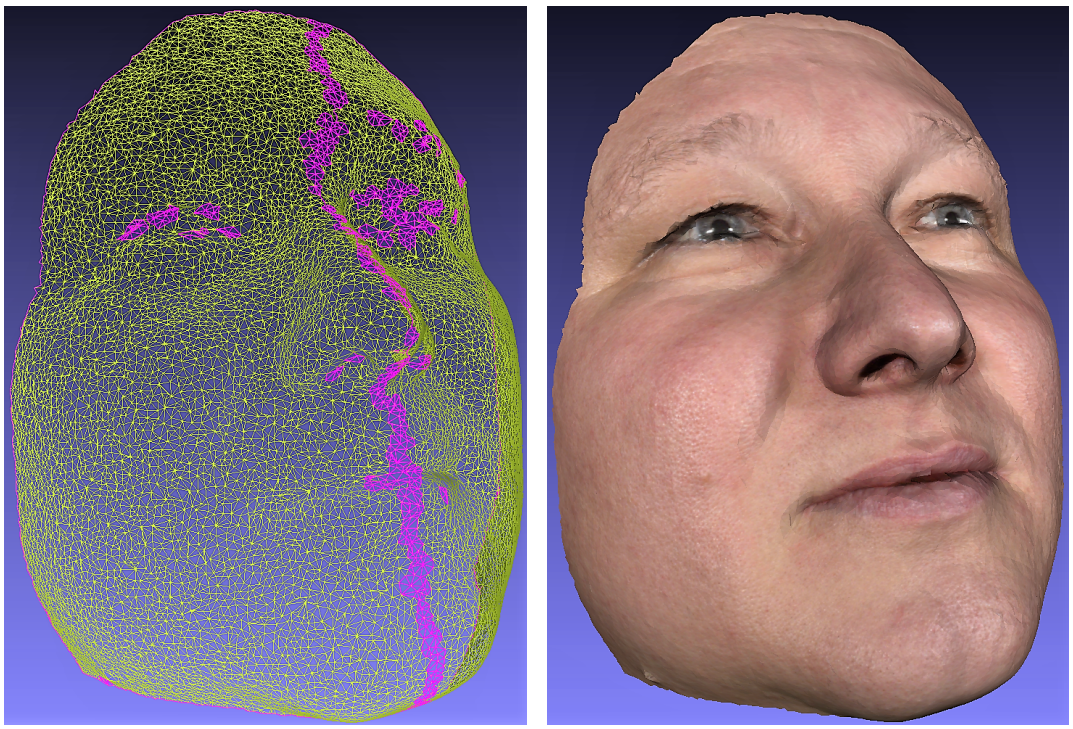
4.2 Vizuálne porovnanie

Cieľom práce nebolo iba zachovanie geometrie. Dôležitou súčasťou veľa scén sú aj iné atribúty, ako napríklad textúra a normály. V tejto sekcii budú porovnané výsledky vizuálne.

Na porovnanie implementovaných algoritmov bola vybraná sieť *pepca.obj*. Na obrázkoch 4.1 je orezaná originálna scéna. Na ľavej strane je sieť so zvýraznenými nespojitosťami v mapovaní textúry. Na pravej strane je vytieňovaná scéna.

Decimáciu scény využívajúcu face collapse operátor a rozšírenú kvadriku `QEF` s atribútmi a zachovávaním nespojitých atribútov vo vrcholoch pomocou `wedge`-ovej dátovej štruktúry je možné vidieť na obrázkoch 4.2. Výsledná sieť je pomerne rovnomerná. Ako je vidieť na pravej pravom obrázku, implementovaná oprava textúry sa dokázala dobre vysporiadať s nespojitosťou mapovania textúry.

O niečo menej rovnomerná je výsledná sieť po decimácii pomocou vertex clustering operátora a kvadriky `QEF` s atribútmi. Výsledná sieť je na obrázkoch 4.3. Oproti face collapse decimácii musela oprava textúr pracovať s väčším počtom



(a) Výsledná sieť, fialová farba zvýrazňuje nespojitosť textúry.

(b) Vytieňovaný výsledok.

Obr. 4.2: Výsledok face collapse decimácie. 25 998 trojuholníkov.

trojuholníkov (množstvo trojuholníkov vo fialovom páse je mierne väčšie). Vytieňovaný a opravený výsledok je však kvalitný a dostatočne hodnoverný.

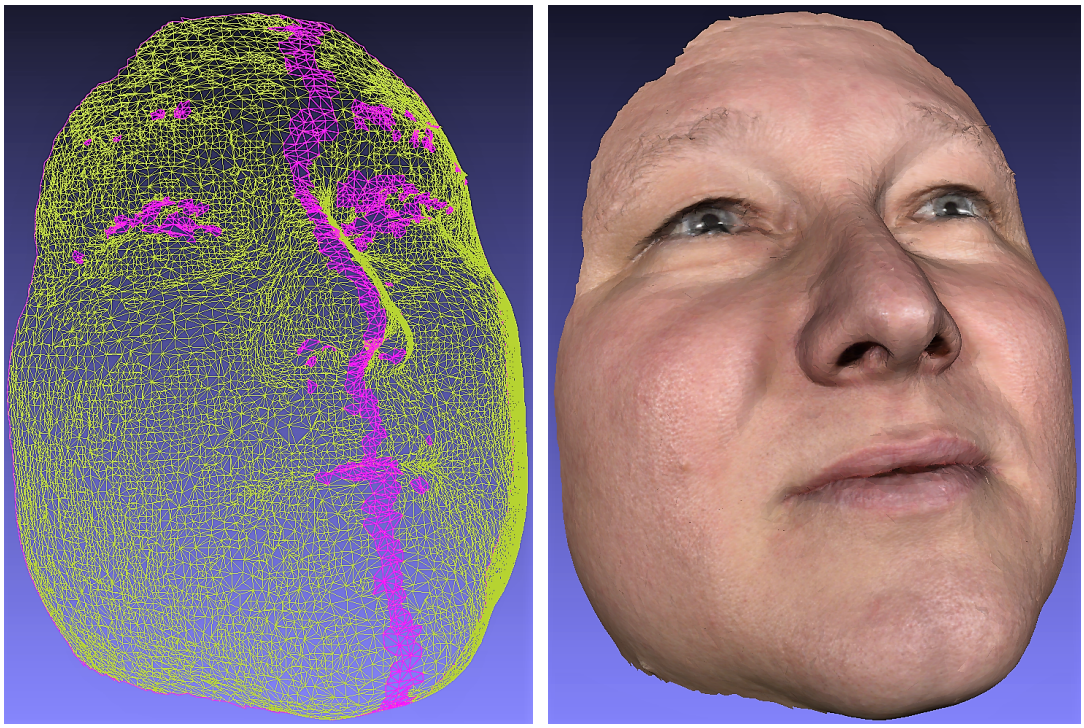
V porovnaní s ostatnými decimáciami dopadla najhoršie decimácia využívajúca pôvodné vrcholy a jednoduchú metriku vzdialenosti vrchol-stena. Na obrázku 4.4 je vidieť, že sieť je nepravidelná, niektoré trojuholníky sú v porovnaní s ostatnými zbytočne veľké. Toto sa podpísalo aj na mierne horšej kvalite pri tieňovaní. Na vytieňovanej sieti je možné pozorovať „kryhy“ (najviditeľnejšie sú na nose). Celkovo je však výsledok aj v tomto prípade uspokojivý.

