SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2710

DINAMIČKA PROMJENA RAZINE SLOŽENOSTI OBJEKATA

Filip Pavletić

Zagreb, veljača 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2710

DINAMIČKA PROMJENA RAZINE SLOŽENOSTI OBJEKATA

Filip Pavletić

Zagreb, veljača 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 15. listopada 2021.

DIPLOMSKI ZADATAK br. 2710

Pristupnik:	**Filip Pavletić (0036487923)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Željka Mihajlović

Zadatak: **Dinamička promjena razine složenosti objekata**

Opis zadatka:

Proučiti tehnike prikaza i promjene dinamičke složenosti objekata (LOD Level of Detail) i pripadnih tekstura ovisno o udaljenosti od promatrača. Razraditi mogućnosti primjene ovakvih promjena na prototipnoj aplikaciji. Posebice obratiti pažnju na promjene u brzini izvođenja ovisno o primjeni promjene složenosti u prikazu. Načiniti testiranje na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Koristiti programski alat Unity. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 4. veljače 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2710

Dinamička promjena razine složenosti objekata

Filip Pavletić

Zagreb, veljača 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2710

Dinamička promjena razine složenosti objekata

Filip Pavletić

Zagreb, veljača 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 15. listopada 2021.

DIPLOMSKI ZADATAK br. 2710

Pristupnik:	**Filip Pavletić (0036487923)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Željka Mihajlović

Zadatak: **Dinamička promjena razine složenosti objekata**

Opis zadatka:

Proučiti tehnike prikaza i promjene dinamičke složenosti objekata (LOD Level of Detail) i pripadnih tekstura ovisno o udaljenosti od promatrača. Razraditi mogućnosti primjene ovakvih promjena na prototipnoj aplikaciji. Posebice obratiti pažnju na promjene u brzini izvođenja ovisno o primjeni promjene složenosti u prikazu. Načiniti testiranje na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Koristiti programski alat Unity. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 4. veljače 2022.

Zahvaljujem se svojoj mentorici, prof. dr. sc. Željki Mihajlović
na pomoći koju mi je udijelila i strpljenju kojeg mi je pokazala

