Evolucijski pristup višekriterijskoj optimizaciji topologije 3D modela

Čondrić, Dominik - Matej

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni diplomski studij računarstva

Diplomski rad

Evolucijski pristup višekriterijskoj optimizaciji topologije 3D modela

Rijeka, rujan 2023.

Dominik-Matej Čondrić
0069085608

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni diplomski studij računarstva

Diplomski rad

Evolucijski pristup višekriterijskoj optimizaciji topologije 3D modela

Mentor: doc.dr.sc. Goran Mauša

Rijeka, rujan 2023.

Dominik-Matej Čondrić
0069085608

Umjesto ove stranice umetnuti zadatak za završni ili diplomski rad

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2023.

_____
Dominik-Matej Čondrić

Zahvala

Zahvaljujem mentoru doc.dr.sc. Goranu Mauši na entuzijazmu prilikom predavanja i mentorstva u ovome radu. Također zahvaljujem svom nastavnom osoblju tijekom mojeg petogodišnjeg studiranja na ovom fakultetu zahvaljujući kojima danas posjedujem znanje koje mi otvara vrata budućim poslovima i prilikama.

Također se želim zahvaliti mojoj obitelji koja mi je bila najveća podrška i svima koji su me poticali na učenje i usavršavanje tijekom studiranja.

Sadržaj

Popis slika

Uvod

Kada se radi o računalnom iscrtavanju trodimenzionalnih scena, kvaliteta 3D modela predstavlja jedan od ključnih elementata koji određuje detaljnost scene. Kako bi scenu mogli prikazati na računalima ograničenih resursa, svaki model na sceni definiran je određenim brojim vrhova koji, povezani geometrijskim primitivima, čini mrežu točaka. Pritom je lako zaključiti da se povećanjem vrhova modela svaki objekt može bolje aproksimirati, a to je osobito slučaj kod izuzetno nepravilnih modela ili onih sa brojnim krivinama (kao npr. sfera). S druge strane, ograničenja današnjih računala postavljaju zahtjev za što manjim brojem vrhova 3D modela kako bi scenu bilo moguće iscrtavati dovoljno često da ljudsko oko ne primjeti prijelaze. Također u obzir treba uzeti i da zbog perspektive objekti udaljeniji od točke gledišta izgledaju manje od objekata bližih točki gledišta, te samim time razina detalja koja je potrebna ne mora biti jednaka. Upravo iz tog razloga u cilju je aproksimirati 3D model sa što manjim brojem vrhova uz dovoljno očuvanu razinu detalja.

S obzirom na povećanje procesne snage današnjih računala te njihovu dostupnost korisnicima, 3D modeli korišteni u *real-time* aplikacijama i programima nerijetko imaju vrlo velik broj vrhova, čak i preko milijun. Kako bi takav 3D model pojednostavili kroz vrijeme se nastale brojne tehnike pojednostavljivanja modela, a u ovom će radu fokus biti na tehnici uklanjanja vrhova.

S obzirom na prirodu navedenog problema i ogromnog prostora pretrage potencijalnih rješenja, tehnika uklanjanja vrhova predstavlja pristup vrlo pogodan za rješavanje stohastičkim metodama, u slučaju ovog rada, evolucijskim algoritmom. U nastavku je predstavljeno rješenje problema algoritmom NSGA-II, pri čemu je cilj optimizirati model po dvije funkcije dobrote - globalnoj pogrešci te broju vrhova

koji su međusobno proturječni. Uklanjanjem vrhova u modelu, globalna pogreška se povećava te je cilj ovoga rada minimizirati globalnu pogrešku uz minimizaciju broja vrhova.

Među brojnim pristupima pojednostavljivanju mreže, kroz godine su se izvojili određeni pristupi koji su se pokazali vrlo uspješnima uzevši u obzir kvalitetu rezultatne mreže i računsku učinkovitost. Poznati pristup koji je do dan danas među najkorištenijima jest pristup uklanjanjem bridova prema *Quadric error* metrici [8]. Algoritam se bazira na odabiru svih validnih bridova koji se mogu ukloniti, izračunu metrike nad njima, dodavanjem bridova u prioritetnu hrpu gdje su na vrhu oni sa najmanjom pogreškom, te uklanjanjem bridova od vrha hrpe prema dnu ovisno o željenoj aproksimaciji modela. Ključni dio je odabrati novu poziciju vrha u kojem se spajaju vrhovi brida koji se uklanja na način da je pogreška što manja. Iako je ovakav pristup standard u današnjem vremenu zbog efikasnosti i dobrih aproksimacija, nije najprecizniji i najbrži te ima problema s implicitnim određivanjem pogreške. Drugim riječima, algoritam je pohlepne prirode pa unaprijed računa pogrešku ne uzimajući u obzir kojim će se redom uklanjati bridovi, te samim time u pogrešku uklanjanja kasnijih bridova može ući i udaljenost nove točke od ploha trokuta koji više ne postoje jer su uklonjeni kroz prethodne korake algoritma.

Osim pristupa pohlepnim algoritmima kakav je i prethodno opisan, javili su se i stohastički pristupi slični onima korištenim u ovom radu. Jedan takav je korištenje algoritma *NSGA-II* sa uklanjanjem vrhova no sa nešto različitom metrikom procjene pogreške[9]. S obzirom da je prostor pretrage rješenja izuzetno velik, upravo genetski algoritam pruža izvrsnu alternativu pohlepnom pristupu opsežnom pretragom prostora te potencijalnim nalaženjem boljih rješenja.

Treba napomenti i da se s razvojem računala i napretkom u području strojnog učenja noviji pristupi baziraju na aproksimaciji neuronskim mrežama [10]. Takvi pristupi daju izvrsne rezultate čak i u stvarnom vremenu dok kvaliteta ostaje vrlo dobra te otvaraju vrata cijelom području istraživanja primjene neuronskih mreža u računalnoj grafici i 3D modeliranju.

Nadalje, prikazani su rezultati dobiveni na tri modela s različitim brojem vrhova te ušteda u performansama za iscrtavanje iste scene.

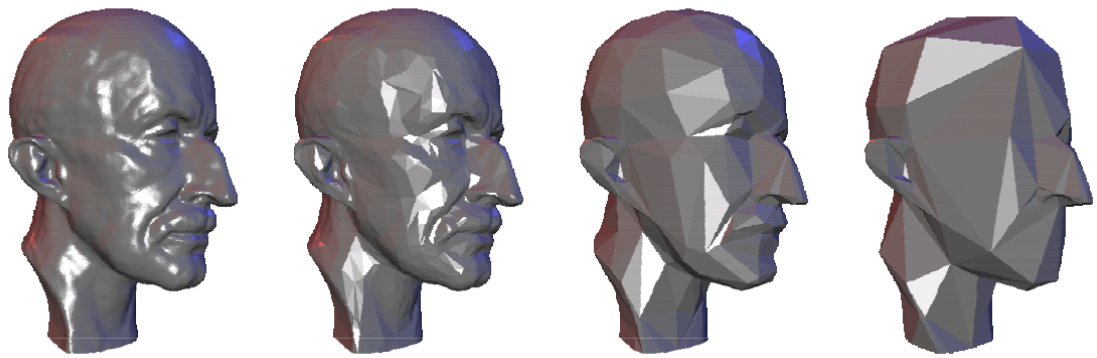
Poglavlje 1

Modeliranje

1.1 Opis 3D modela

Topologija svakog 3D modela opisana je vrhovima sa pripadajućim koordinatama u trodimenzionalnom prostoru. Vrhovi su potom povezani primitivima, uglavnom trokutima (zbog njihove planarnosti i jednostavnosti rasterizacije), koji čine površinu modela odnosno njenu površinsku mrežu (*eng. mesh*). Pri tome je bitno naglasiti da se 3D modeli dijele na pune kod kojih je cijeli volumen modela opisan vrhovima, te one kod kojih je samo površina modela opisana vrhovima, o kojima se ovaj rad i bavi.

Prilikom 3D modeliranja, cilj dizajnera je što preciznije opisati objekt koji se modelira, za što je potrebno i što više vrhova u prostoru. Na slici 1.1 može se uočiti razlika u detaljnosti modela sa različitim brojem vrhova, te razumjeti motivacija za što većim brojem vrhova modela. Iako je vizualno detaljniji model puno prezicniji i vjerniji predmetu kojeg se moderlira, sa povećanjem vrhova raste i količina operacija, a samim time je iscrtavanje procesno zahtjevnije. Upravo taj problem rješava LOD.



Slika 1.1 Prikaz 3D modela sa različitim brojem vrhova (preuzeto iz [1])

1.2 LOD

Iako je očito da je detaljniji model puno precizniji te samim time pogodniji za korištenje, u obzir treba uzeti da perspektivna projekcija dalje objekte čini manjima, a bliže većima te da detalji sa udaljenosti objekata postaju manje uočljiviji. Kako bi se dobilo na performansama programa, odnosno ubrzalo vrijeme potrebno za iscrtavanje scene, udaljeniji modeli se iscrtavaju u manjim rezolucijama budući da je razlika u većini slučajeva neprimjetna. Takva ideja modeliranja objekata u nekoliko razina detalja poznata je pod nazivom LOD, te predstavlja jednu od glavnih tehnika dobivanja na performansama prilikom iscrtavanja velikih scena.

Ideja LOD-a nastala je 1976. godine kada je James H. Clark obajvio članak u kojem je izjavio kako najplodonosniji napretci u smislu brzine izvođenja transformacija i obrezivanja nastaju strukturiranjem iscrtavane scene. Uzmemo li primjer sfere, na dovoljno velikoj udaljenosti iscrtavanje dodekahedrona biti će dovoljno slično sferi, a značajno će ubrzati performanse programa[4].

Postoje dvije glavne tehnike uporabe LOD-a u računalnoj grafici:

- *Discrete Levels of Detail* (DLOD)
- *Continuous Levels of Detail* (CLOD)

DLOD odnosi se na kreiranje nekoliko razina detaljnosti modela unaprijed te odabiru prigodnog modela prema udaljenosti modela od točke gledišta. Prednost ovakvog pristupa jesu bolje performanse s obzirom da se svi modeli učitavaju prije prvog iscrtavanja scene, no zbog diskretnih razina modela dolazi do naglih vizualnih promjena (eng. *visual popping*). CLOD s druge strane koristi strukturu koja sadrži varijabilni spektar detalja te se po potrebi bira razina primjerena udaljenosti objekta. Prednost ovakvog pristupa je mogućnost lokalne promjene razine detalja na jednom modelu, pa dijelovi modela bliže točki gledišta mogu imati veću razinu detalja od udaljenih dijelova modela. U ovom radu riječ je samo o DLOD pristupu budući da se CLOD bazira na drugačijim rješenjima od onog prikazanog u ovom radu.

1.3 Formati 3D modela

3D modeli pohranjuju se u jednom od brojnih formata, često nastalih kao posljedica razvoja nekog od programa za modeliranje. Tako se danas koriste formati *Standard Tessellation Language* (STL), *3D Studio* (3DS), *MAX*, *COLLAborative Design Activity* (COLLADA), *OBJ* itd. S obzirom da je za pohranu jednog modela obično potrebno mnogo prostora s obzirom na količinu informacija koju je potrebno pohraniti, različiti formati traže kompromis između veličine i čitljivosti odnosno lakoće parsiranja. Upravo zbog lakoće parsiranja u kontekstu ovog rada koristi se samo OBJ format i to s onim modelima koji sadrže samo jedan objekt.