4.3 Iné nástroje na decimáciu meshí

V súčasnosti je dostupných niekoľko nástrojov na decimáciu sietí. V tejto sekcii budú predstavené dva konkurenčné programy. Jeden open-source program `MeshLab` a druhý, komerčný `RapidForm`.

4.3.1 MeshLab

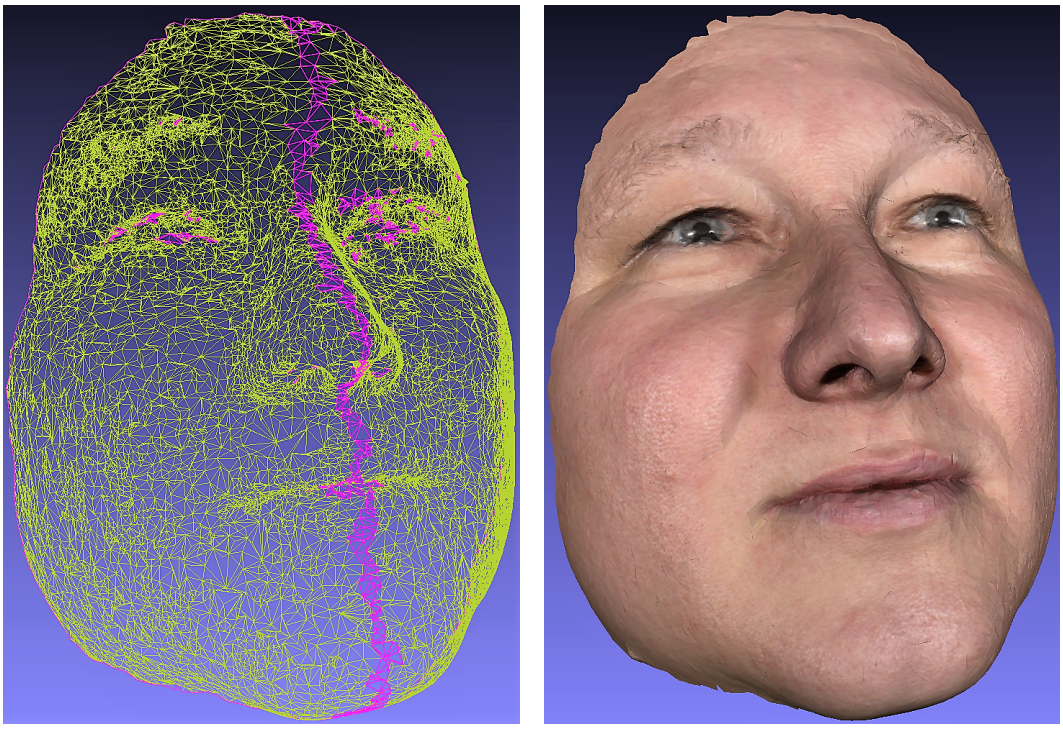
`MeshLab` je open-source program na editáciu 3D trojuholníkových sietí. Obsahuje množstvo algoritmov a nástrojov, ktoré dokážu sieť upravovať alebo merať jej vlastnosti. Z pohľadu tejto práce sú zaujímavé decimačné algoritmy. Program je vybavený hneď štyrmi implementáciami: marching cubes edge collapse, clustering decimácia, edge collapse s využitím kvadriky a edge collapse s kvadrikou s textúrami. Vhodná na porovnanie s decimáciami implementovanými v tejto práci bola práve posledná zmienená.



(a) Výsledná sieť, fialová farba zvýrazňuje nespojitosť textúry.

(b) Vytieňovaný výsledok.

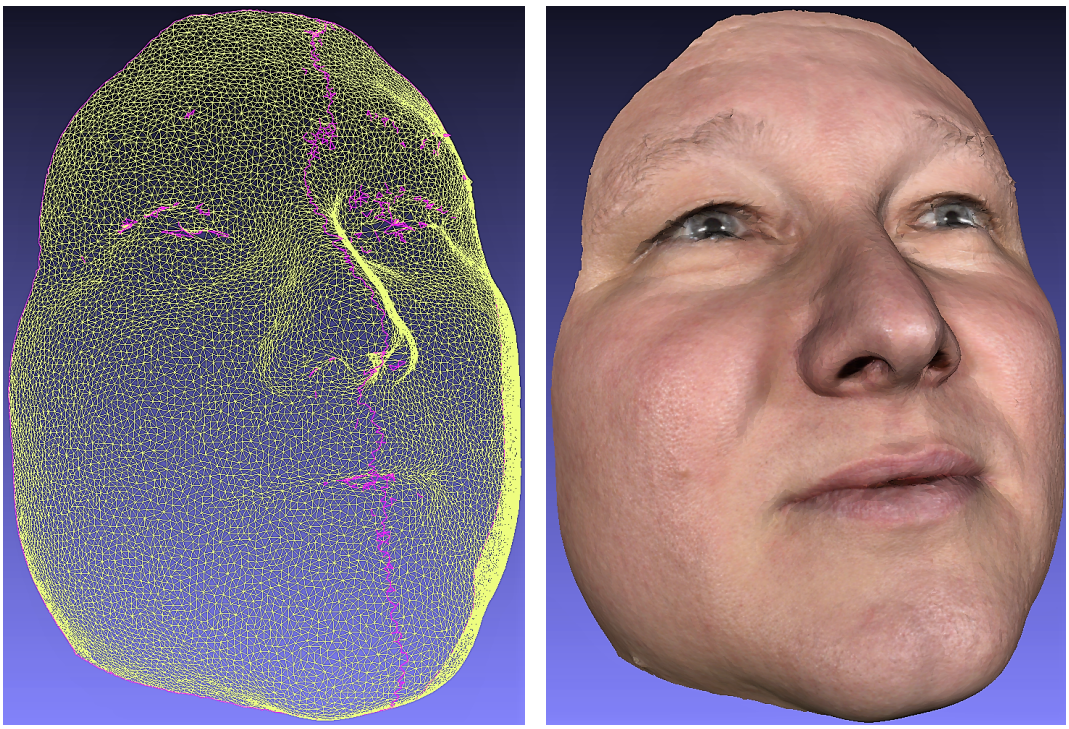
Obr. 4.3: Výsledok vertex clustering decimácie s využitím metriky `QEF`. 27 207 trojuholníkov.



(a) Výsledná sieť, fialová farba zvýrazňuje nespojitosť textúry.

(b) Vytieňovaný výsledok.

Obr. 4.4: Výsledok vertex clustering decimácie využívajúcej originálne vrcholy. 26 134 trojuholníkov.