Zahvaljujem se roditeljima i bratu na neograničenoj
podršci koju su mi pružili

Sadržaj

Uvod 7

1. Mreže poligona 8

1.1. Vrhovi 8

1.2. Poligoni 9

1.3. Normale 9

1.4. UV mape 10

1.5. Višestrukost 10

2. Razina detalja (LOD) 11

2.1. Sustav razina detalja u Unityu 11

2.1.1. Sustav razine detalja (LODGroup) 12

2.1.2. Razina detalja (LOD) 12

3. Algoritmi za pojednostavljivanje mreža poligona 13

3.1. Algoritam grupiranje vrhova 13

3.1.1. Grupiranje vrhova 14

3.1.2. Odabir nove reprezentativnog vrha 14

3.1.3. Izgradnja novih poligona 15

3.2. Algoritam uklanjanja vrhova 15

3.2.1. Odabir vrha za uklanjanje 15

3.2.2. Uklanjanje vrha 16

3.2.3. Izgradnja novih poligona 16

3.3. Algoritam uklanjanja bridova 17

3.3.1. Odabir brida za uklanjanje 17

3.3.2. Uklanjanje brida 17

3.3.3. Prespajanje poligona 18

4. Rezultati 19

4.1. Modeli stvoreni algoritmom grupiranja vrhova 20

4.2. Modeli stvoreni algoritmom uklanjanja vrhova 22

4.3. Modeli stvoreni algoritmom uklanjanja bridova 24

4.4. Utjecaj pojednostavljivanja objekata na performanse 26

Zaključak 28

Literatura 29

Sažetak 30

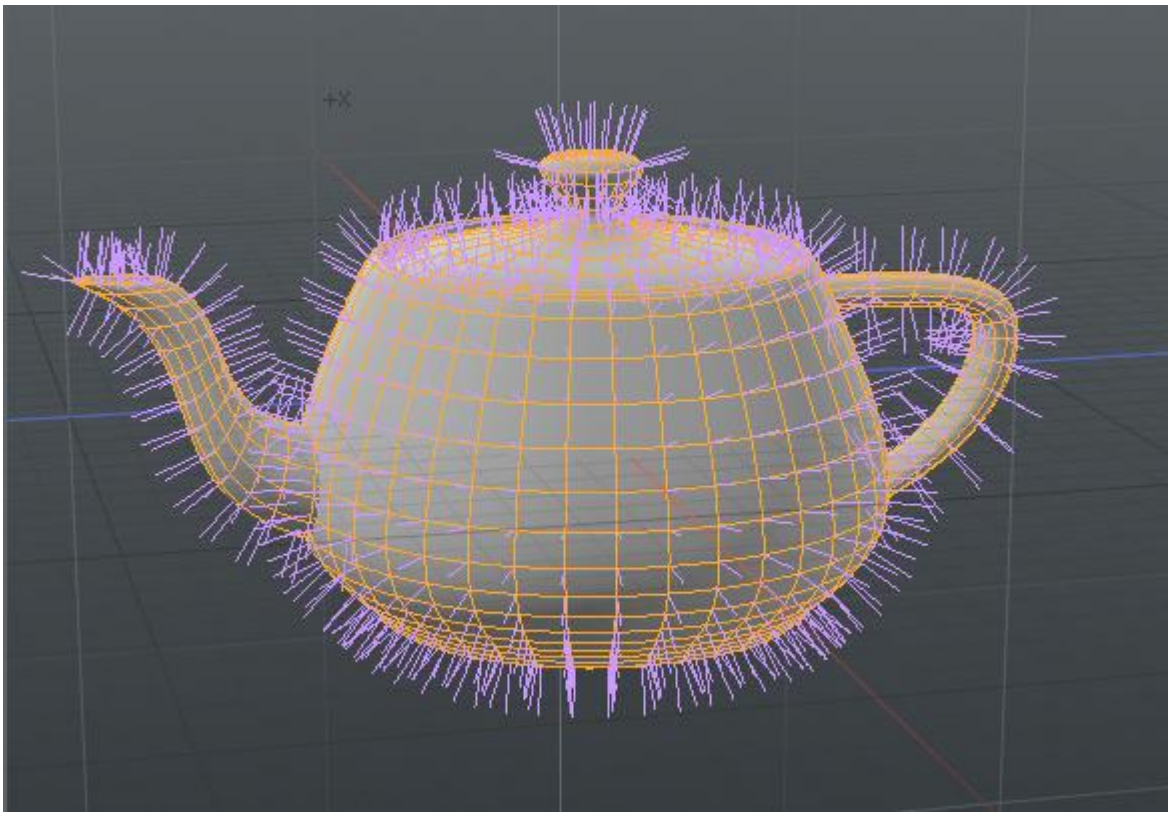
Summary 31

Uvod

Pedesetih godina prošlog stoljeća razvijena je prva video igra. Radilo se o jednostavnoj simulaciji tenisa koja se igrala na osciloskopu. Na zaslonu se mogla vidjeti horizontalna crta koja je predstavljala mrežu i tri točkice. Jedna za lopticu te jedna za svakog od igrača. U zadnjih 70 godina mnogo se toga promijenilo. Interaktivna računalna grafika izašla je iz domene razonode i našla primjenu u vojnoj, medicinskoj, i brojnim drugim industrijama. Uz povećane opsega računalne grafike rasla su i očekivanja. Svake godine pretpostavlja se da će vizualizacije biti realnije, detaljnije i gušće ispunjene nego prošlogodišnje. Kako Mooreov „zakon“ svake godine postaje sve bliži veoma optimističnoj nadi nego zakonu problemu moramo pokušati pristupati i iz drugih kutova. Ovaj rad bavi se jednim od tih pokušaja. Prikazujmo lošije modele da bismo mogli prikazati bolje; ili logičnije sročeno sustav razine detalja.

U prvom ćemo poglavlju proći osnovne pojmove računalne grafike poput vrhova, poligona i UV mapa. Nakon toga ćemo u drugom poglavlju razraditi tehniku razine detalja i ukazati na prednosti i mane korištenja iste. U trećem poglavlju predstavljamo i objašnjavamo rad tri algoritma za pojednostavljivanje modela; algoritma grupiranja vrhova, algoritma uklanjanja vrhova i algoritma uklanjanja bridova. Konačno, u četvrtom poglavlju pokazujemo dobivene pojednostavljene modele te ukazujemo na probleme svakog od algoritama te radimo usporedbu performansi scene ovisno o složenosti korištenih modela i agresivnosti primjene sustava razine detalja.

1. Mreže poligona



Sl 1.1 Mreža poligona s normalama

Mreže poligona osnovna su struktura podataka koju koristimo za modeliranje 3D objekata u računalnoj grafici. Dvije nužne informacije koje svaka mreža sadrži su podatci o vrhovima i podatci o poligonima. Uz njih mreže poligona mogu sadržavati i normale vrhova, tangente vrhova te informacije o preslikavanju UV mape na 3D model.

1.1. Vrhovi

Vrhovi su točke u prostoru pomoću kojih ćemo definirati poligone mreže. Zadani su svojim koordinatama u svakom od kardinalnih smjerova.

Vrhove koji pripadaju 2-višestrukoj mreži i sa svih su strana okruženi poligonima nazivamo jednostavnim vrhovima.

U Unityu su vrhovi mreže definirani koristeći strukturu podataka Vector3 [1].

1.2. Poligoni

Poligoni definiraju sto će se zapravo iscrtati na našim ekranima. U računalnoj grafici najčešće radimo s trokutima, no može se koristiti i druge konveksne poligone. Na slici (Sl 1.1) možemo vidjeti primjer mreže poligona gdje su bridovi poligona označeni narančastim crtama, Zadani su indeksima vrhova koje povezuju. Treba paziti na to da izrada prikaza zbog optimizacije najčešće iscrtava isključivo poligone koji su okrenuti prema kameri te je stoga redoslijed vrhova u poligonu bitan.