1.3.1 *OBJ* format

OBJ format tekstovni je format koji služi za geometrijsku reprezentaciju 3D modela. Drugim riječima *OBJ* format za svaki vrh najčešće opisuje koordinate njegove pozicije u prostoru, koordinate teksture (uv koordinate), normalu te poligone koji povezuju vrhove. Dodatno se u *OBJ* formatu mogu pronaći informacije o krivuljama, grupiranje, dali se koristi glatko sjenčanje i slično. Kako bi se uspješno parsirao model iz *OBJ* formata (ekstenzije .obj), potrebno je poznavati strukturu dokumenta. Svaka linija počinje sa labelom koja označava o kojem tipu podatka se radi te je slijede sami podaci u obliku ovisnom o samoj labeli. U kontekstu ovog rada bitno je

znati sljedeće labele:

- `#` - komentar, sve iza labele se može ignorirati prilikom parsiranja
- `v` - koordinate vrha u prostoru, tri realna broja predstavljaju x, y i z koordinatu
- `vn` - koordinate normale vrha, tri realna broja predstavljaju vektor
- `vt` - uv koordinate teksture, dva realna broja u rasponu $u, v \in [0.0, 1.0]$
- `f` - poligon, u nastavku slijedi proizvoljan broj vrhova u obliku $i_v/i^*_{vn}/i^*_{vt}$ pri čemu indeksi označeni zvjezdicom označavaju opcionalne indekse (vrh mora imati koordinate u prostoru, ali ne i normalu i uv koordinate).

Poglavlje 2

Optimizacija trodimenzionalne mreže

Kao što je već spomenuto u poglavlju 1.2, ideja optimizacije trodimenzionale mreže nastala je s idejom LOD-a. Od tada pojavili su se brojni algoritmi optimizacije trodimenzionalne mreže uz razne metrike kvalitete rezultatnog modela. Sama optimizacija ima dva ključna koraka: tehniku pojednostavljivanja mreže te metriku koja određuje preciznost rezultantne mreže[5]. Tu treba napomenti kako nije praktično kombinirati svaku tehniku pojednostavljivanja sa svakom metrikom preciznosti rezultantnog modela s obzirom da po prirodi algoritama neke kombinacije nisu primjenjive. U nastavku rada opisane su poznate tehnike rješavanja problema optimizacije sa naglaskom na one korištene u ovom radu.

2.1 Tehnike pojednostavljivanja mreže

Algoritmi pojednostavljivanja mreže dijele se na iterativne koji mrežu mijenjaju lokalno, i direktne koji mrežu mijenjaju globalno. Kako je ideja ovoga rada primjena pojednostavljivanja u svrhu kreiranja LOD razina, u nastavku je riječ samo o iterativnim tehnikama.

Iterativni algoritmi primjenjuju jedan od način pojednostavljivanja mreže lokalno, na manjem dijelu mreže, te iterativno poboljšavaju rješenje dok se ne dostigne

određeni kriterij. Kriterij zaustavljanja mogu biti razni te mogu ovisiti o željenoj pogrešci, broju točaka u mreži, broju poligona u mreži i slično. Kada govorimo o iterativnim tehnikama pojednostavljivanja mreže, tri tehnike su najčešće spominjane i korištene u literaturi a to su:

- Uklanjanje vrhova
- Urušavanje bridova
- Urušavanje polubridova

Slika 2.1 prikazuje sve tri tehnike na minimalnom primjeru.

Kako bi rješenje nakon određenog broja iteracija bilo što preciznije, odabiru se oni dijelovi mreže koji rezultiraju najmanjim pogreškama te oni imaju prioritet pri pojednostavljivanju, no problem koji s time dolazi je računska zahtjevnost s obzirom da za svaku moguću modifikaciju treba izračunati rezultantnu pogrešku. Ideja ovoga rada je pokušati s drugačijim pristupom zaobići navedeni problem te stohastičkim načinom doći do suboptimalnih rješenja. Također treba napomenuti da se s obzirom na prirodu rješenja koje je predstavljeno u nastavku, ovaj se radi bavi isključivo tehnikom optimizacije uklanjanjem vrhova te ne ulazi u detalje ostalih tehnika.

2.1.1 Uklanjanje vrhova

Tehnika uklanjanja vrhova jedna je od prvih predloženih tehnika pojednostavljivanja mreža, te je prema nazivu lako zaključiti kako algoritam radi. Prvi korak algoritma sastoji se od odabira vrhova koje je potrebno ukloniti iz mreže, te uklanjanja svih trokuta koji su dijelili uklonjeni vrh. Kao što je vidljivo na slici 2.1a, uklanjanjem vrha i trokuta u mreži ostaje neispunjen poligon koji se sastoji od susjednih vrhova uklonjenog vrha te se s time ide u sljedeći korak algoritma. Da bi mreža ostala povezana, šupljinu u mreži koja nastaje uklanjanjem vrha potrebno je ponovno ispuniti trokutima (*retriangulirati*). Sama problematika triangulacije poligona je zaseban problem, te je u pododjeljku 2.3 dan uvod u problem, te predstavljeno korišteno rješenje. Zadnji korak algoritma sastoji od izračuna pogreške novonastale strukture, no više o tome u potpoglavlju 2.2.

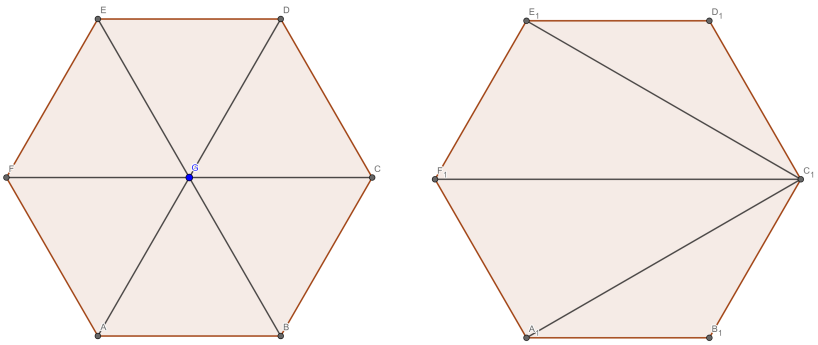
Najveća prednost ove metode u odnosu na druge jest zadržavanje originalne topologije mreže s obzirom da se ne unose dodatni vrhovi već se samo mijenja način na koji se postojeći povezuju, što može biti vrlo korisno kao npr. kod metode konačnih elemenata[5].

2.1.2 Urušavanje bridova

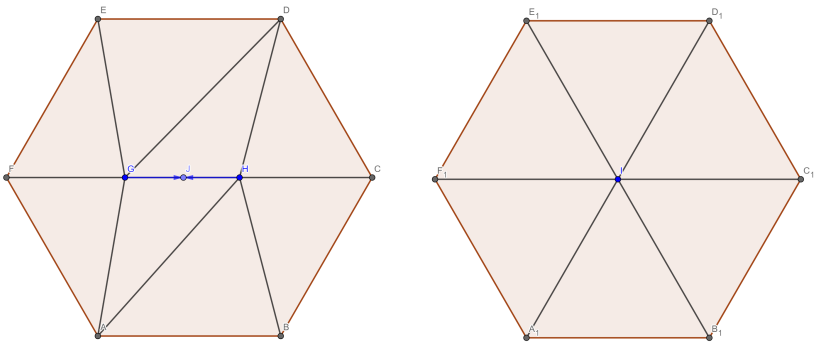
Metoda urušavanja bridova najpopularnija je od iterativnih tehnika te se svodi na zamjenu brida sa vrhom. Lako je iz toga zaključiti da svaka operacija uklanjanja vrha smanjuje broj vrhova u modelu za jedan, te da ova operacija za razliku od uklanjanja vrhova mijenja topologiju 3D mreže s obzirom da se unosi novi vrhovi u mrežu. Prilikom ponovne triangulacije, svi trokuti koji su sadržavali uklonjeni vrh mijenjanju taj vrh sa novonastalim dok se trokuti koji su dijelili uklonjeni brid uklanjanju. Bitan faktor kvalitete ovog pristupa je optimalna pozicija novog vrha koja mijenja uklonjeni brid, te se ona određuje nekom od metrika preciznosti mreže.

2.1.3 Urušavanje polubridova

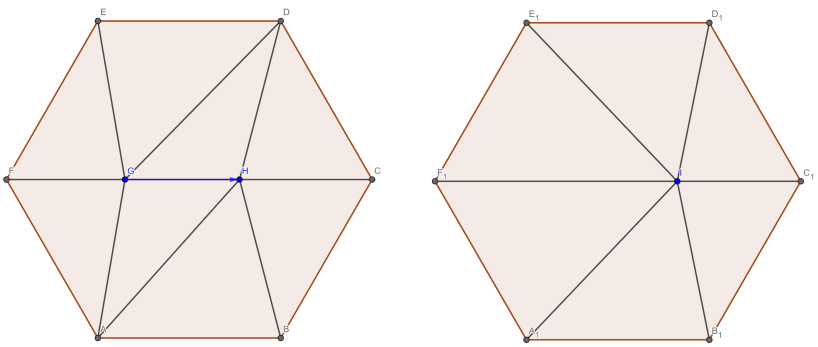
Metoda urušavanja polubridova vrlo je slična metodi urušavanja bridova, te je jedina razlika u činjenici da ova metoda ne unosi nove vrhove već se dva vrha koje dijeli urušeni brid spajaju u jedan. Ostali dio algoritma je isti kao i kod urušavanja bridova, a glavna prednost ove metode je jednostavnost i efikasnost s obzirom na stupanj slobode manje kod pozicioniranja vrha koji mijenja brid, te zadržavanje topologije mreže s obzirom da se ne unose novi vrhovi.