(a) Výsledná sieť, fialová farba zvýrazňuje nespojitosť textúry.

(b) Vytieňovaný výsledok.

Obr. 4.5: Decimácia siete pomocou programu `MeshLab` na 25 999 trojuholníkov.

Zdecimovanú sieť je možné vidieť na obrázku 4.5. Štruktúra siete je mierne pravidelnejšia ako vo face collapse decimácii implementovanej v tejto práci. Menšia je aj časť s nespojitým mapovaním textúry. Vizuálne je sieť kvalitou totožná s face collapse decimáciou.

Decimácia	#△	**Max** $[mm]$	**Priemer** $[mm]$	**RMS**[1] $[mm]$
`MeshLab` edge collapse	25 999	0,368	0,020	0,031
Implementovaný face collapse	25 998	0,462	0,015	0,027

[1] kvadratický priemer

Tabuľka 4.2: Geometrické porovnanie najlepších decimačných algoritmov programu `Meshlab` a tejto práce.

V tabuľke 4.2 je geometrické porovnanie zdecimovanej siete pomocou `Meshlab`-u a face collapse decimácie implementovanej v tejto práci. Priemerne teda face collapse decimácia lepšie zachováva geometriu siete.

4.3.2 Rapidform

Ďalšou alternatívou je komerčný `Rapidform`. Pri vypracovávaní práce nebol softvér k dispozícii (cena licencie sa pohybuje od $10 000 do $30 000 [14]). Z dokumentácie [15, str. 494] nie je možné určiť presný algoritmus decimácie. Užívateľ

Obr. 4.6: Detail zdecimovanej siete tváre pomocou programu `Rapidform`.

môže určiť niekoľko atribútov decimácie ako je napríklad decimačný pomer, zachovanie detailov v členitých častiach siete, využitie originálnych vrcholov a pod. Výhodou decimácie implementovanej v tomto programe je pravidelnosť štruktúry výslednej siete. Z dostupných skenov tvári z Prírodovedeckej fakulty (z dôvodu ochrany osobných údajov nie sú tieto dáta súčasťou prílohy práce) je však vidieť, že miera zachovania textúry je horšia v porovnaní s programom `MeshLab` alebo decimáciami implementovanými v tejto práci. Ďalšou nevýhodou je, že výsledná textúra je rozdelená na trojuholníky a preskladaná do nového obrázku (tzn. textúra nie je v tvare ako na obrázku 3.4). To znamená, že nové mapovanie textúry na sieti je nepojité. To môže znamenať ťažkosti pri ďalšom spracovaní. Čiastočnú snímku výsledku decimácie je možné vidieť na obrázku 4.6. Viditeľná je strata detailu, textúra pôsobí rozmazaným dojmom (akoby efekt maľby štetcom) a v okolí úst sú viditeľné artefakty čiernej a bielej farby.

Zhrnutie

Výsledné siete troch implementovaných decimácii boli porovnané s pôvodnými. Okrem toho boli výsledky práce porovnané s existujúcimi nástrojmi — open-source programom `Meshlab` a komerčným Rapidform-om. Nasledujúca kapitola predstaví užívateľské rozhranie programu, aby čitateľ mohol výsledky replikovať.

5. Užívateľská dokumentácia

V tejto kapitole bude predstavený program z užívateľskej perspektívy. Najprv budú uvedené požiadavky na správne fungovanie programu. Ďalej bude uvedený návod na inštaláciu a predstavené bude aj užívateľské rozhranie. Na záver budú diskutované parametre, ktoré môže užívateľ nastaviť v aplikácii.

5.1 Systémové požiadavky

Aplikácia `MeshDecimation` je určená pre operačné systémy Windows. Preto na správne fungovanie je potrebný Windows 7 SP1 a novší. Ďalšou nevyhnutným softvérom je .NET Framework verzie 4.6.1. Inštalácia je dostupná na adrese `https://www.microsoft.com/en-us/download/details.aspx?id=49981`.

Po hardvérovej stránke je odporúčané minimum operačná pamäť 4 GB.

5.2 Inštalácia

Inštalácia programu `MeshDecimation` sa začne spustením súboru `setup.exe` z inštalačnej zložky. Užívateľ je potom vyzvaný na potvrdenie inštalácie. V okne je uvedený názov aplikácie (*Name: MeshDecimation*). Je potrebné vybrať možnosť `Install`. Následne sa inštalácia dokončí a priamo sa spustí nainštalovaná aplikácia `MeshDecimation`. Pri inštalácii bude pridaný odkaz na program do zoznamu aplikácii v ponuke `Štart`, odkiaľ je možné program kedykoľvek spustiť.

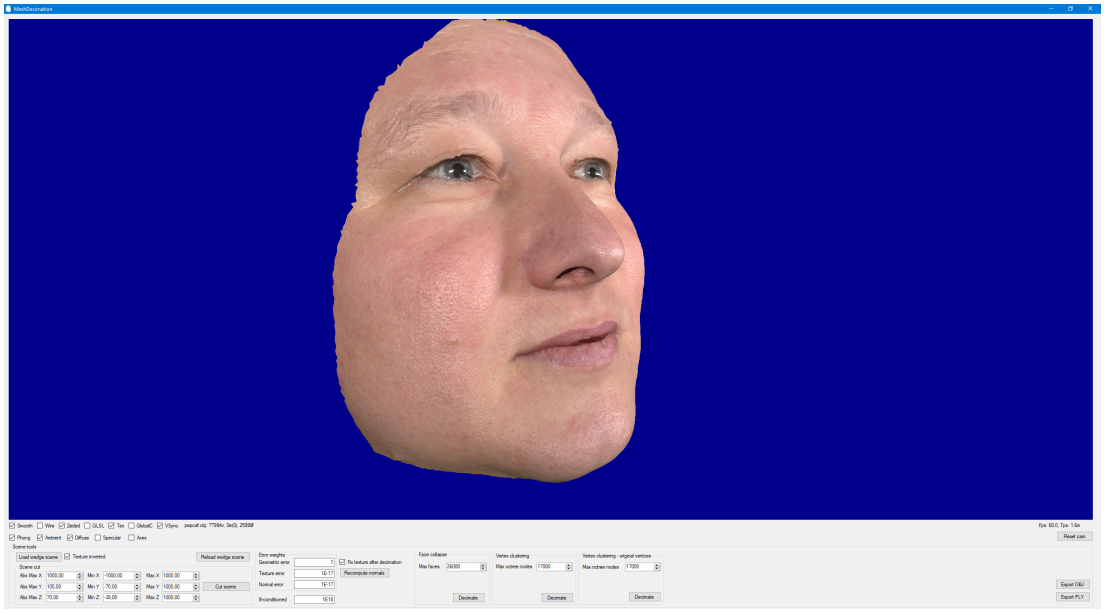
5.3 Užívateľské rozhranie a prehľad funkcii

Po štarte aplikácie `MeshDecimation` je užívateľovi k dispozícii grafické rozhranie ako je na obrázku 5.1. Najväčšiu časť okna zaberá zobrazenie scény, ktoré je po štarte prázdne. V spodnej časti okna sú k dispozícii ovládacie prvky.