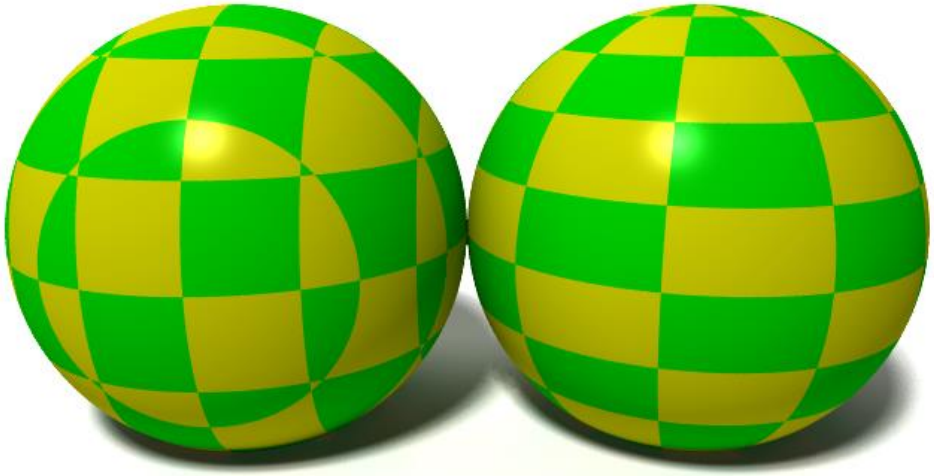
Poligone kojima je vise vrhova u istoj točki nazivamo degeneriranim poligonima.

U Unityu su poligoni definirani kao polje cijelih brojeva gdje svaka trojka predstavlja jedan trokut[1].

1.3. Normale

Normale vrhova nužne su kako bi se mogla pravilno odrediti orijentacija poligona. Uz to, također se koriste i pri sjenčanju. Na slici (Sl 1.1) su normale iscrtane ljubičastom bojom. Ukoliko normale nemamo zadane, moramo ih izračunati. Postoji nekoliko algoritama koji nam to omogućavaju. U Unityu su također definirane Vector3 strukturom podataka, a za računanje normala odabran je algoritam koji zbraja pa na kraju normira normale svih poligona pojedinačnog vrha [1].

1.4. UV mape



Sl 1.2 Prikaz mapiranja teksture na model bez (lijevo) i s (desno) korištenjem UV mape

UV mape potrebne su nam kako bi 2D teksturu pravilno precrtali na 3D objekte. One povezuju vrhove mreže s točkama u teksturi. Probleme koji se javljaju kada bez UV mape pokušamo primijeniti pravokutnu teksturu na sferičan objekt možemo vidjeti na slici (Sl 1.2). Prijelazom na kompleksnije objekte ovi problemi postaju sve veći. Unity UV koordinate mreže sprema u polje Vector2 struktura podataka [1].

1.5. Višestrukost

Svaka mreža poligona ima mjeru višestrukosti. Njena je vrijednost maksimalni broj poligona u mreži koji sadrže isti brid. Ova mjera bitna je jer u slučajevima kada njena vrijednost nije 2 neki od algoritama za pojednostavljivanje mreže gube svojstvo zadržavanja originalne topologije. Više o tome kasnije.

2. Razina detalja (LOD)

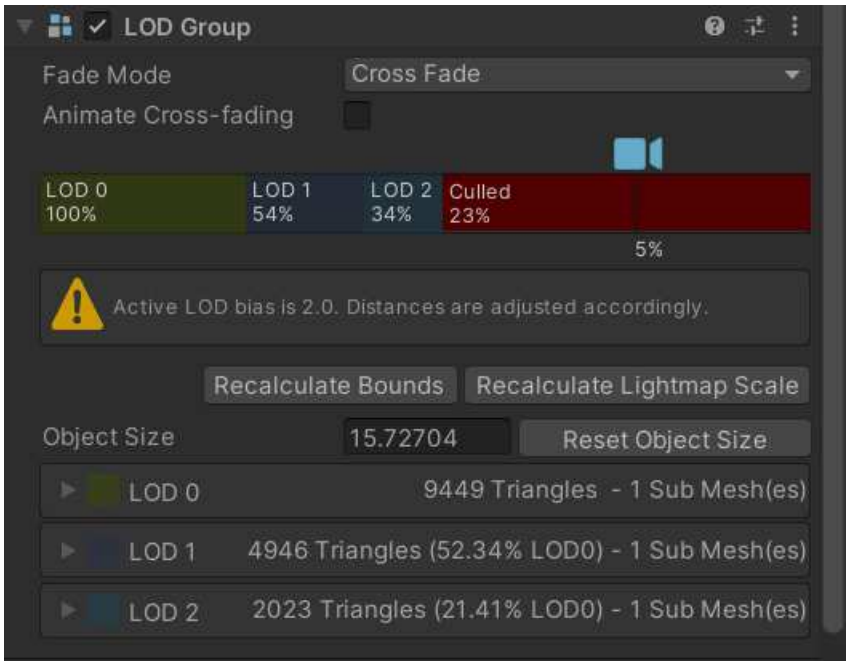
Želja programera, a i korisnika je da svaki objekt u svakoj sceni bude prikazan što je detaljnije moguće. Razmatrajući ubrzanja u radu grafičkih kartica kroz prošlo desetljeće, moglo bi se doći do zaključka da računalni resursi više ne predstavljaju problem u ovom pothvatu. Nažalost, to nije istina. Mogućnosti računala jesu vrtoglavo rasle, ali su uz njih rasli su i zahtjevi korisnika. Tako danas često radimo s 4k teksturama, a kvaliteta modela koja se pred desetak godina smatrala vrhunskom danas je neprihvatljiva. To nas dovodi do situacije gdje kao programeri imamo isti problem balansiranja između kvalitete i performansi kao i u prošlosti. Postoje razne metode kojima se služimo kako bismo doskočili ovom problemu. Jedna od njih su sustavi razine detalja.