(a) Uklanjanje vrhova



(b) Urušavanje bridova



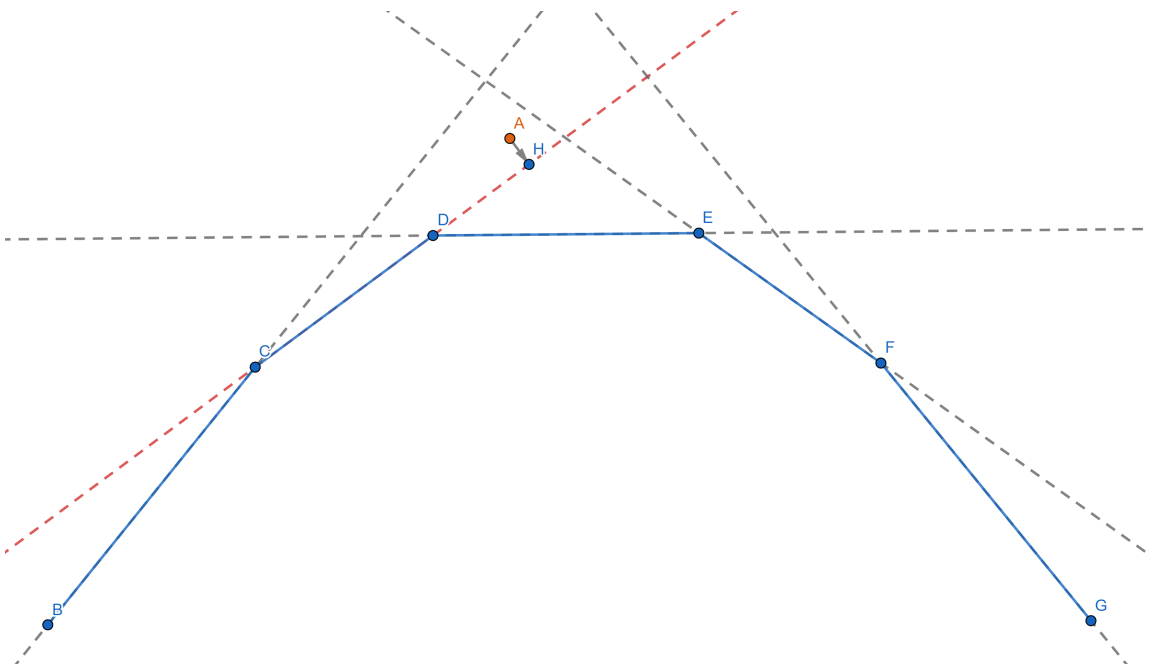
(c) Urušavanje polubridova

Slika 2.1 Iterativne tehnike pojednostavljivanja 3D mreže

2.2 Korištene metrike preciznosti

2.2.1 Udaljenost vrha od plohe

Metoda udaljenosti vrha od plohe ima mnoge varijacije razvijene tijekom godina. Ukoliko problem gledamo iz konteksta uklanjanja vrhova, pitanje je u odnosu na koju plohu računati pogrešku uklonjenog vrha. Jedno rješenje je uzeti plohu prema prosječnoj udaljenosti svih okolnih vrhova, no takav pristup je problematičan s obzirom da ne daje realno odstupanje novonastale strukture u odnosu na originalnu mrežu. Drugo, preciznije rješenje, jest zbrojiti kvadratne udaljenosti uklonjenog vrha od ploha na kojima leže svi novonastali trokuti koji ispunjavaju poligon. S obzirom da ovakav pristup koristi jednostavne matematičke operacije, vrlo je brz i precizan, ali problem koji ovakav pristup ima jest podcjenjivanje pogreške kod zakrivljenih površina s obzirom da ne uzima u obzir pripada li projicirana točka na plohu trokutu ili ne. S obzirom da je problem analogan istome problemu u dvije dimenzije, na slici 2.2 možemo bolje vidjeti navedeni problem:



Slika 2.2 Podcjenjivanje pogreške pri udaljenosti vrha od plohe

Vrh A predstavlja vrh koji želimo ukloniti iz krivulje. Točka H predstavlja točku na pravcu na kojoj leži dužina $C-D$ najbližu vrhu A. Možemo vidjeti da je ovakvim pristupom udaljenost $A-H$ puno manja nego realna udaljenost od vrha A do najbliže točke dužine $C-D$, te je time pogreška manja nego što bi trebala biti. Uzmemo li još u obzir da bi se susreli s istim problemom u slučaju ovakve metrike udaljenosti sa ostalim dužinama (osim dužine $D-E$), vidimo koliko je ovaj pristup neprecizan kod vrlo zakrivljenih segmenata.

Drugi problem koji ovakav pristup ima jest problem izrazito velikih brojeva. Veličina 3D modela javno dostupnih na internetu jako varira pa se mogu pronaći modeli koji su modelirani u izrazito velikom prostoru. Zbog toga kvadratna odstupanja mogu dostići vrlo velike vrijednosti te, čak i ako se uzme prosječno kvadratno odstupanje, radi se o vrlo velikim vrijednostima.

Konačno, rješenje koje je korišteno u ovom radu jest suma kvadratnih udaljenosti uklonjenih točaka do najbližih površina na novome modelu. Konkretno, za svaku uklonjenu točku, nakon ispunjavanja zaostale šupljine trokutima, traži se najmanja kvadratna udaljenost od uklonjenog vrha do novih trokuta. U ovom radu to je izvedeno tako da se za svaki novonastali trokut provjerava dali se projicirana točka na plohu nalazi unutar trokuta ili ne, te se za svaki takav trokut uzima najmanja kvadratna udaljenost. Provjera nalazi li se točka unutar trokuta problem je analogan provjeri sjecišta zrake i trokuta koji se može susresti kod algoritma praćenja zrake, te se rješava *Möller-Trumbore* algoritmom. Jedina razlika od originalnog *Möller-Trumboreovog* algoritma jest da se u slučaju ovoga rada ne radi o sjecištu zrake koja je opisana usmjerenim vektorom već točki u prostoru pa se samim time pri projekciji koristi normala kao smjer u kojem se projicira vrh.

Označimo li uklonjeni vrh sa $\mathbf{v}$, njegovu projekciju na plohu na kojoj leži trokut s vrhovima $\mathbf{p_1 p_2 p_3}$ možemo označiti sa $\mathbf{v'}$. Kako bi dobili projekciju $\mathbf{v'}$, potrebno je prvo izračunati normalu plohe na kojoj leži trokut pomoću koje možemo dobiti udaljenost vrha od plohe te projekciju točke na plohu. Normalu plohe $\hat{\mathbf{n}}$ moguće je dobiti vektorskim produktom dvije stranice trokuta pri čemu treba paziti na orijentaciju s obzirom da vektorski produkt nije komutativna operacija, te dobiveni vektor normalizirati.

$$\mathbf{n} = (\mathbf{p2} - \mathbf{p1}) \times (\mathbf{p3} - \mathbf{p1}) \tag{2.1a}$$

$$\hat{\mathbf{n}} = \frac{\mathbf{n}}{\|\mathbf{n}\|} \tag{2.1b}$$

Nadalje, udaljenost d možemo dobiti skalarnim produktom novodobivene normale sa vektorom koji povezuje bilo koju točku na plohi sa točkom $\mathbf{v}$.

$$d = (\mathbf{v} - \mathbf{p_0}) \cdot \hat{\mathbf{n}} \tag{2.2}$$

Na kraju, projicirani vrh $\mathbf{v}'$ se može dobiti translacijom vrha v u smjeru suprotnom normali na plohu za izračunatu udaljenost d.

$$\mathbf{v}' = \mathbf{v} - d * \hat{\mathbf{n}} \tag{2.3}$$

Kada imamo točku $\mathbf{v}'$, preostaje još odrediti nalazi li se ona unutar trokutu ili ne, pri čemu izvrsno koriste baricentrične koordinate. Kako bi se bolje razumjelo baricentrične koordinate, slika 2.3 prikazuje točku $\mathbf{v}'$ unutar trokuta **p1p2p3**, te se može primjetiti da površinu trokuta $\mathbf{p_1p_2p_3}$ čini suma površina trokuta $\mathbf{p_1p_2v'}$, $\mathbf{p_2p_3v'}$ i $\mathbf{p_3p_1v'}$.

Ukoliko omjere površina manjih trokuta u odnosu na površinu velikog trokuta označimo sa u, v i w, možemo primjetiti da vrijedi:

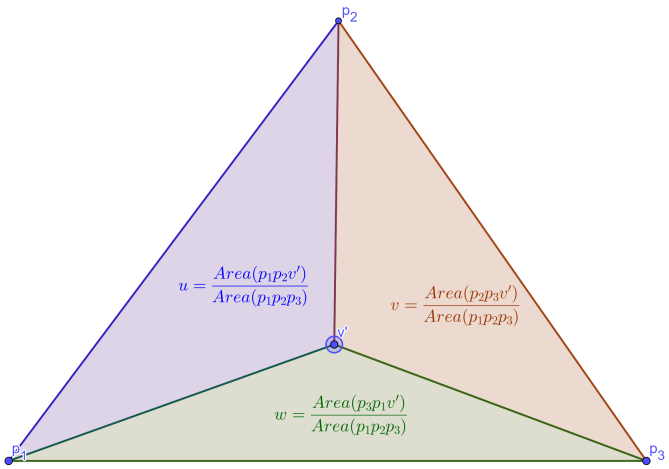
$$u + v + w = 1 \tag{2.4a}$$

$$0 \leq u, v, w \leq 1 \tag{2.4b}$$

te na temelju toga svaku točku unutar trokuta možemo prikazati pomoću baricentričnih koordinata na sljedeći način:

$$P = u\mathbf{p_1} + v\mathbf{p_2} + w\mathbf{p_3} \tag{2.5}$$

Prema izrazu (2.4b) možemo primjetiti da ukoliko je i jedna baricentrična koordinata manja od 0 ili veća od 1, točka se ne nalazi unutar trokuta te takvu relaciju



Slika 2.3 Baricentrične koordinate

možemo iskoristiti za provjeru nalazi li se točka unutar trokuta. Koristan detalj koji treba imati na umu pri izračunavanju baricentričnih koordinata je i da je magnituda vektorskog produkta stranica trokuta ujedno i površina paralelograma kojeg te stranice zatvaraju, odnosno dvostruka površina trokuta[6]. Ta informacija je korisna zbog efikasnosti izračuna s obzirom da se zbog projiciranja točke **v** i potrebe za računanjem normale plohe pomoću stranica velikog trokuta njegova površina može dobiti i u prethodnom koraku algoritma.

2.2.2 Udaljenost plohe od plohe

Metrika udaljenosti plohe od plohe oslanja se na *Hausdorff* udaljenost koja predstavlja udaljenost između dva podseta točaka u prostoru. S obzirom da *Hausdoff* udaljenost između dva podseta nije jednaka s obe strane, ukupna *Hausdoff* udaljenost odnosi se na veću od dvije jednostane udaljenosti. Matematički *Hausdorff* udaljenost možemo prikazati na sljedeći način:

$$d_{H_1}(\mathcal{M}_1, \mathcal{M}_2) = max_{p \in \mathcal{M}_1} min_{q \in \mathcal{M}_2} \|p - q\| \tag{2.6a}$$

$$d_H(\mathcal{M}_1, \mathcal{M}_2) = max\{d_{H_1}(\mathcal{M}_1, \mathcal{M}_2), d_{H_2}(\mathcal{M}_1, \mathcal{M}_2)\} \tag{2.6b}$$

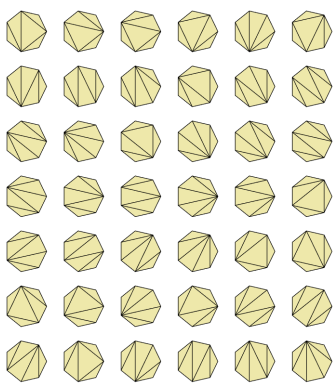
Dakle, u kontekstu pojednostavljivanja mreže, korištenjem *Hausdorff* udaljenosti računa se udaljenost između novonastale površine i površine na originalnoj mreži. Također treba napomenuti kako se u svrhu bržeg izračunavanja koristi *Hausdorff* udaljenost sa heuristikom.

2.3 Triangulacija mreže

U odjeljku 2.1.1 rečeno je kako nakon procesa uklanjanja vrha iz mreže u mreži ostaje šupljina koju je potrebno triangulirati. Svaki konveksni poligon može se triangulirati sa dijagonalama koje se međusobno ne presjecaju na $\frac{n(n+1)...(2n-4)}{(n-2)!}$ načina (drugi Catalanov broj) [2]. Na slici 2.4 može se vidjeti primjer heptagona te svih 42 načina trianguliranja poligona.