5.3.1 Ovládanie zobrazenia

Upravovať vlastnosti zobrazovania scény je možné v spodnej časti okna. K dispozícii sú nasledujúce funkcie:

- **Smooth**: nastavenie vlastnosti OpenGL `ShadeModel` na hodnoty `ShadingModel.Smooth`, resp. `ShadingModel.Flat`
- **Wire**: pri zapnutí zobrazí iba obrysy trojuholníkov (nezobrazí steny)
- **2sided**: steny majú prednú aj zadnú stranu
- **GLSL**: zapne tieňovanie
- **GlobalC**: použije globálnu farbu (textúra musí byť vypnutá)
- **VSync**: zapne vertikálnu synchronizáciu



Obr. 5.1: Užívateľské rozhranie.

- **Phong**: zapne tieňovanie
- **Ambient**: zapne okolité svetlo
- **Diffuse**: zapne difúzne svetlo
- **Specular**: zapne odrazivosť
- **Axes**: zobrazí osi scény

V pravej časti sú informácie o frekvencii vykresľovania scény a tlačidlo na obnovenie polohy kamery (tlačidlo **Reset cam**).

5.3.2 Práca so scénou

Scénu je možné načítať zo súboru Wavefront OBJ (.obj). Toto je možné pomocou tlačidla **Load wedge scene**. Po stlačení sa zobrazí okno s výberom súboru scény. V prípade, ak je textúra scény obrátená, je potreba pre správne načítanie označiť možnosť **Texture inverted** vedľa tlačidla načítania.

Export scény je možný do súborov Wavefront OBJ a Stanford PLY. Na tieto účely slúžia tlačidlá **Export OBJ** a **Export PLY**. Potom sa užívateľovi zobrazí okno s výberom umiestnenia exportovaného súboru. V prípade exportu do formátu Wavefront OBJ sa odkáže exportovaná scéna na `mtl` súbor pôvodnej scény, ktorý popisuje materiály scény a odkazuje na textúru.

Ďalším významným nástrojom je **Scene cut**. Tváre produkované skenerom sú neorezané, okraje sú veľmi nepravidelné a sú nežiadúce. Program ponúka preto jednoduchý nástroj na orezanie. V kolónkach **Abs Max X/Y/Z** je zadávaná maximálna absolútna súradnica scény, v prípade ak je potreba aby scéna bola asymetricky orezaná, sú k dispozícii **Min X/Y/Z** a **Max X/Y/Z**. Po nastavení hodnôt sa stlačením tlačidla **Cut scene** odstránia časti scény mimo uvedeného

kvádra.

Originálnu scénu (načítanú zo súboru) je možné kedykoľvek rýchlo načítať pomocou tlačidla **Reaload wedge scene**. Jej kópia je udržiavaná v pamäti.

5.3.3 Decimácie

K dispozícii sú tri implementované decimácie. Ovládanie je užívateľovi dostupné zo spodnej časti obrazovky. Okrem nastavenia veľkosti výstupu z decimácie je dôležité pre dobré výsledky správne nastaviť hodnoty parametrov.

Hodnoty parametrov

Pred vykonaním decimácii je vhodné správne nastaviť parametre. Nastaviteľné sú tri parametre určujúce váhy jednotlivých zložiek (geometria, textúra a normály) a parameter určujúci, kedy sa kvadratická chybová funkcia `QEF` považuje za zle podmienenú.

Pre decimačné algoritmy využívajúce `QEF` sú odporučené parametre v tabuľke 5.1 (teda pre **Face collapse** a **Vertex clustering** decimáciu). Pre **Vertex clustering - original vertices** sú odporúčané koeficienty väčšej hodnoty. Tieto hodnoty však závisia od konkrétnych požiadavok užívateľa na výstup.

Zložka	**Odporúčaná hodnota**
Geometric error	1
Texture error	1^{-17}
Normal error	1^{-17}

Tabuľka 5.1: Odporúčané hodnoty váhových koeficientov pre decimácie využívajúce `QEF`.

Dôvodom veľmi nízkych hodnôt geometrickej a normálovej zložky je, že nadrovina $P' \subset \mathbb{R}^{3+m}$ (spoločný priestor geometrie a atribútov) bude viac rovnobežná s rovinou $P \subset \mathbb{R}^3$ (priestor geometrie) [7]. Pri nastavení váh atribútov na hodnoty blížiace sa nule je potom matica reprezentujúca `QEF` numericky stabilnejšia a vydáva lepšie výsledky.

Hodnota vlastnosti **Ill-conditioned** určuje hodnotu od kedy je `QEF` považovaná za zle podmienenú. Vtedy decimačné algoritmy nevyužijú `QEF` na minimalizáciu chyby. Namiesto toho použijú priemerné hodnoty spájaných vrcholov. Ak by sa algoritmy pokúsili o využitie zle podmienenej `QEF` na minimálizáciu, mohli by vzniknúť v decimovanej scéne chyby. Odporúčaná hodnota je od 10^{10} a viac.

Oprava textúr

Pri decimovaní siete môžu nastať chyby textúry na miestach nespojitosti (v prípade skenov tvári je to napríklad oblasť nosa, očí, úst a čela). Vybraním možnosti **Fix texture after decimation** sa po decimácii vykoná oprava chýb v textúre.

Výpočet normál

Výsledné siete nemusia mať dobre nastavené normály v niektorých vrcholoch, čo môže mať negatívne následky na tieňovanie (zapnuté **GLSL**). Ak užívateľ požaduje opravu normál, je k dispozícii tlačidlo **Recompute normals**. Normály sa vypočítajú v každom vrchole na základe susedných trojuholníkov (ako súčet normál a následne normalizovaný). Výsledkom je jemné tieňovanie.

Face collapse decimácia

Jeden z implementovaných algoritmov je decimácia využívajúca face collapse operátor a rozšírenú metriku `QEF`. Navyše okrem rozšírenej metriky zachováva nespojitosť atribútov vo vrchole. Pre to je potrebná práve *wedge scene*, ktorá reprezentuje rôzne hodnoty atribútov vo vrchole pomocou štruktúry *wegde* [13]. Regulácia výsledného počtu trojuholníkov je možná v kolónke **Max faces**. Regulácia je presná na 4 trojuholníky (jedna operácia môže odstrániť až 4 trojuholníky).

Vertex clustering

Ďalšia implementovaná decimácia využíva vertex clustering a metrika je opäť rozšírená `QEF`. Na rozdiel od predchádzajúceho algoritmu však nezachováva nespojitosť atribútov vo vrchole. Regulácia veľkosti výstupu je menej presná. Nastavuje sa parameter **Max octree nodes**. Veľkosť výstupnej siete (počet trojuholníkov) je však pre každú sieť individuálny. Približne 17 000 uzlov stromu octree odpovedá 26 000 trojuholníkom výslednej siete.