Koncept je prvi put, doduše ne imenom, 1974. opisao James H. Clark u članku za listopadsko izdanje časopisa „Communications of the ACM“. Dok implementacijski detalji koje je naveo više nisu relevantni, ideja je. Dovoljno udaljeni objekti su nam na ekranu dovoljno maleni da nema smisla prikazivati modele pune kvalitete sa stotinama tisuća ili čak milijunima poligona. Scenu treba segmentirati pa prikazivanje prilagoditi uočljivosti objekta. Umjesto modela pune rezolucije mogu se koristiti pojednostavljeni modeli s pola, trećinom, desetinom ili još manjim brojem poligona u odnosnu na osnovni model. Sustav razine detalja ovisno o udaljenosti od kamere, postotku zauzeća ekrana ili nekom trećem parametru odabire razinu detalja modela koju iscrtava.

2.1. Sustav razina detalja u Unityu

Unity u svojoj standardnoj biblioteci sadrži nekoliko gotovih klasa za rad s diskretnim LOD-om. Diskretni LOD radi na principu ubacivanja jednog od dostupnih modela ovisno o veličini objekta. Kod diskretnog LOD-a može doci do veoma uočljivih naglih promjena pri zamjeni modela. Unity nudi *cross-fade* sjenčare koji donekle rješavaju problem. Za nas su bitne klasa LODGroup i struktura LOD.

Unity također podržava automatsko generiranje LODGroupa, ali samo ako za vrijeme prevođenja programa imamo pristup modelima svih razina detalja. Kako modele niže kvalitete generiramo tijekom izvođenja programa, ovu mogućnost ne možemo iskoristiti.



Sl 2.1 Sučelje s podatcima o stanju klase LODGroup

2.1.1. Sustav razine detalja (LODGroup)

Klasa LODGroup definira sva svojstva LOD sustava za jedan model [2]. Klasa podržava do 8 razina detalja. Posjeduje reference na prikazivače mreža s manje detalja te mapu prijelaza između njih. Mapa je napravljena u ovisnosti o postotku visine ekrana koji je zauzet objektom kojem je pridodana LODGroup instanca. Na slici (Sl 2.1) vidimo primjer stanja jedne instance klase LODGroup s postavljene 3 razine detalja i potpunim skrivanjem objekata čija je visina u prikazu manja od 23% visine ekrana. Mapu prijelaza i reference na prikazivače moguće je osvježavati u stvarnom vremenu koristeći metodu SetLODs(LOD[] lods).

2.1.2. Razina detalja (LOD)

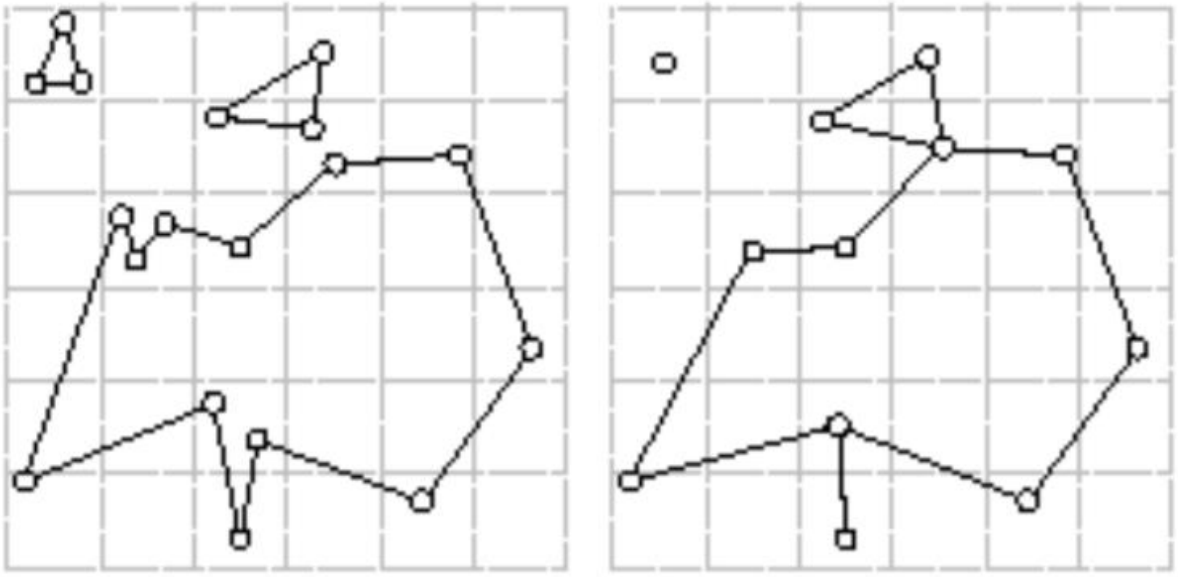
Klasa LOD je struktura koja definira minimalni postotak zauzetosti visine ekrana i listu prikazivača mreže poligona za jednu razinu detalja [3].