Ukoliko u obzir uzmemo da ovisno o metrici kojom evaluiramo kvalitetu triangulacije nije svako rješenje ekvivalentno dobro, kroz vrijeme su nastali razni algoritmi triangulacije poligona sa svojim prednostima i nedostacima.

S obzirom da je triangulacija problem za sebe koji ima brojna rješenja, u ovom je samo dan uvod u problem. U svrhu triangulacije korištena je *CGAL* programska knjižnica koja efikasno rješava navedeni problem korištenjem koncepta *Delaunay-eve triangulacije* koja maskimizira minimum svih kutova novonastalih trokuta te se bazira na *Voronoi* dijagramu[7].



Slika 2.4 Triangulacija heptagona na svih 42 načina (preuzeto iz [2])

2.4 Evolucijski pristup

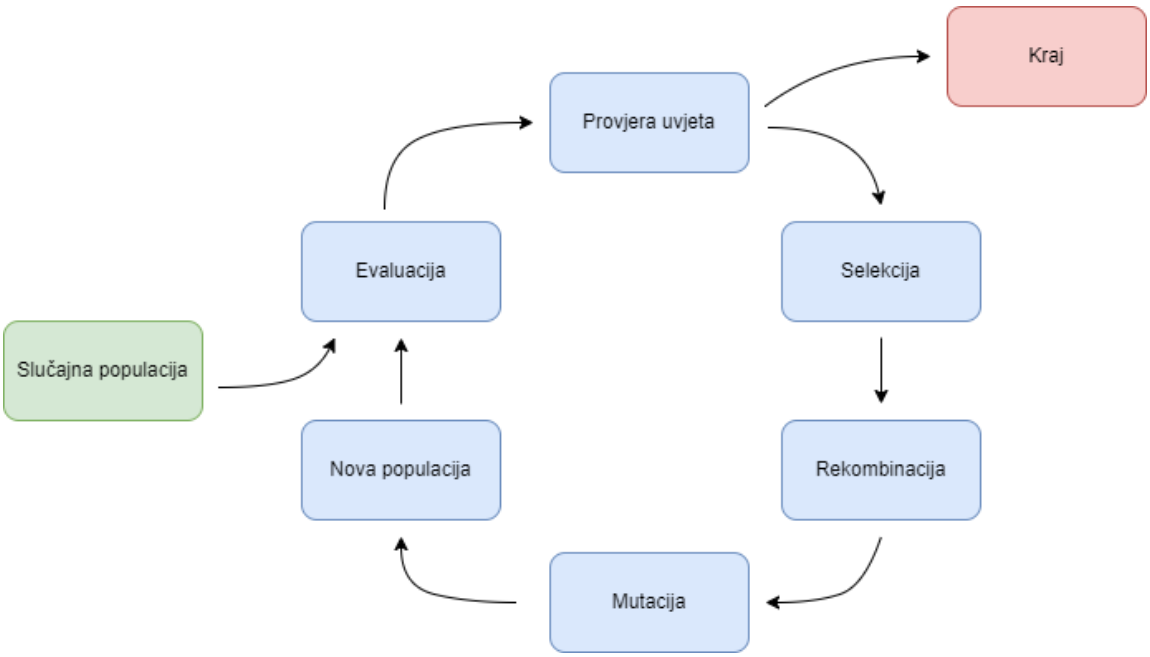
Dosada opisane iterativne tehnike funkcionaju na način da se odaberu kandidati za pojednostavljivanje (vrhovi ili bridovi), te se na temelju izračunate metrike redom provode operacije pojednostavljivanja mreže dok se ne dostigne određeni kriterij. Takav pristup može biti iscrpan s obzirom na količinu operacija pri određivanju redoslijeda nad kojim kandidatom prvo provoditi pojednostavljivanje. Također treba uzeti u obzir i ogromnu količinu vrhova koje današnji 3D modeli imaju te činjenicu da je izuzetno teško znati koji kandidate odabrati kako bi dobili optimalno rješenje. Iz tog razloga cilj ovog rada je razmotriti stohastički pristup problemu, odnosno probati dobiti suboptimalna rješenja evolucijskim pristupom.

Evolucijski algoritmi temeljeni su na ideji da se nasumično odabare N jedinki koje predstavljaju kandidate za rješenje problema te se njihova rješenja kombiniraju kako bi se dobila bolja, a odbacila loša rješenja. Na taj način kroz vrijeme rješenja "evoluiraju", te se nakon određenog broja generacija ili dostignute ciljne metrike dolazi do suboptimalnog rješenja. Način na koji rješenja evoluiraju ovisi od algoritma do algoritma, no u ovom radu riječ je isključivo o genetskom algoritmu, točnije o njegovoj višekriterijskoj inačici po imenu *NSGA-II*.

2.4.1 Genetski algoritam

Kako bi razumjeli algoritam *NSGA-II* na kojem je usredotočen ovaj rad, potrebno je prvo razumjeti ideju i motivaciju genetskog algoritma. Ideju genetskog algoritma kao metaheurističkog rješenja razvio 1992. godine J.H.Holland na temelju Darwinove ideje biološke evolucije. Sam algoritam sastoji se od N jedinki koje predstavljaju populaciju, fitness funkcije koja određuje dobrotu rješenja, te operatora nad jedinkama temeljenima na biološkoj evoluciji a to su rekombinacija, mutacija i selekcija. Slika 2.5 prikazuje dijagram toga genetskog algoritma gdje se može vidjeti da algoritam počinje sa inicijalizacijom slučajne populacije te se svakom generacijom evaluira funkcija dobrote, koriste genetski operatori selekcije, mutacije i rekombinacije i stvara nova populacija dok se ne zadovolji određeni uvjet.

Jedinke genetskog algoritma mogu predstavljene na razne načine ovisno o pro-



Slika 2.5 Dijagram toka genetskog algoritma

blemu koji se pokušava riješiti, pa to mogu biti binarna polja, prikaz cijelim ili realnim brojevima, prikaz permutacijama i slično. S obzirom da je u ovom radu jedinka kodirana upravo binarnim poljem gdje sa 1 predstavlja vrh koji se uzima u rezultantu mrežu, a s 0 onaj koji se ne uzima odnosno koji se uklanja, u nastavku rada se i u kontekstu genetskih operatora podrazumijeva binarni kromosom kao jedinka populacije.

Svaki genetski operator ima svoju ulogu u genetskom algoritmu kao i više različitih varijacija kako funkcionira. Selekcija, odnosno odabir roditelja, određuje brzinu konvergencije rješenja, rekombinacija je osnovni operator pretraživanja dok je mutacija pretraživanje okoline roditelja u svrhu bijega iz lokalnih ekstrema. Konkretne varijacije operatora korištene u ovome radu detaljnije su objašnjene u pododjeljku 2.4.2 s obzirom da sam algoritam određuje korištene operatore. S obzirom da genetski algoritam u klasičnom obliku funkcionira u kontekstu optimizacije po jednoj funkciji cilja, a fokus ovog rada je optimizacija po dvije funkcije cilj, potrebno je koristiti inačicu genetskog algoritma pod nazivom *NSGA-II*.

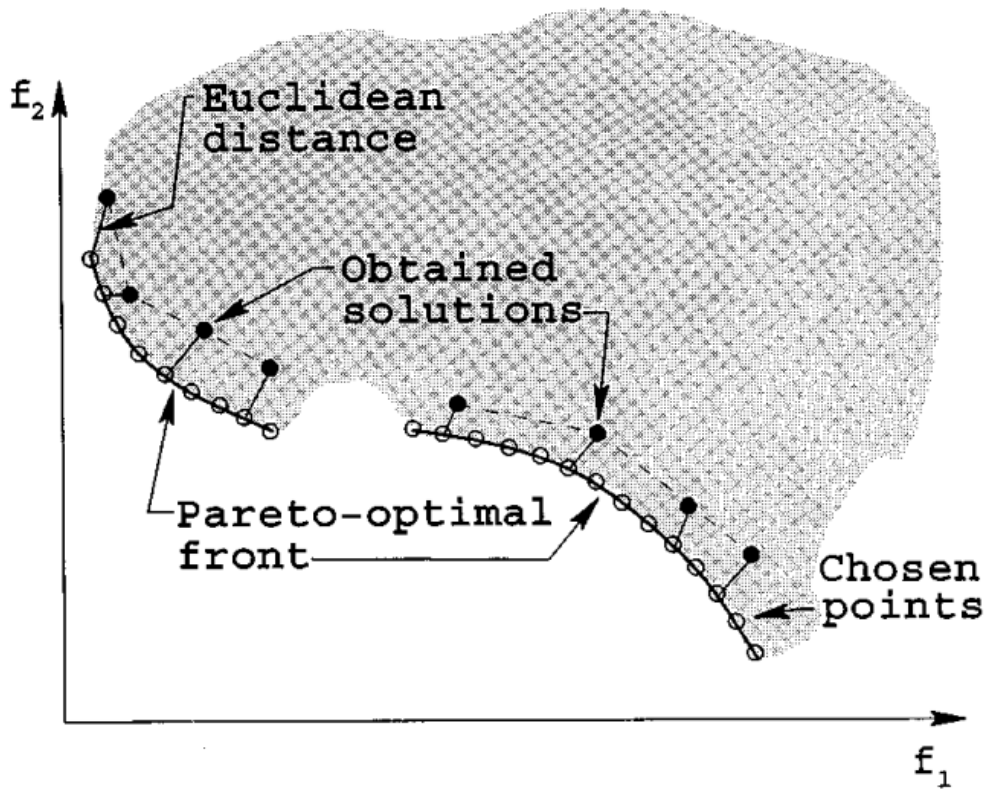
```
fast-non-dominated-sort(P)
for each p ∈ P
   S_p = ∅
   n_p = 0
   for each q ∈ P
      if (p ≺ q) then
         S_p = S_p ∪ {q}
      else if (q ≺ p) then
         n_p = n_p + 1
   if n_p = 0 then
      p_rank = 1
      F_1 = F_1 ∪ {p}
i = 1
while F_i ≠ ∅
   Q = ∅
   for each p ∈ F_i
      for each q ∈ S_p
         n_q = n_q − 1
         if n_q = 0 then
            q_rank = i + 1
            Q = Q ∪ {q}
   i = i + 1
   F_i = Q
```

Slika 2.6 Nedominantno sortiranje - pseudokod (preuzeto iz [3])

2.4.2 *NSGA-II*

NSGA-II algoritam nastao je kao efikasno rješenje višekriterijske optimizacije u domeni evolucijskih algoritama. Ono što višekriterijski pristup čini drugačijim od klasičnog je pitanje kako odrediti dominantno rješenje u slučaju ni jedan kandidat ne dominira po svim funkcijama dobrote nad drugim rješenjem. Upravo iz tog razloga *NSGA-II* koristi brzo nedominantno sortiranje u Pareto fronte, gdje su sva rješenja u istoj fronti ekvivalentno dobra, elitistički pristup zadržavanju kandidata i operator "niširanja" bez parametara[3]. Pseudokod nedominantnog sortiranja prikazan je na slici 2.6.