Vertex clustering - original vertices

Posledná implementovaná decimácia používa vertex clustering, metriku typu vzdialenosť vrchol-stena a navyše výsledná sieť je postavená z tých istých vrcholov ako pôvodná, tj. modifikuje iba konektivitu. Táto vlastnosť môže byť zaujímavá, ak je potrebné nemodifikovať *vertex buffer*, resp. zoznam vrcholov ako taký.

Zhrnutie

Kapitola previedla čitateľa od inštalácie až po nastavenie parametrov potrebných na dosiahnutie požadovaných a kvalitných výsledkov pri decimácii sietí. Nasledujúca, posledná kapitola prevedie čitateľa podrobnosťami týkajúcimi sa implementácie.

6. Programátorská dokumentácia

Táto kapitola sa venuje programátorskej dokumentácii. V úvode je predstavený základný projekt, z ktorého vychádza implementačná časť tejto práce. Ďalej budú predstavené využité knižnice a aj formáty ukladania meshí. Na koniec bude uvedená dokumentácia k vlastnej implementácii.

6.1 Základný projekt

Ako základ tejto práce poslúžil projekt 086shader poskytnutý RNDr. Josefom Pelikánom. Projekt sa využíva pri výučbe. Program vykresľuje pomocou knižnice OpenTK 3D scénu. Grafické rozhranie umožňuje prispôsobenie vykreslenia.

Z pohľadu tejto práce sú dôležité triedy z pôvodného projektu: trieda Form1, SceneBrep a StanfordPly. Form1 obsahuje obsluhu udalostí z grafického rozhrania. Počas práce bola teda táto trieda upravená tak, aby bolo možné jednoducho obsluhovať načítanie, decimáciu a export meshí. Trieda ScneneBrep reprezentuje 3D scénu. Obsahuje zoznam vrcholov, textúr, normál a trojuholníkov, ktoré dokáže dodať knižnici OpentTK na vykreslenie. Nakoniec, trieda StanfordPly umožňuje export scény reprezentovanej instanciou SceneBrep do súboru vo formáte PLY.

6.2 Využité knižnice

Pri implementácii boli využité dve knižnice. Prvou knižnicou je OpenTK [16]. OpenTK je knižnica umožňujúca využitie OpenGL a OpenAL priamo zo C# kódu. Pôvodný projekt ju integruje do užívateľského rozhrania a slúži na vykresľovanie scény.

Ďalšou využitou knižnicou je Math.NET Numerics [17]. Knižnica obsahuje špeciálne funkcie, pravdepodobnostné modely, náhodné čísla, interpoláciu, regresiu a metódy na optimalizačné problémy. Z pohľadu tejto práce je dôležitá časť lineárnej algebry. Implementované chybové kvadriky majú maticovú reprezentáciu. Okrem manipulácie s maticami je aj nevyhnutné minimalizovať sústavy rovností (minimalizácia kvadrík) a takisto je využitá lineárna algebra aj pri oprave chýb v textúre.

6.3 Formát pre ukladanie meshí

V implementácii práce sú využité dva formáty ukladania 3D scény: Wavefront OBJ [18, str. 946] (import a export) a Stanford PLY [19] (export).

Wavefront OBJ

Wavefront OBJ [18, str. 946] je formát na ukladanie 3D scén. Umožňuje ukladať čiary, mnohouholníky, krivky a povrchy. Reprezentácia je buď v ASCII (znakovo) alebo binárne (prípona súboru .MOD). V tejto práci je implementovaná podpora pre textový formát.

Komentáre v súbore sú na riadku začínajúcim #, riadky je možné logicky

spojiť znakom \ na konci riadku a inak sú definované nasledujúce kľúčové slová: **v** pre geometrickú polohu, **vt** pre textúrovú súradnicu, **vn** pre normálovú súradnicu a **f** pre face. Na jeden riadok sa píše jedno kľúčové slovo, za ním nasledujú argumenty oddelené medzerou (napr. **v**␣1.0␣ − 2.0␣3.0 je bod $(1, -2,3)$). Obvykle sú uložené v súbore trojuholníkové siete. V súbore sú najskôr popísané vrcholy, potom voliteľne textúry a/alebo normály a nakoniec trojuholníky. Trojuholník (alebo mnohouholník) sa zapíše pomocou príkazu **f** za ktorým nasleduje zoznam indexov vrcholov, ktoré ho tvoria. Ak sú definované aj textúry a/alebo normály, každý bod v zozname mnohouholníka sa zapíše v tvare vrchol/textúra/normála. Materiály (textúry) sú definované v súbore mtl. Ten sa definuje príkazom **mtllib**, za ktorým nasleduje názov mtl súboru. Súčasná implementácia umožňuje načítanie textúry zo súboru mtl.

Stanford PLY

Stanford PLY [19] formát je využitý len na export dát. Opäť má textovú a binárnu formu. Pre jednoduchosť a dobrú dokumentáciu je využitá textová (ASCII) forma. Pretože export do súboru bol implementovaný už v základnom projekte a ďalej nebol upravovaný v rámci tejto práce, nebude ďalej tento formát rozpísaný.

6.4 Prehľad tried

Táto sekcia je venovaná prehľadu implementovaných tried. Všetok kód, ktorý bol do pôvodného projektu pridaný (okrem drobných zmien v niektorých triedach) je uložený v `namespace MeshDecimation`. Upravené triedy pôvodného projektu majú charakter opráv chýb a pridania funkcii obsluhujúcich užívateľský vstup.

Poznámka k implementácii v jazyku C#

Jazyk C# je pomerne známy člen rodiny „céčkových“ jazykov. Najnovšia verzia jazyka C# však môže obsahovať niekoľko nových, menej známych, syntaktických štruktúr.

Vlastnosti (preperties) triedy

Jazyk ponúka koncept vlastností (properties) [20]. Vlastnosť je člen triedy, ktorý umožňuje čítať, zapisovať prípadne vypočítať hodnoty privátnych polí. Jednoduchá vlastnosť môže vyzerať napríklad ako `public int Cislo { get; set; }`. Tento kód teda znamená, že trieda obsahuje vlastnosť `Cislo`, ktorá je verejná a pretože `get` ani `set` nemajú pred sebou modifikátor `private`, je možné vlastnosť čítať aj zapisovať.

Novinkou v C# 6 sú vlastnosti len na čítanie zapísané pomocou novej syntaxe:

```
private int cislo;
public int Cislo => cislo;
```

V prípade, že sa použije operátor `=>`, neuvádza sa výraz `return` ani `get`. Vo verzii 7.0 bola umožnená nasledujúca konštrukcia:

```
private int cislo;
public int Cislo
{
        get => cislo;
        set => cislo = value;
}
```

Interpolated string

Novinkou od verzie 6 je tzv. „interpolated string“ [21]. Jedná sa o skrátený zápis formátovaného reťazca `String.Format(...)`. Po novom je teda možný zápis `string foo = $"dnes je {DateTime.Today.DayOfWeek}";`. Teda pred reťazec sa umiestni symbol $ a do reťazca je potom možné vložiť výraz alebo volanie funkcie do zložených zátvoriek {}, ktorý sa vyhodnotí.