3. Algoritmi za pojednostavljivanje mreža poligona

Jedan od problema LOD sustava su resursi potrebni za izgradnju mreža manje rezolucije. Tražiti dizajnera da umjesto jednog dostavi 4 do 8 modela kako bi ih mogli koristiti za različite razine detalja je skupo, ali izvedivo. Postoje situacije kada jednostavnije modele uopće ne možemo dobiti. Skeneri 3D prostora u stvarnom vremenu generiraju ogroman broj točaka koje treba u stvarnom vremenu prikazati na mobilnim uređajima s često nezavidnim performansama. Ne postoji način da se pojednostavljivanje tih modela obavi ručno.

Iz razloga navedenih gore javila se potreba za algoritamskim načinima pojednostavljivanja mreža. Počevši u ranim ’90-im godinama prošlog stoljeća skoro svake godine na SIGGRAPHu pojavljuje se ili novi algoritam za pojednostavljivanje mreže ili značajni evolucijski napredak u već postojećem. U ovom radu baviti ćemo se s nekolicinom najznačajnijih.

3.1. Algoritam grupiranje vrhova



Sl 3.1 Grupiranje mreže i generiranje novih poligona

Algoritam grupiranja vrhova uvode J. Rossignac i Paul Borrel u svom radu iz 1993. godine [4]. Algoritam dijeli prostor modela na manje kvadrove, grupira sve vrhove u tim dijelovima u jedan koji predstavlja grupu te nakon toga prema povezanostima starih vrhova povezuje nove vrhove.

3.1.1. Grupiranje vrhova

Grupiranje vrhova radi se na vrlo jednostavan način. Odaberemo broj vrhova ciljne mreže poligona. Algoritam iz usporedbe trenutnog broja vrhova sa željenim izračuna dimenzije kvadrova na koje će dijeliti prostor. Svaku od koordinata svakog od vrha cjelobrojno podijelimo s brojem kvadrova u toj dimenziji te ih po dobivenoj vrijednosti grupiramo.

3.1.2. Odabir novog reprezentativnog vrha

Trenutno preferirani algoritam odabira novog vrha u slučajevima gdje nema strogih vremenskih ograničenja je Garland-Heckbertovog postupak [8]. Svakom originalnog vrhu dodijelimo simetričnu 4x4 matricu Q te grešku u vrhu **v** (v_1, v_2, v_3, 1) definiramo kao $\Delta v = \mathbf{v}^T Q \mathbf{v}$. Matrica Q predstavlja sve poligone koji se sastaju u vrhu **v**.

$$\Delta v = \Delta(v1, v2, v3, 1)^T = \sum_{\boldsymbol{p} \in ravnine(\boldsymbol{v})} (\boldsymbol{p}^T \boldsymbol{v})^2 \tag{3}$$

Koristeći formulu (3) uspjeli smo osigurati ovisnost o svakoj od pripadnih ravnina bez da pratimo svaki originalni poligon. Skup ravnina svakog vrha inicijalno su ravnine svih pripadnih poligona iz početne mreže.

$$\begin{aligned} \Delta v &= \sum_{\boldsymbol{p} \in ravnine(\boldsymbol{v})} (\boldsymbol{v}^T \boldsymbol{p})(\boldsymbol{p}^T \boldsymbol{v}) \\ \Delta v &= \sum_{\boldsymbol{p} \in ravnine(\boldsymbol{v})} \boldsymbol{v}^T (\boldsymbol{p}\boldsymbol{p}^T) \boldsymbol{v} \\ \Delta v &= \boldsymbol{v}^T \Big(\sum_{\boldsymbol{p} \in ravnine(\boldsymbol{v})} \boldsymbol{K_p} \Big) \boldsymbol{v} \end{aligned} \tag{4}$$