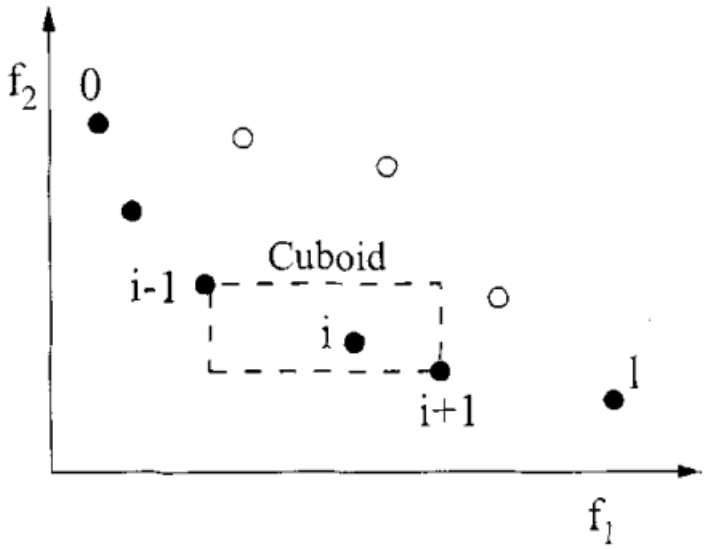
Rezultat nedominantog sortiranja su upravo jedinke raspoređene u Pareto fronte gdje sva rješenja u jednoj fronti dominiraju nad svima rješenjima u sljedećoj fronti, a bivaju dominirana od svih rješenja u prethodnoj fronti. Slika 2.7 jasnije prikazuje ideju Pareto-optimalne fronte.



Slika 2.7 Pareto-optimalna fronta

Iz istoga možemo zaključiti da se sva suboptimalna rješenja algoritma nalaze u prvoj, Pareto-optimalnoj fronti, s obzirom da nad njima ne dominira ni jedno rješenje, a međusobno su ekvivalentno dobra.

Novo pitanje koje se postavlja je kako evaluirati rješenja iz iste fronte pri selekciji i izboru jedinki za novu generaciju ako su sva rješenja ekvivalentno dobra. S obzirom da je jedan od ciljeva evolucijskih algoritama maksimalno proširiti prostor pretrage kako ne bi zapeli u lokalnim optimumima, unutar jedne fronte se preferiraju rješenja koja su manje sklona grupiranju. Kako bi se to postiglo prvo je potrebno sva rješenja u fronti sortirati uzlazno po svakoj varijabli te prvom i zadnjem rješenju dodijeliti beskonačnu udaljenost grupiranja. Zatim je potrebno izračunati najveću udaljenost po svakoj varijabli, te za svako rješenje izračunati udaljenost od susjednih rješenja s



Slika 2.8 Udaljenost grupiranja (preuzeto iz [3])

obje strane i podijeliti sa maksimalnom udaljenosti kako bi dobili omjer udaljenosti u odnosu na maksimalnu moguću udaljenost rješenja. Matematički to možemo zapisati na sljedeći način:

$$d_{I_j^m} = d_{I_j^m} + \frac{f_m(\mathbf{x}^{I_{j+1}^m}) - f_m(\mathbf{x}^{I_{j-1}^m})}{f_m^{max} - f_m^{min}} \tag{2.7}$$

pri čemu m označava jedinku u fronti, a I^m index jedinke u fronti. Možemo primjetiti na slici 2.8 da je udaljenost grupiranja svakog rješenja analogna površini kuboida u odnosu na kuboid koji pokriva cijelu frontu.

Nakon izračunavanja udaljenosti grupiranja pri selekciji i izboru jedinki za novu generaciju prvo se gleda rank (broj fronte kojoj jedinka pripada), pa onda udaljenost grupiranja gdje se preferira jedinka sa većom udaljenosti grupiranja. Prema tako definiranom kriteriju odabira boljih rješenja, mogu se konkretno specificirati genetski operatori korišteni u ovom radu.

Kako je određivanja boljeg rješenja kod višekriterijskog pristupa specifično, najpraktičnije rješenje za operator selekcije je binarna turnirska selekcija pri čemu se

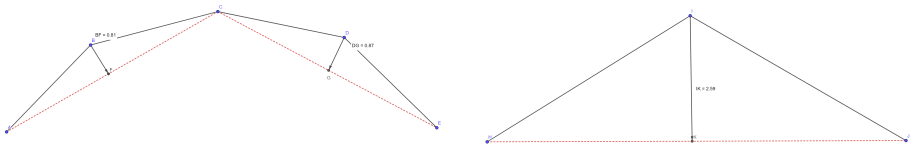
bira rješenje boljeg ranka, odnosno veće udaljenosti grupiranja ako se radi o istom ranku. Treba napomenuti i da se u ovom radu koristi turnirska selekcija s ponavljanjem isključivo zbog optimizacije, odnosno potrebnih operacija zbog provjera već korištenih rješenja. Analogno selekciji i kod izabira jedinki za novu generaciju koristi se elitistički pristup što znači da se redom uzimaju bolja rješenja po frontama od onog sa najvećom udaljenosti grupiranja prema onom sa najmanjom udaljenosti grupiranja. Na taj način se osim odabira najkvalitetnijih rješenja osigurava i širina prostora pretraživanja pa je veća šansa pronaći optimume koje je teže pronaći pohlepnim pristupom. Operator rekombinacije ne ovisi o algoritmu *NSGA-II* s obzirom da je neovisan o izboru rješenja, već ovisi isključivo o reprezentaciji jedinke. Kako se u ovom radu koristi binarni kromosom kao reprezentacija, potrebno je u svrhu rekombinacije koristiti jedno od varijacija križanja kromosoma. U ovom je radu korišteno uniformno križanje jer je eksperimentima dalo zadovoljavajuće rezultate.

2.4.3 Funkcija dobrote

Funkcija dobrote metrika je prema kojoj određujemo kvalitetu rješenja te je samim time kriterij s kojim određujemo koja rješenja želimo zadržati u populaciji a koja ne. U klasičnom genetskom algoritmu samo je jedan kriterij određivanja koje je rješenje bolje pa je samim time potrebna i jedna funkcija dobrote, no ukoliko pričamo o višekriterijskoj optimizaciji, potrebno je imati jednu funkciju dobrote za svaki kriterij. Ono što je bitno naglasiti jest da su funkcije dobrote najčešće obrnuto proporcionalne te bolja rješenja po jednom kriteriju često znače i lošija rješenja po drugom kriteriju. U kontekstu ovoga rada slučaj je upravo takav - cilj je minimizirati broj vrhova mreže 3D modela uz što manje odstupanje od originalnog modela po metrici opisanoj u potpoglavlju 2.2.1.

Ukoliko razmotrimo činjenicu da je uklanjanjem svakog vrha u modelu pogreška minimalno jednaka 0 za ravne segmente, a veća od 0 kod zakrivljenih segmenata, možemo vidjeti da dobar izbor vrhova koje se želi ukloniti iz modela izravno određuje pogrešku koju pojednostavljeni model ima. Na slici 2.9 možemo vidjeti navedeni problem u dvije dimenzije.

Iz istoga se da zaključiti i da ukoliko bolje biramo više vrhova možemo dobiti



(a) 2 uklonjena vrha, mala pogreška (b) 1 uklonjeni vrh, velika pogreška

Slika 2.9 Usporedba pogreške pri različitom odabiru vrhova

bolju aproksimaciju originalnog modela nego u slučaju lošeg izbora manjeg broja vrhova pri uklanjanju. Upravo zato je u ovom radu korišten *NSGA-II* kako bi se pretražio što veći broj mogućih rješenja uz zadržavanje što manjeg broja vrhova, te minimalnu pogrešku.

Poglavlje 3

Implementacija

3.1 Radno okruženje

Za izradu popratnog programa korišten je programski jezik *C++* primarno zbog mogućnosti direktnog korištenja *OpenGL* aplikacijskog programskog sučelja za iscrtavanje grafike čiji je izvorni kod napisan u programskom jeziku C. Uz standardne *C++* knjižnice, te knjižnicu standardnih predložaka (eng. *Standard Template Library* (STL)) korištena je i knjižnica *Cala* za jednostavno iscrtavanje grafike i rad sa ulazom s periferije, *CGAL* za algoritme računske geometrije zbog ispune šupljina trokutima, te *matplotlib-cpp* kao omotač za *Python matplotlib* modul, izuzetno popularan za crtanje grafova. S obzirom da se za crtanje grafova koristi *Python* modul, potrebno je na računalu imati i *Python* interpreter.

Za generiranje okruženja za izgradnju programa korišten je *CMake* koji je *de facto* standard u svijetu *C++-a*. *CMake* je alat koji standardizira proces izgradnje programa pisanih u *C* i *C++* tako što generira okruženje i sve potrebne datoteke za više manje sve poznate sustave za izgradnju kao što su *Makefile, Visual Studio, Ninja* itd. Zahvaljujući *CMake-u* programeri koji rade sa različitim prevodiocima mogu na jednak način izgraditi program te na njemu surađivati.

Za samo kodiranje korišten je *Visual Studio Code* koji, zahvaljujući svojoj svestranosti, malim potrebama za resursima te mogućnosti nadogradnje brojnih dodataka (eng. *plug-in*), postaje sve popularniji alat za pisanje koda. Također je korišten i

CMake Tools dodatak za *Visual Studio Code* zbog kojeg je rad sa *CMake-om* puno brži i lakši.

3.2 Učitavanje modela

Kako bi uopće bilo moguće provesti *NSGA-II* i pojednostaviti model, potrebno je model učitati u radnu memoriju. Kako je i opisano u potpoglavlju 1.3.1, glavna prednost *OBJ* formata i razlog zašto je korišten u kontekstu ovog rada je jednostavnost parsiranja i čitljivost. S obzirom da je delimiter koji odvaja novi unos novi red, dovoljno je datoteku učitavati red po red te parsirati ovisno o početnoj labeli. U tu svrhu korišteno je zaglavlje `fstream` iz standardne knjižnice koja nudi sve funkcije i klase za manipulaciju datotekama u obliku tokova (eng. *stream*).

3.3 Struktura koda

Kod cijelog programa strukturiran je u četiri cjeline: *ModelOptimizer.h* i *ModelOptimizer.cpp* datoteke koje sadržavaju `ModelOptimizer` klasu koja sadrži sve dijelove evolucijskog algoritma za pojednostavljivanje modela, *main.cpp* datoteku od kojeg kreće izvođenje programa, *MathUtils.cpp* i *MathUtils.h* datoteke koje sadrže pomoćne matematičke funkcije te *MyApplication.cpp* i *MyApplication.h* koja učitava model, poziva kod za optimizaciju te isrctava modele na scenu.

3.4 Višedretveno izvođenje

Paralelizacija evolucijskog algoritma jedna je od glavnih tehnika ubrzanja izvođenja algoritma. Ako uzmemo u obzir da se prilikom generiranja početne populacije i stvaranja nove djece sve jedinke generiraju neovisno jedna o drugoj, to daje izvrsnu priliku za paralelizaciju tih procesa. Iako su sami procesi neovisni, treba paziti na detalje koji mogu uzrokovati krivo izvođenje te neželjene rezultate programa.