ValueTuple

ValueTuple [22] bol pridaný v C# 7.0. Častokrát je potrebné predávať niekoľko rôznych hodnôt naraz. Riešením je napríklad definovanie novej štruktúry alebo triedy. To však zaberie ďalší (programátorský) čas a je to pomerne ťažkopádne. ValueType umožňuje jednoduchú syntax na zápis(napr. dvojica čísel `(int, int)`). Dokonca je možné si položky pomenovať: `(int cislo1, int cislo2)`. Prístup k položkám potom prebieha rovnako, ako by sa jednalo o štruktúru: `mojTuple.Item1`, resp. `mojTuple.cislo1`.

Kľúčové slovo yield

Zaujímavosťou môže byť riešenie niektorých pomocných funkcii v triede WedgeScene využívajúcich kľúčové slovo `yield` [23]. Takáto konštrukcia zjednodušuje kód (nie je potrebná trieda, ktorá by udržiavala stav zoznamu) v prípade, že je potrebná implementácia vlastnej kolekcie. Napríklad kód:

```
public IEnumerable<Vector3> FaceVertices(int face)
{
        yield return GetFaceVertex(face, 0);
        yield return GetFaceVertex(face, 1);
        yield return GetFaceVertex(face, 2);
}
```

vráti instanciu zoznamu `IEnumerable<Vector3>` naplnenú troma vrcholmi trojuholníka č. `face`.

Namespace MeshDecimation

`Namespace MeshDecimation` obsahuje najpodstatnejší zdrojový kód práce. V prípade potreby by sa dal oddeliť do knižnice, ktorá by sa dala využívať v iných programoch. `Namespace MeshDecimation` obsahuje ďalšie tri menné priestory (anglicky `namespace`): `WedgeScene`, `VertexClustering` a `FaceCollapse`. Triedy

nachádzajúce sa v mennom priestore `MeshDecimation`: abstraktná trieda `MeshDecimation`, `MinHeap` a `QEF`.

Trieda MeshDecimation

Abstraktná trieda `MeshDecimation` definuje rozhranie decimácie. Udržuje originál decimovanej scény a jej pracovnú kópiu. Okrem scén si pamätá aj váhové koeficienty pre geometriu, textúry a normály.

Trieda MinHeap

Trieda `MinHeap` je implementácia minimovej haldy prebraná z `https://raw.githubusercontent.com/JetStream96/MinMaxHeap/master/src/MinMaxHeap/MinHeap.cs`. Najpodstatnejšia je možnosť dodania vlastného komparátoru. V práci je halda využitá na ukladanie indexov. Komparátor dodaný minimovej halde pristupuje do polí s dátami, na základe ktorých je porovnanie vykonané.

Trieda QEF

Trieda `QEF` je implementáciou chybovej metriky popísanej v sekcii 2.3.2. Konkrétne je implementovaná forma z formule 2.12. Na reprezentáciu matice A je využitá trieda knižnice Math.NET `Matrix<double>`, vektory g_i a b sú uložené v `Vector<double>`, konšanta c je v premennej typu `double`. Okrem toho je uložená aj diagonála matice a váhové koeficienty.

Funkcie `IsIllConditioned()` a `ConditionNumber()` sú dôležité pre decimačné algoritmy. Matica A môže byť zle podmienená, čo môže mať za následky chyby v decimovanej meshi. QEF je považovaná za zle podmienenú, ak `ConditionNumber()` matice A je väčšie ako 10^8 (vyšlo z pozorovania na skenoch tvári).

Výpočet matice A prebieha podľa popisu v druhej kapitole. Dôležitý poznatok je, že kvadrika nie je okamžite po inicializácii pripravená na výpočet chyby, resp. na minimalizáciu. Pred tým treba ešte volať funkciu `ConstructQEFFromWedgeQEFs(params QEF[] wedgeQEFs)`, ktorá v prípade, že sa jedná o využitie rozšírenej kvadriky podľa [7, odsek č. 5], postaví odpovedajúcu maticu s rozmerom $3+km$, kde $m = 3$ (počet atribútov) a k je počet spájaných wedge-í. Ak je funkcii dodaná len jedna kvadrika, postaví sa obyčajná QEF s rozmerom $3 + m$.

Namespace WedgeScene

Menný priestor `WedgeScene` obsahuje triedy slúžiace na reprezentáciu a manipuláciu spojenú s scénou umožňujúcou zachovanie nespojitostí atribútov. Trieda reprezentujúca túto scénu sa rovnako volá `WedgeScene`. Menný priestor obsahuje ešte triedy `WedgeSceneTextureFixer`, triedu na opravu textúry decimovanej siete, a `WedgeSceneTools` zoskupujúca niekoľko užitočných nástrojov.

Trieda WedgeScene.WedgeScene

Trieda `WedgeScene.WedgeScene` reprezentuje 3D scénu. Okrem geometrie udržuje aj textúry a normály. Scéna je rozložená na vrcholy, normály, textúry, wedge-e a trojuholníky. Trojuholníky sú udržiavané v poli typu `int[]`, i-ty trojuholník je uložený na pozíciach $3i$ až $3i+2$. Vrchol trojuholníka je index v zozname wedge-í.

Wedge-e sú tiež uložené v poli. Každá wedge nesie informáciu o vrchole, normále a textúre. Opäť sa jedná o indexy do poľa vrcholov, normál a textúr. Existuje aj spätné mapovanie, teda vrchol → wedge-e a wedge → trojuholníky. Tieto mapovania sú uložené v slovníku `int` → `List<int>`.

Trieda má funkciu `LoadFromObj(string fileName)` umožňujúcu načítanie a funkciu `SaveToObj(string file)` na export 3D scény z/do súboru Wavefront OBJ. Funkcia `ToScene(Scene3D.SceneBrep scene)` umožňuje uloženie scény do triedy `Scene3D.SceneBrep`, ktorú pôvodný projekt `086shader` dokáže zobraziť užívateľovi.

Trieda WedgeScene.WedgeSceneTextureFixer

Trieda `WedgeScene.WedgeSceneTextureFixer` implementuje opravu textúry predstavenú v 3.3. Algoritmus prejde všetky trojuholníky z novej zdecimovanej scény, zistí, či je potrebná oprava textúr vrcholov daného trojuholníka na základe postupu zo state Detekcia vrcholov na opravu a prípadne vykoná opravu. Vo funkcii `ComputeNewTextures(int face)`, ktorá zabezpečuje opravu konkrétneho trojuholníka je zaujímavé, že bola využitá schopnosť knižnice `Math.NET`, vyriešiť sústavu $AX = B$, kde B je matica s vhodnými rozmermi. To prinieslo výhodu, že rovnica 3.7 nemusí byť riešené pre každé i zvlášť. Matica B rovnosti je zostavená zo stĺpcových vektorov, kde i-ty stĺpec odpovedá pravej strane rovnice 3.7 pre i-ty vrchol trojuholníka.