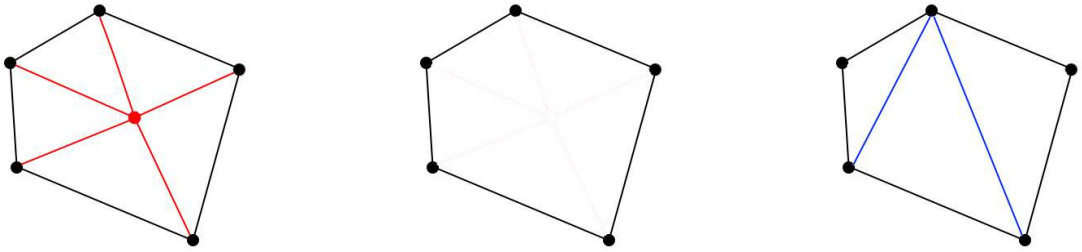
Pretvaranjem formule (3) u kvadratnu formu (4) možemo izlučiti matricu ravnina K_p. Suma svih K_p jednog vrha je njegov Q. Kada vrhove grupiramo, zbrajamo njihove Q matrice. Konačno poziciju novog reprezentativnog vrha računamo tako da minimiziramo umnožak inverza matrice Q i pozicije novog vektora **v'**.

Nažalost pri testiranju ovaj algoritam ne zadovoljava uvjet izvođenja u skoro stvarnom vremenu pa smo morali uzeti manje kvalitetan, ali puno brži način izračuna novih vrhova, prosjek starih.

3.1.3. Izgradnja novih poligona

Gradimo mapu koja povezuje originalne vrhove s novim zamjenskim vrhom u njihovom kvadru. U svakom poligonu mijenjamo stare vrhove za nove zamjenske vrhove. Uklanjamo duplikate i degenerirane poligone.

3.2. Algoritam uklanjanja vrhova



Sl 3.2 Uklanjanje vrha iz dijela mreže poligona

Algoritam uklanjanja vrhova prvi su predstavili Schroeder et al. [5]. Ovo je prvi od iterativnih algoritama koje ćemo koristiti. Algoritam kao što samo ime nalaže radi na principu uklanjanja vrhova. Na slici (Sl 3.2) vidimo tri koraka rada ovog algoritma. Odaberemo vrh koji želimo ukloniti, uklonimo ga i konačno susjede bivšeg vrha povežemo novim poligonima. Ovaj postupak ponavljamo dok ne uklonimo željeni broj vrhova. Algoritam održava topologiju mreže. Kako ne dodajemo nove vrhove algoritam je ograničen postojećima što može rezultirati lošijim rezultatima.

3.2.1. Odabir vrha za uklanjanje

Da bi vrh uopće bio razmatran za uklanjanje, on mora biti jednostavan. Ako vrh je jednostavan ubacujemo ga u prioritetni red. Prioritetni red postavljen je tako da se prvo vade članovi nižeg prioriteta. Prioritet vrha jednak je udaljenosti vrha od prosječne ravnine koju tvore njegovi poligoni.

$$\boldsymbol{N} = \frac{\sum \boldsymbol{n_i} * A_i}{\sum A_i}, \qquad \boldsymbol{n_i} = \frac{\boldsymbol{N_i}}{|\boldsymbol{N_i}|}, \qquad \boldsymbol{x} = \frac{\sum \boldsymbol{x_i} * A_i}{\sum A_i} \tag{1}$$

Da bismo izračunali udaljenost od proporcionalne ravnine svakom vrhu koji je zadovoljio gornji uvjet nalazimo pripadajuće poligone. Prosječna ravnina računa se prema formuli iz jednadžbe (1) gdje su $\mathbf{n}_i$ normale pripadnih poligona, $\mathbf{x}_i$ njihovi centroidi, a A_i njihove površine.

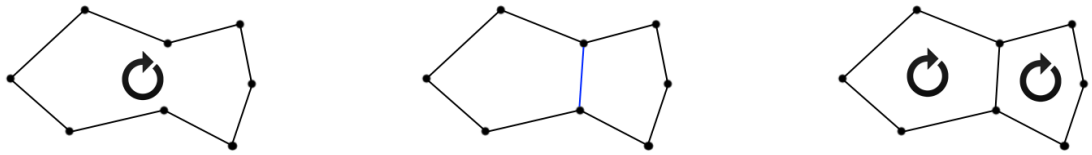
$$d = |\, \boldsymbol{n} \cdot (\, \boldsymbol{v} - \boldsymbol{x}\,)| \tag{2}$$

Konačno, prema formuli (2) računamo udaljenost vrha od njegove prosječne ravnine te vrh ubacujemo u prioritetni red.

3.2.2. Uklanjanje vrha

Uklanjanje vrha je vrlo jednostavan postupak. Prvo iz liste poligona uklonimo sve pripadne poligone pa onda iz liste vrhova uklonimo vrh.