Jedan takav problem je generiranje nasumičnih brojeva koje često funkcionira

na temelju *seed* vrijednosti kojom se inicijalizira generator. Ukoliko se ne pazi na sinkronizaciju pri pristupu generatoru, može se dogoditi da više dretvi pristupaju istoj vrijednosti u generatoru što može dovesti do sličnosti između jedinki te samim time suženom prostoru pretrage rješenja. Za to postoji više rješenja, no dva korištena u ovom radu su zaključavanje pristupa generatoru sa *mutex-ima* i kreiranje varijabli lokalnih za dretvu. S obzirom da se drugo rješenje pokazalo boljim za performanse, zadržalo se i u finalnoj verziji programa.

Sama implementacija višedretvenosti realizirana je pomoću funckije `std::async` iz zaglavlja `future` koje prima funkcijski objekt i argumente funkcije a vraća objekt tipa `std::future` u kojem se pohranjuje rješenje nakon završetka izvođenja funkcije. Bitno je imati na umu da se destruktor `std::future` objekta ne smije pozvati prije završetka funkcije koja je pozvana paralelno s obzirom da se objekt neće uništiti dok god funkcija ne završi izvođenje.

3.5 Kontrole

Kako bi bilo jednostavnije pregledavati iscrtanu scenu, iskorišten je ulaz sa periferije za manipulaciju programom. Na početku izvođenja programa sa standardnog ulaza se učitavaju parametri programa počevši sa modelom kojeg se pojednostavljuje, te parametrima evolucijskog algoritma (broj generacija ($G \in [1, 200]$), veličina populacije ($P \in [1, 500]$), broj potomaka ($O \in [1, P]$), vjerojatnost mutacije ($M \in [0.0, 1.0]$), te broj kandidata turnirske selekcije ($TP \in [1, 10]$)). Nakon učitavanja parametara slijedi optimizacija modela te se nakon G generacija iscrtava graf pareto fronti, te se iscrtavaju modeli na sceni.

Kada program krene sa izvođenjem glavne petlje za iscrtavanje, scena se može manipulirati sa sljedećim kontrolama:

- W,A,S,D, *Left Shift*, razmak - pomak kamere po *uw* osi kamere
- L - uključivanje i isključivanje LOD moda
- P - uključivanje i isključivanje moda za crtanje poligona bez ispune
- Pomak miša uz desni klik - promjena orijentacije kamere.

Poglavlje 4

Rezultati

U svrhu prezentacije rezultata, korišteni su besplatno dostupni 3D modeli koji se mogu preuzeti na poveznicama iz dodatka B. Bitno je naglasiti da svi grafovi pareto fronti na x osi sadrže postotak zadržanih vrhova u modelu, a na y osi pogrešku u grafičkim jedinicama koja direktno ovisi o veličini originalnog modela. Mogućnost LOD-a podešena je tako da se ovisno u unesenom broju LOD razina N_{lod} učitava model niže rezolucije svakih d_{lod} grafičkih jedinica prema formuli:

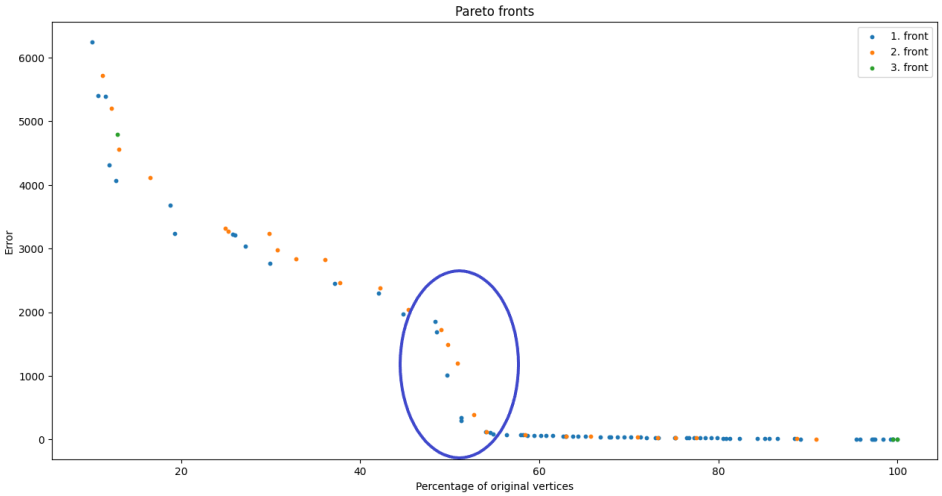
$$d_{lod} = \frac{100}{N_{lod}} \tag{4.1}$$

s time da se nakon 100 grafičkih jedinica udaljenost koristi model najniže rezolucije. Mjeren broj sličica u sekundi usrednjena je vrijednost broja sličica u sekundi posljednjih 50 ciklusa iscrtavanja. Treba napomenuti i da su svi rezultati dobiveni na istome računalu u istim uvjetima rada.

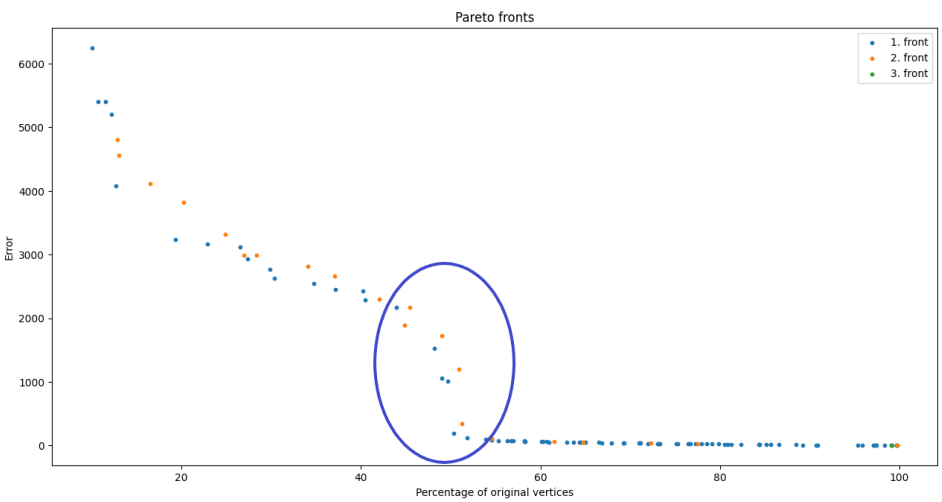
4.1 Model glave

Prvi korišteni model je model glave koji sadrži 299,712 vrhova i 99,904 trokuta. Slika 4.1 prikazuje grafove pareto fronti nakon 10 i 20 generacija algoritma *NSGA-II*. U populaciji je korišteno 100 jedinki, 5 potomaka po generaciji, faktor mutacije 0.1 i 5 kandidata u turnirskoj selekciji. Rezultati nisu provedeni za više od 20 generacija

zbog dugotrajnosti izvršavanja s obzirom na veličinu modela.



(a) 10 generacija

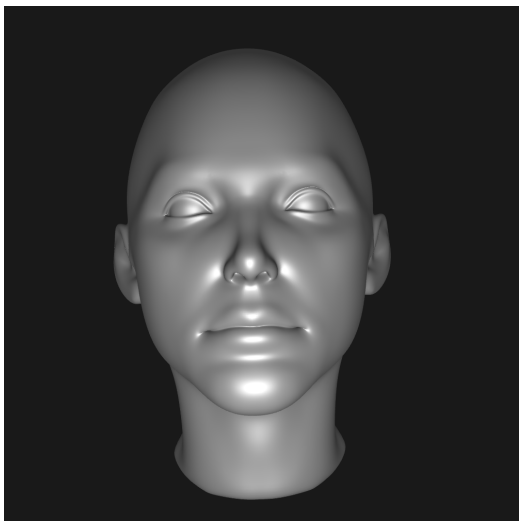


(b) 20 generacija

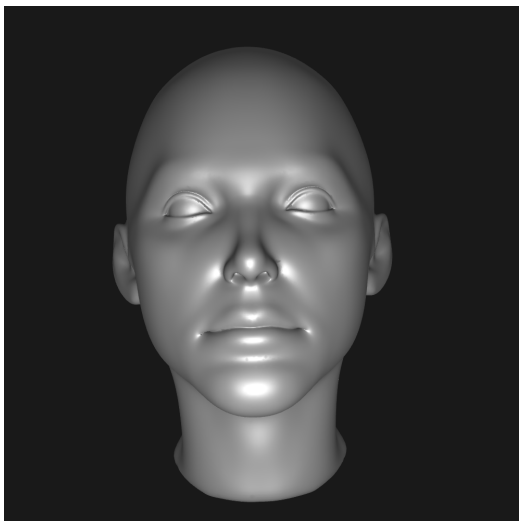
Slika 4.1 Graf pareto fronti za $P = 100$, $O = 5$, $M = 0.1$ i $TP = 5$

Kao što je vidljivo na grafovima pareto fronti, broj generacija ne čini značajnu razliku te se može zaključiti da dodatni broj generacija nije potreban s obzirom na

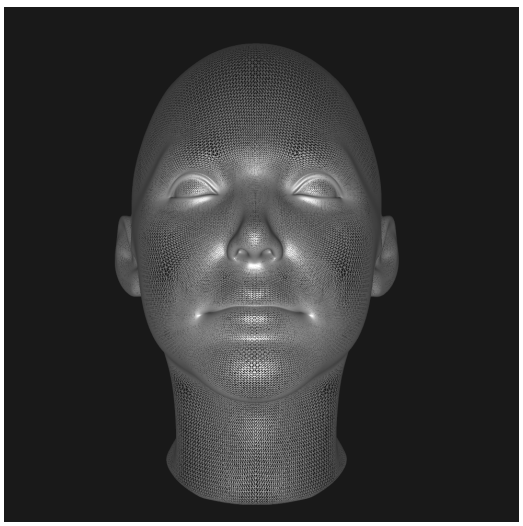
količinu vremena koja je potrebna za izvršenje optimizacije. Također treba primjetiti nagli prijelaz na otprilike 50% vrhova modela označen na slici gdje sa smanjenjem broja vrhova pogreška naglo raste. Pretpostavka koja objašnjava takvu teoriju jest da oko 50% mreže objekta čine površine koje nemaju velike prijelaze (npr. tjeme i čelo), te algoritam pretragom prostora prvo uklanja te vrhove jer oni uzrokuju najmanju pogrešku. To se može i vidjeti na slici 4.2d gdje je vidljivo da je sa obraza modela glave, koji su prilično ravni, uklonjeno više vrhova nego sa dijelova poput nosa. Prikaz modela sa i bez LOD-a te sa i bez ispune trokuta vidljiv je na slici 4.2.