Trieda WedgeScene.WedgeSceneTools

Trieda `WedgeScene.WedgeSceneTools` zoskupuje niekoľké nástroje manipulujúce s 3D scénou uloženou v `WedgeScene.WedgeScene`. Konkrétne sa jedná o nástroj na odstraňovanie trojuholníkov, orezávanie scény a výpočet normál scény.

Úplné odstránenie trojuholníkov zo scény zabezpečuje funkcia `RemoveFaces (scene, facesToRemove)`. Tá okrem odstránenia trojuholníkov odstráni aj nepoužívané vrcholy, textúry, normály a wedge-e a obnoví zoznamy mapujúce vrchol → wedge-e a wedge → trojuholníky.

Funkcia `WedgeSceneCut(...)` odstráni všetky trojuholníky mimo zadaných hraníc. Scénu potom vyčistí pomocou vyššie uvedenej funkcie.

Nakoniec, funkcia `ComputeNormals(WedgeScene scene)` vypočíta nové normály pre scénu. Normály sú normalizované na jednotkovú veľkosť. Vypočítaním nových normál sa zabezpečí, že scéna bude pekne vytieňovaná pri zobrazení.

Namespace VertexClusering

Menný priestor `VertexClusering` zoskupuje decimácie využívajúce dátovú štruktúru octree. Dátová štruktúra octree je implementovaná v triede `Adaptive-Octree`. Prototyp a spoločné funkcie pre decimačné algoritmy fungujúce na octree DS sú v abstraktnej triede `VertexClustering`. Potomkovia tejto triedy, `Vertex-ClusteringGeometricQEF` a `VertexClusteringQEF`, sú konkrétne implementácie decimácii.

Trieda VertexClusering.AdaptiveOctree

Trieda `VertexClusering.AdaptiveOctree` implementuje dátovú štruktúru octree. Okrem reprezentácie obsahuje aj funkcie pracujúce s touto DS — funkcie na stavbu a vkladanie nových položiek do stromu a lazy decimačný algoritmus.

Pretože pri niektorých operáciach (ako je napríklad vloženie vrcholu do uzlu) je potrebné vykonať špecifické operácie (závisiace na konkrétnom decimačnom algoritme), trieda obsahuje niekoľko odkazov na metódy, resp. funkcie, ktoré musí konkrétna decimácia implementovať a doplniť. Odkazy na funkcie sú vyriešené pomocou tried `Action<...>` a `Func<...>`. Ich naplnenie zabezpečuje abstraktná trieda `VertexClustering.VertexClustering`, z ktorej sú odvodené konkrétne realizácie decimačných algoritmov. V prípade potreby je možné teda implementovať aj inú realizáciu octree decimácie.

Trieda si ukladá len nevyhnutné informácie o strome, ktoré sú potrebné na stavbu a udržiavanie. Práve vyššie spomenuté funkcie na doplnenie umožňujú konkrétnym implementáciam decimačných algoritmov pridanie vlastných potrebných dát. Každý uzol stromu si pamätá rodiča, potomkov, chybu a príznak určujúci, či sa jedná o vnútorný vrchol. Celý strom je potom reprezentovaný ako zoznamy (`List<...>`) uvedených vlastností uzlov, maximum z absolútnej hodnoty koordinátov a roh scény (vrchol $(-max, -max, -max)$). Polohu a veľkosť jednotlivých uzlov sa počíta pri prechádzaní stromu (pretože každý uzol $\sim$ kocka sa delí na rovnakých osem potomkov).

Trieda VertexClusering.VertexClusering

Abstraktná trieda `VertexClusering` definuje rozhranie decimácii založenej na octree dátovej štruktúre. Má definované abstraktné funkcie, ktoré vyžaduje Trieda VertexClusering.AdaptiveOctree a je ich potrebné v konkrétnych decimáciach správne implementovať. Okrem funkcii pre octree DS definuje aj niekoľko iných metód, ktoré sú pre tieto decimačné algoritmy spoločné (napríklad rekonštrukcia meshe).

Trieda VertexClusering.VertexClusteringGeometricQEF

Trieda `VertexClusteringGeometricQEF` je implementáciou decimácie založenej na octree DS a je odvodená od abstraktnej triedy `VertexClusering`. Do octree pridáva vlastnosť zoznam vrcholov v uzle (vlastnosť `NodeVertices`). Na QEF využíva kvadriku kvantifikujúcu vzdialenosť vrchol-plocha. Pre potreby kvadriky si teda ešte pamätá pre každý vrchol aj normálový vektor trojuholníka, s ktorým susedí (ak je susedných trojuholníkov viac, pridá sa na vstup dvojica vrchol-normála viackrát). Identifikačné čísla vrcholov pridávané do octree sú indexy do zoznamu `VertexFaces`, ktorý obsahuje dvojicu index do vrcholov scény a index do zoznamu normál trojuholníkov (`FaceNormals`).

Trieda VertexClusering.VertexClusteringQEF

Trida `VertexClusteringQEF` implementuje decimáciu založenú na octree DS s metrikou QEF. Pre každý uzol octree pridáva QEF. Túto vlastnosť reprezentuje zoznamom `NodeQEFs`, kde index uzlu stromu odpovedá indexu QEF daného uzlu. Na vstup sa pridávajú dvojice vrchol-QEF trojuholníka susediaceho s vrcholom,

niektoré vrcholy sú teda v zozname viackrát. Asociácia vrchol-QEF je zabezpečená zoznamami `VertexIDs` a `VertexQEFs`, kde hodnota v VertexIDs odpovedá indexu do zoznamu vrcholov scény.

Trieda FaceCollapse.FaceCollapseDecimation

Trieda `FaceCollapseDecimation` implementuje decimáciu pomocou face collapse operátoru. Na meranie chyby využíva rozšírenú metriku QEF. Umožňuje lepšie zachovávať nespojitosť atribútov — vrchol je rozdelený do wedge-í a pre každú sú zvlášť vypočítavané atribúty.

Radenie trojuholníkov zabezpečuje minimová halda `Queue`, ktorá obsahuje indexy trojuholníkov. Halda dostáva pri svojej inicializácii vlastný komparátor, ktorý zoradí indexy i a j podľa hodnoty QEF odpovedajúcich trojuholníkov. Hodnoty QEF pre trojuholníky sú uložené vo `FaceErrors`. Pretože je využitý lazy greedy radiaci algoritmus, je potrebné si pamätať trojuholníky, ktoré boli v susedstve pozmenených trojuholníkov. Na tento účel slúži zoznam `DirtyFaces`. Na pamätanie si QEF pre jednotlivé wedge-e slúži zoznam `WedgeQEFs`. Index wedge-e v scéne odpovedá indexu QEF v tomto zozname. Pretože sa trojuholníky neodstraňujú priamo zo scény okamžite, udržuje sa aj zoznam odstránených trojuholníkov, ktorý je po decimácii dodaný nástroju na odstraňovanie trojuholníkov zo siete.