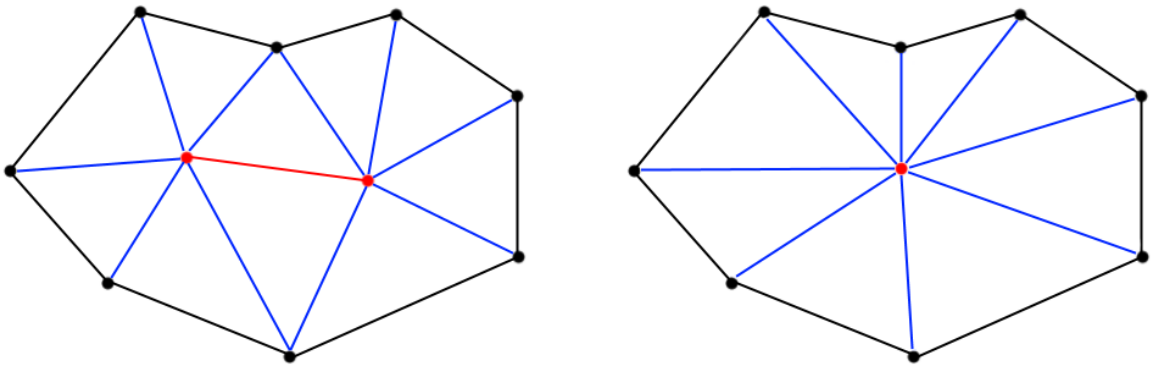
3.2.3. Izgradnja novih poligona



Sl 3.3 Korak u podjeli petlje na dvije petlje s manje članova

Za ponovnu izgradnju poligona odabrali smo algoritam podjele petlje. Susjede bivšeg vrha sortiramo tako da su prema njemu poredani u smjeru kazaljke na satu. Kada ih imamo tako sortirane od njih tvorimo petlju. Sada koristimo rekurzivni algoritam podjele petlje (Sl 3.3). Ako u petlja sadrži 3 vrha, ta 3 vrha postaju jedan od novih poligona mreže. Ako sadrži više vrhova moramo ju nekako podijeliti. Način koji smo odabrali temelji se na omjeru udaljenosti ravnine koju tvore taj brid i normala prosječne ravnine od najbližeg vrha petlje s duljinom brida po kojoj petlju dijelimo uz uvjet da odabrani brid ne smije povezivati dva susjedna vrha.

3.3. Algoritam uklanjanja bridova



Sl 3.4 Uklanjanje brida iz dijela mreže

Algoritam koji barata bridovima prvi su predstavili Hoppe et al. [6]. Taj algoritam osim uklanjanja bridova koristi i operacije koje su nazvali zamjena bridova i podjela bridova. Mi ćemo ipak koristiti algoritam koji je Hoppe objavio nekoliko godina kasnije. U svom radu [7] Hoppe je pokazao kako ove dodatne dvije operacije zapravo nisu nužne.

Ovaj algoritam kao i algoritam uklanjanja vrhova funkcionira iterativno. Odabiremo dva povezana vrha, odredimo željenu poziciju novog vrha pa poligone oba stara vrha povežemo s novim (Sl 3.4). Algoritam kao i prošli zadržava topologiju mreže, ali dopušta stvaranje novih vrhova što nam daje dodatan stupanj slobode pri pojednostavljivanju mreže.

3.3.1. Odabir brida za uklanjanje

Za odabir bridova opet koristimo prioritetni red sortiran uzlazno. Za izračun prioriteta uklanjanja brida postoje mnoge opcije. Vođen potrebom da se pojednostavljivanje obavlja što je bliže moguće stvarnom vremenu odabrao sam koristiti kvadrat duljine brida. Ovaj način izračuna prioriteta rezultira mrežama manje kvalitete od postupaka poput Garland-Heckbertovog, ali nudi mnogo bolje performanse. Ponovno treba paziti na to da uklanjamo isključivo jednostavne bridove.

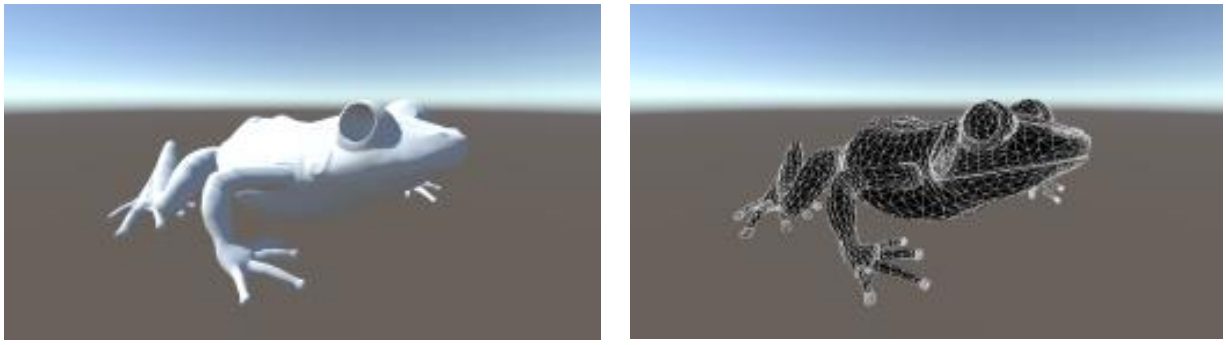
3.3.2. Uklanjanje brida

Nakon što smo odabrali brid koji želimo ukloniti moramo izračunati poziciju vrha koji će ga zamijeniti. Ovdje opet postoji mnogo opcija. Iz istih razloga kao i kod odabira novog vrha kod algoritma grupiranja vrhova koristiti ćemo prosjek vrhova brida kao novi vrh.