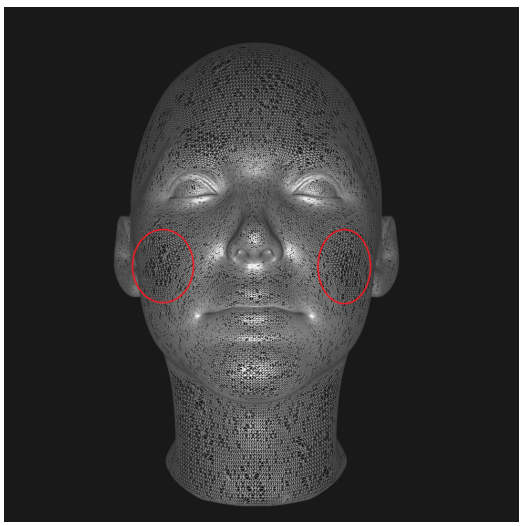
(a) Prikaz modela sa ispunom trokuta, LOD isključen



(b) Prikaz modela sa ispunom trokuta, LOD uključen



(c) Prikaz modela bez ispune trokuta, LOD isključen



(d) Prikaz modela bez ispune trokuta, LOD uključen

Slika 4.2 Prikaz modela glave

4.2 Model tijela

Drugi korišteni model je model ljudskog tijela koji sadrži 146,754 vrha i 48,918 trokuta. Slika 4.3 prikazuje grafove pareto fronti nakon 10, 20 i 100 generacija algoritma *NSGA-II*. Korišteno je 200 jedinki u populaciji, 10 potomaka po generaciji, faktor mutacije 0.1, te 7 kandidata u turnirskoj selekciji.

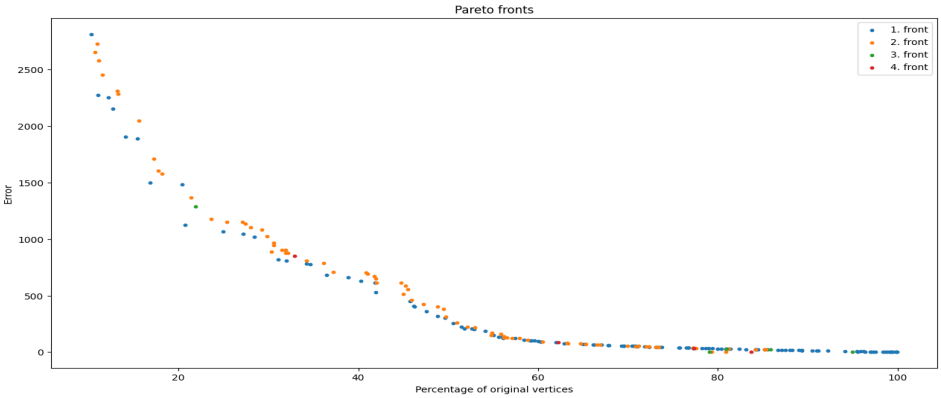
Prvo što se može primjetiti jest da je ne sva tri grafa forma krivulja slična te da je sa više generacija manje raspršena pa su samim time i prijelazi više očiti. Bitno je primjetiti da krivulja ima dvije bitne točke prijeloma, označene na zadnjem prikazanom gradu: jednu na cca 60% vrhova a drugu na cca 40%. Vidljivo je da od 100% do 60% zadržanih vrhova greška modela raste izuzetno sporo, od 60% do 40% nešto brže, te od 40% na dalje kreće rasti jako brzo. Kao i kod prvog modela, može se pretpostaviti da algoritam prvo uklanja ravnije dijelove modela koji sadržavaju oko 40% modela. Slična analogija može se upotrijebiti i za raspon do sljedeće točke prijeloma na 40% gdje se vjerojatno uklanjaju vrhovi na onim prijelazima koji su nešto strmiji ali nisu preekstremni.

Na slici 4.4 vide se vizualni rezultati iscrtane scene sa i bez LOD-a. Scena sadrži 441 model (21 x 21) centriranih oko nule. Kao što se može primjetiti, sa uključenim dinamičkim LOD-om vizualna kvaliteta scene se ne mijenja bitno dok se ista scena iscrtava gotovo dvostruko brže (26.5 naspram 13.5 sličica u sekundi).

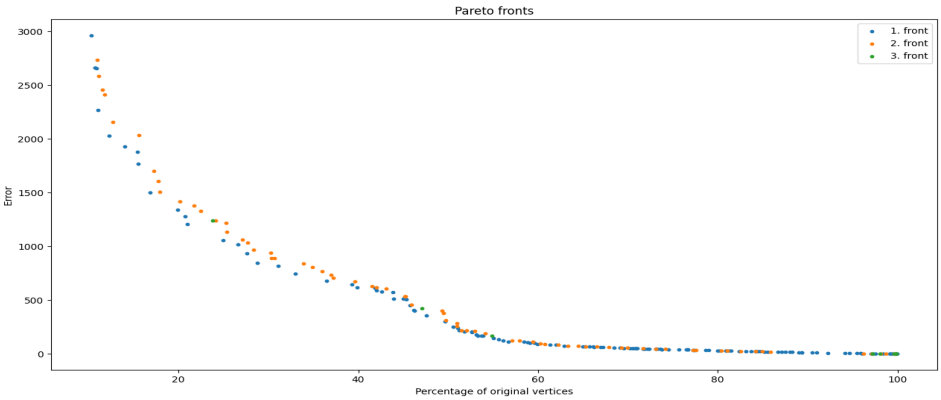
4.2.1 Treći model

Treći korišteni model je model ljudske ruke koji sadrži 299,136 vrhova i 99,712 trokuta. Slika 4.5 prikazuje grafove pareto fronti nakon 10 generacija algoritma *NSGA-II*. U populaciji je korišteno 100 jedinki, 5 potomaka po generaciji, faktor mutacije 0.1 i 5 kandidata u turnirskoj selekciji.

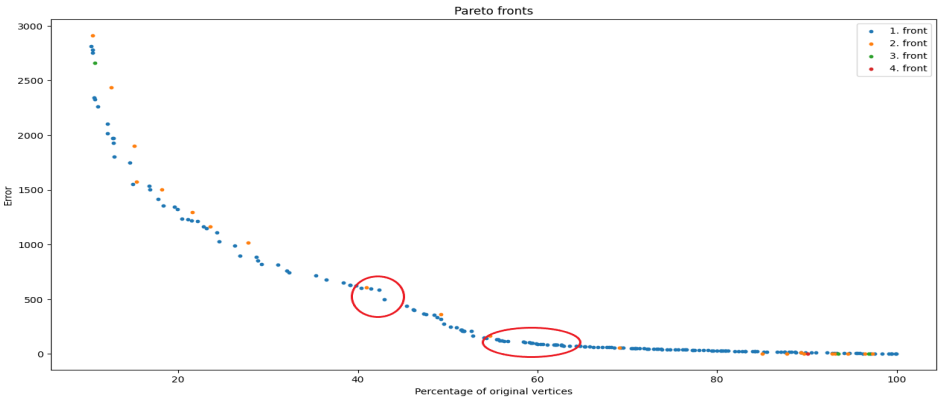
Kao i kod prva dva modela, i ovaj model pokazuje prijelaze koji se mogu primjetiti na 40% i 50%, vjerojatno iz istih razloga kao i kod modela tijela. Na slici 4.6d se može vidjeti gdje je na "glađim" dijelovima modela, označenih crveno, mreža rijeđa od dijelova poput zglobova prstiju, označenih zeleno.