Zhrnutie

Posledná kapitola predstavila základný projekt, z ktorého bola implementačná časť práce ďalej stavaná a uvedené boli aj využité knižnice. Spomenuté boli dva formáty ukladania 3D scény — formát Wavefront OBJ a Stanford PLY. Pretože sa jazyk práce C# neustále vyvíja a rozširuje o zaujímavé syntaktické štruktúry, niekoľké z nich boli spomenuté. Zvyšok kapitoly bol venovaný prehľadu menných priestorov a tried programu.

Záver

Technológie sa dnes vyvíjajú míľovými krokmi. Tento pokrok sa nevyhol ani 3D skenerom, kde najnovšie modely dokážu zaznamenať aj ten najjemnejší detail. Schopnosť 3D skenerov detailne zaznamenávať reálne objekty však prekračuje schopnosti hardvéru zobrazovať a spracovávať takéto scény. Tu na rad prichádzajú techniky z disciplíny *level of detail* (LOD).

Cieľom práce bolo implementovať a porovnať niekoľko decimačných algoritmov trojuholníkových sietí. Dôraz bol kladený na zachovanie geometrie sietí a ich atribútov (normálových vektorov a najmä textúr). V implementačnej časti práce boli vyskúšané tri decimácie. Ako najlepšia sa ukázala byť decimácia s využitím operátora face collapse, metrikou QEF a dátovou štruktúrou wedge. Najlepšie dokázala zachovať geometriu a aj jej vizuálny výstup bol kvalitný. Druhý prístup, vertex clustering operátor s rozšírenou kvadrikou QEF dokázal tiež veľmi kvalitne redukovať počet vrcholov. Posledný prístup, vertex clustering zachovávajúci pôvodné vrcholy dosiahol mierne horšie výsledky v porovnaní s ostatnými decimáciami, avšak tiež dosiahol uspokojivé výsledky pri pomerne krátkom behu decimácie (typicky okolo 5 sekúnd pre decimáciu skenu tváre z približne 400 000 na 26 000 trojuholníkov).

V ďalšej časti práce bol problém decimácie trojuholníkových sieti bližšie popísaný. Práca sa venuje algoritmickému prístupu k problému, rieši vonkajšiu optimalizáciu (radenie operácii) a vnútornú optimalizáciu (aké operácie použiť). Ďalej boli popísané chybové metriky, ktoré pomáhajú pri optimalizácii. Nakoniec boli tieto poznatky spojené do implementovaných decimácii.

Výsledkom je teda prehľad o niekoľkých prístupoch k decimáciam a ich samotná implementácia. K dispozícii je ucelený nástroj s užívateľským rozhraním, ktorý dokáže načítať a zobraziť 3D scénu, zdecimovať ju vybraným algoritmom a podľa zadaných parametrov a nakoniec exportovať výsledok. Vďaka oddeleniu kódu do jedného menného priestoru je v prípade potreby možné jednoducho zdrojový kód vyčleniť do samostatnej knižnice na použitie v inom softvéri. Príkladom môže byť `Morphome3cs` vyvíjaný Laboratóriom 3D zobrazovacích a analytických metód ma PřF UK so Skupinou počítačovej grafiky UK.

Implementované decimačné algoritmy boli testované na väčšom počte skenov tvárí, avšak z dôvodu ochrany osobných údajov nie sú tieto dáta ani výsledky súčasťou práce.

Zoznam použitej literatúry

[1] Mostafa Abdel. 3D LASER SCANNERS: HISTORY, APPLICATIONS, AND FUTURE. 2011.

[2] Marc Levoy. The Digital Michelangelo Project. `https://graphics.stanford.edu/papers/digmich-3dimaging99/`, 1999, navštívené 05.04. 2018.

[3] David P Luebke. *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2003.

[4] Pedro V Sander, John Snyder, Steven J Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM, 2001.

[5] Jonathan David Cohen. *Appearance-preserving Simplification of Polygonal Models*. PhD thesis, 1998. AAI9914826.

[6] The Stanford 3D Scanning Repository. `http://graphics.stanford.edu/data/3Dscanrep/`, 2014, navštívené 05.04. 2018.

[7] Hugues Hoppe. New quadric metric for simplifiying meshes with appearance attributes. In *Proceedings of the conference on visualization'99: Celebrating ten years*, pages 59–66. IEEE Computer Society Press, 1999.

[8] Nicolas Aspert, Diego Santa-Cruz, and Touradj Ebrahimi. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Multimedia and Expo, 2002. ICME'02. Proceedings. 2002 IEEE International Conference on*, volume 1, pages 705–708. IEEE, 2002.

[9] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216. ACM Press/Addison-Wesley Publishing Co., 1997.

[10] Michael Garland and Paul S Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *Visualization'98. Proceedings*, pages 263–269. IEEE, 1998.

[11] File: Octree2.svg. `https://commons.wikimedia.org/wiki/File:Octree2.svg`, 2010, navštívené 07.04. 2018.

[12] Scott Schaefer and Joe Warren. Adaptive vertex clustering using octrees. *SIAM geometric design and computing*, 2(6), 2003.

[13] Hugues Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 22(1):27–36, 1998.

[14] Inc. Direct Dimensions. 3d Software - Modeling Software - 3D CAD - 3D Program - 3D Modeling from Direct Dimensions, inc., 2018, navštívené 18.04. 2018.

[15] Inc. INUS Technology. *Rapidform XOR User Manual*.

[16] The Open Toolkit. https://opentk.github.io/, 2018, navštívené 09.04. 2018.

[17] Math.NET Numerics. https://numerics.mathdotnet.com/, 2018, navštívené 09.04. 2018.

[18] James D Murray and William vanRyper. *Encyclopedia of Graphics File Formats: The Complete Reference on CD-ROM with Links to Internet Resources*. O'Reilly, 1996.

[19] Paul Bourke. PLY - Polygon File Format. http://paulbourke.net/dataformats/ply/, 2017, navštívené 09.04. 2018.

[20] Properties (c# Programming Guide). https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/properties, 2017, navštívené 10.04. 2018.

[21] $ - string interpolation (c# Reference). https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated, 2017, navštívené 10.04. 2018.

[22] Mark Zhou. C# 7 Series, part 1: Value Tuples. https://blogs.msdn.microsoft.com/mazhou/2017/05/26/c-7-series-part-1-value-tuples/, 2017, navštívené 10.04. 2018.

[23] yield (c# Reference). https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/yield, 2015, navštívené 10.04. 2018.

Zoznam obrázkov

Zoznam tabuliek

A. Prílohy

A.1 Príloha č. 1

Príloha č. 1 obsahuje tento dokument, inštalačný program `MeshDecimation` a zdrojové súbory. Z dôvodu ochrany osobných údajov nie sú priložené žiadne skeny tvárí. V prípade záujmu sú tieto dáta k dispozícii na vyžiadanie u RNDr. Josefa Pelikána (mail: pepca@cgg.mff.cuni.cz).

Štruktúra:

- **/doc**: dokument práce,
- **/bin**: inštalácia programu `MeshDecimation`,
- **/src**: zdrojové súbory.