(a) 10 generacija

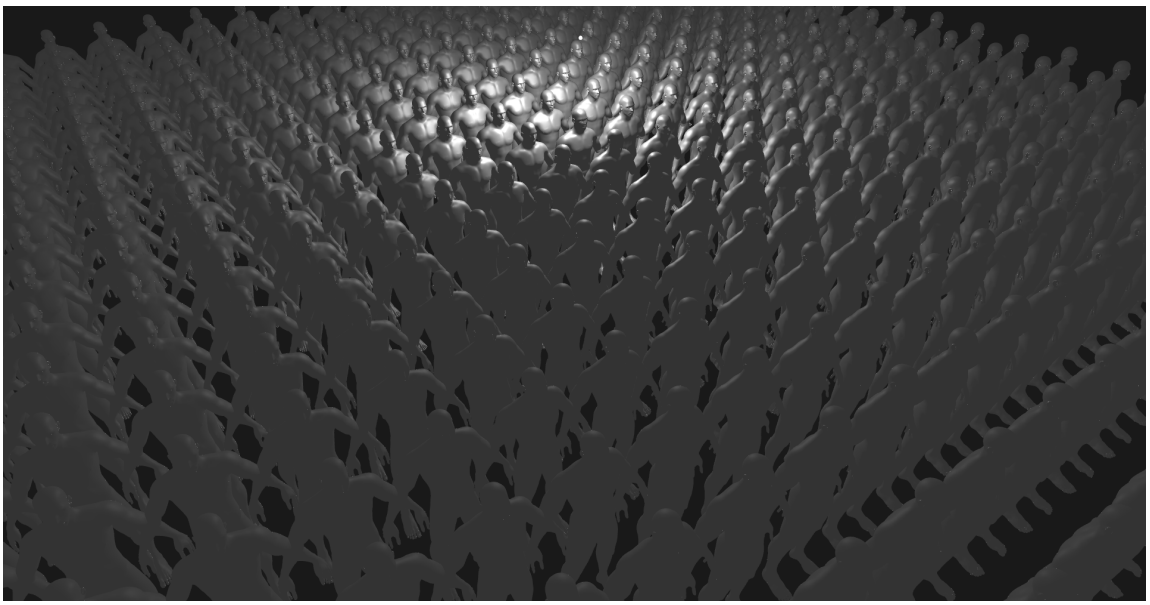


(b) 20 generacija

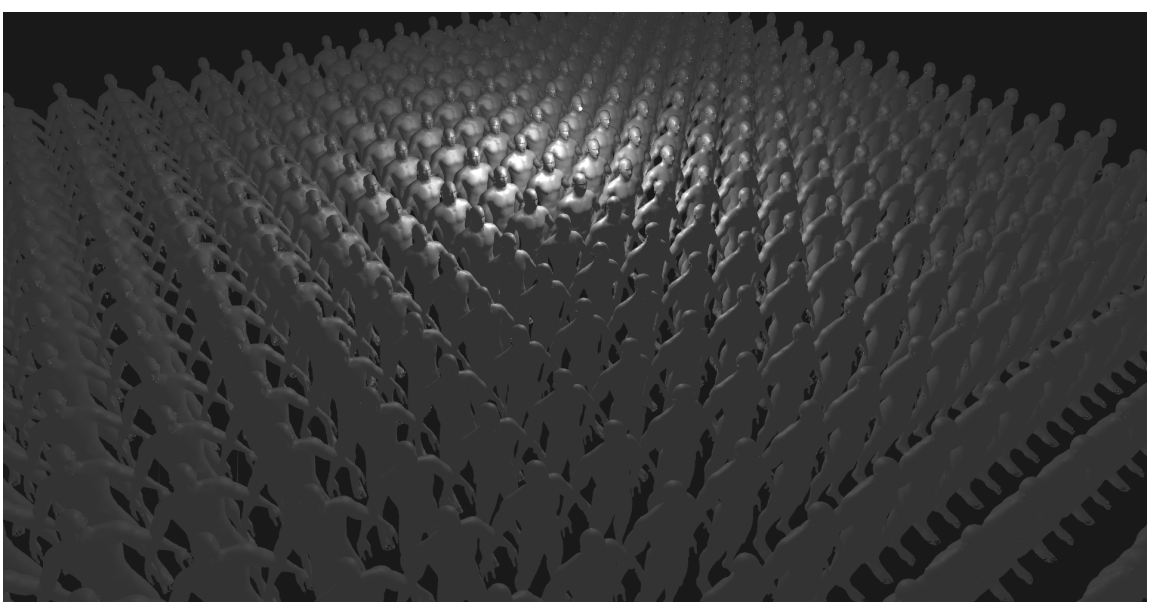


(c) 100 generacija

Slika 4.3 Graf pareto fronti za $P = 200$, $O = 10$, $M = 0.1$ i $TP = 7$

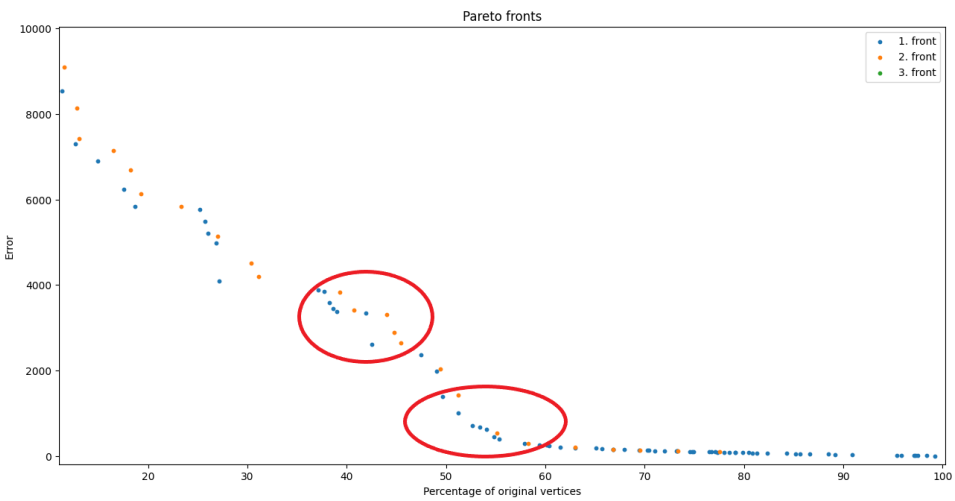


(a) Isključen LOD - 13.5 sličica po sekundi

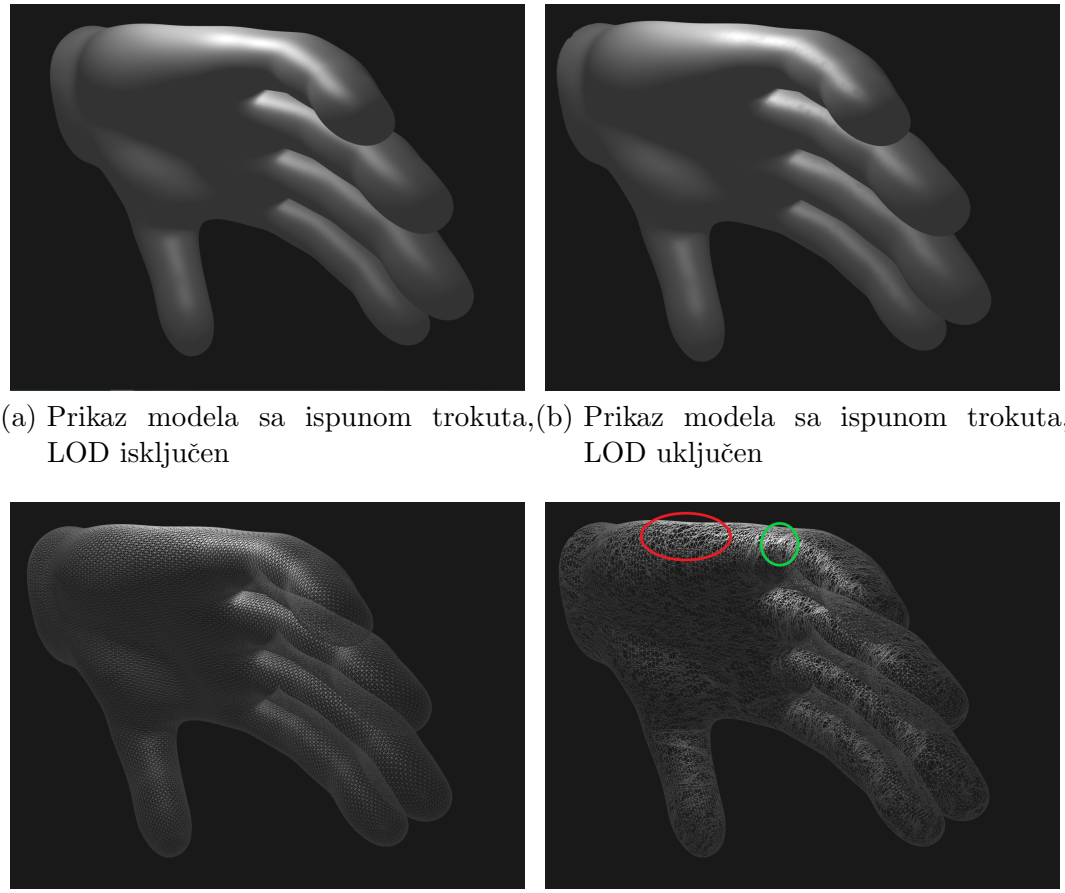


(b) Uključen LOD - 26.5 sličica po sekundi

Slika 4.4 Prikaz scene sa i bez uključene LOD mogućnosti za $G = 100$, $P = 200$, $O = 10$, $M = 0.1$ i $TP = 10$



Slika 4.5 Graf pareto fronti za $P = 100$, $O = 5$, $M = 0.1$ i $TP = 5$



(a) Prikaz modela sa ispunom trokuta, LOD isključen

(b) Prikaz modela sa ispunom trokuta, LOD uključen

(c) Prikaz modela bez ispune trokuta, LOD isključen

(d) Prikaz modela bez ispune trokuta, LOD uključen

Slika 4.6 Prikaz modela ruke

Zaključak

Ovim radom uspješno je prikazano kako se višekriterijska opzimizacija u obliku genetskog algoritma može uspješno iskoristiti za pojednostavljivanje topologije 3D modela. Kroz tri primjera modela sa velikim brojem vrhova i različitim topologijama, mogu se primjetiti jednaki uzorci, a to je da kod svakog modela algoritam uspješno pojednostavi model do neke razine uz vrlo malu pogrešku, pa se nakon određenog broja vrhova pogreška naglo poveća. Iz toga se da zaključiti da opsežnom pretragom prostora algoritam prvo pojednostavljuje glatke površine koje rezultiraju malim pogreškama, pa tek onda kreće sa dijelovima mreže s većim prijelazima. Također je vidljivo kako se ti prijelazi događaju i u nekoliko navrata, ovisno o kompleksnosti modela pa se negdje može primjeiti jedno "koljeno", dok na drugim modelima i više "koljena". Uz to se može primjetiti da broj generacija ne čini preveliku razliku kod nalaženja suboptimalnih rješenja pa se najčešće ne isplati pokretati algoritam sa više od 10 generacija.

Treba napomenuti i da ovaj pristup daje vrlo dobre rezultate, ali je prilično spor, pogotovo kod izuzetno velikih modela u usporedbi sa metodom spomenutom u uvodu. To može i ne mora biti problem ovisno o primjeni algoritma, no ukoliko je potrebno izgenerirati pojednostavljene modele samo jednom, brzina algoritma nije ključna.

Jedan od problema koji nije riješen je i činjenica da se smanjenjem broja vrhova gubi i informacija o normalama i koordinatama tekstura u tom vrhu. Posljedično je vidljivo da, iako modeli ostaju detaljni, osvjetljenje često izgleda krivo i prijelazi boja izgledaju naglo i nekonzistentno. U budućnosti bi se rad mogao unaprijediti tako da se nakon retrianguliranja šupljina sve normale u vrhovima koji su ostali isprave usrednjivanjem normala svih trokuta koji dijele taj vrh. Također se slično može

napraviti i za koordinate tekstura. Dodatno bi se u budućnosti algoritam mogao poboljšati boljom triangulacijom koja bi minimizirala metriku pogreške, testiranjem više vrsta rekombinacijskih tehnika i odabirom najpovoljnije i slično.

Bibliografija

[1] "Mesh simplification," , kolovoz 2023. , s Interneta, https://graphics.stanford.edu/courses/cs468-10-fall/LectureSlides/08_Simplification.pdf

[2] "Polygon triangulation," , kolovoz 2023. , s Interneta, https://en.wikipedia.org/wiki/Polygon_triangulation

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[4] "Level of detail (computer graphics)," , kolovoz 2023. , s Interneta, https://en.wikipedia.org/wiki/Level_of_detail_(computer_graphics)

[5] E. Ovreiu, "Accurate 3d mesh simplification," Ph.D. dissertation, INSA de Lyon, 2012.

[6] "Ray-tracing: Rendering a triangle - barycentric coordinates," , kolovoz 2023. , s Interneta, https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates.html

[7] "Delaunay triangulation," , kolovoz 2023. , s Interneta, https://en.wikipedia.org/wiki/Delaunay_triangulation

[8] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 209–216.

[9] B. R. Campomanes-Álvarez, S. Damas, and Ó. Cordón, "Mesh simplification for 3d modeling using evolutionary multi-objective optimization," in *2012 IEEE Congress on Evolutionary Computation*. IEEE, 2012, pp. 1–8.

[10] R. A. Potamias, S. Ploumpis, and S. Zafeiriou, "Neural mesh simplification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 18 583–18 592.

Pojmovnik

3DS *3D Studio*. 5

CLOD *Continuous Levels of Detail*. 4, 5

COLLADA *COLLAborative Design Activity*. 5

DLOD *Discrete Levels of Detail*. 4, 5

LOD *Level of Detail*. vi, viii, 3, 4, 7, 25, 26, 28–30, 32, 34, 38

STL *Standard Tessellation Language*. 5

STL *Standard Template Library*. 23

Sažetak

Ovaj radi bavi se višekriterijskom optimizacijom topologije 3D modela evolucijskim pristupom. Nakon predstavljanja problema i neka od dosadašnjih uspješnih rješenja, predstavljeno je rješenje algoritmom *NSGA-II* pri čemu se optimizira broj vrhova modela i pogreška u odnosu na originalni model.

Na kraju su prikazani rezultati algoritma na tri modela, te je dokazana prednost korištenja pojednostavljenih modela u korist performansi. Rad prati i programski kod napisan u programskom jeziku *C++* te je javno dostupan na *GitHub-u*.

Optimizacija 3D modela, LOD, genetski algoritam, NSGA-II —

Abstract

This thesis is covering an evolutionary approach to multiobjective optimization of 3D model topology. After presenting the problem and some of the currently used solutions, the solution using *NSGA-II* is presented, which optimizes number of model vertices and error in regards to original model.

At the end, the results of the algorithm using three models are shown, while proving that using simpler models benefits performance. Thesis includes publicly available source code written in *C++* that can be found on *GitHub*.

3D model optimization, LOD, genetic algorithm, NSGA-II —

Dodatak A

Izvorni kod

Izvorni kod nalazi se na *GitHub* repozitoriju na sljedećoj poveznici: `https://github.com/dominikcondric/EvolutionaryMeshSimplification.git`

Dodatak B

Korišteni 3D modeli

Modeli korišteni u radu mogu se besplatno preuzeti sa sljedećih poveznica:

- Model glave - `https://free3d.com/3d-model/femalehead-v4--971578.html`
- Model ljudskog tijela - `https://free3d.com/3d-model/male-base-mesh-6682.html`
- Model ruke - `https://free3d.com/3d-model/hand-v1--675788.html`