3.3.3. Prespajanje poligona

Ponovna izgradanja pripadnih poligona mnogo je jednostavnija nego kod uklanjanja vrhova. Dovoljno je u polju poligona oba uklonjena vrha zamijeniti novim i nakon toga ukloniti degenerirane poligone.

4. Rezultati



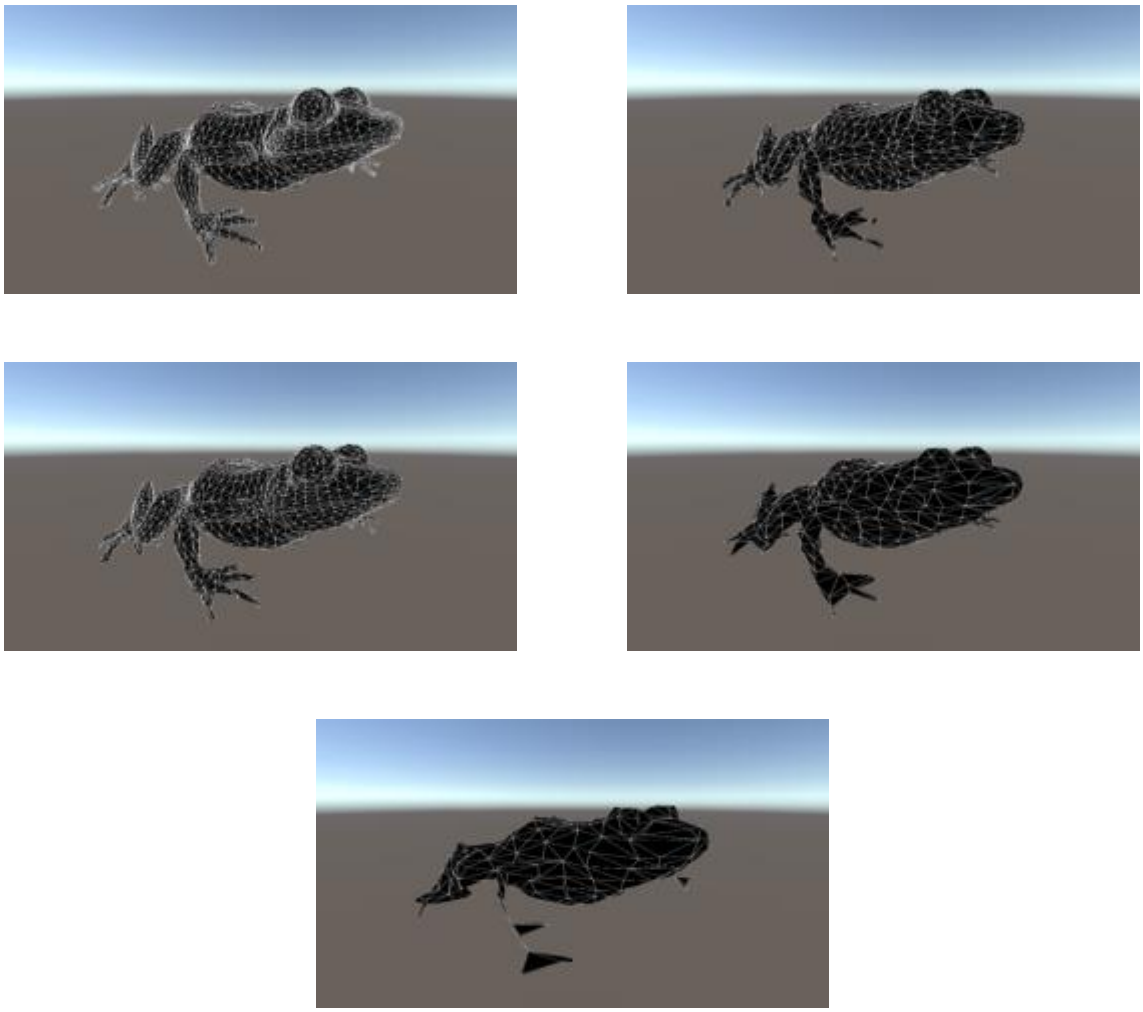
Sl 4.1 Jedan od modela nad kojim se vrše ispitivanja

Ispitivanja uspješnosti svakog od algoritama raditi ćemo generiranjem derivativnih modela više razina složenosti. Specifično uzeti ćemo uzorke na 75, 50, 25, 10 i 5 posto originalnog broja vrhova modela te ih ocijeniti na temelju topološke sličnosti s originalnom mrežom (

Sl 4.1) i količinom detalja koje su mreže manje složenosti uspjele očuvati. Uz to, algoritme ćemo također rangirati po vremenu izvođenja.

Potom ćemo u predgenerirani set objekata uvesti sustav razine detalja i generirati mreže niže složenosti za te objekte pa usporediti performanse scene bez korištenja sustava sa performansama na različitim razinama smanjene složenosti udaljenijih modela.

4.1. Modeli stvoreni algoritmom grupiranja vrhova



Sl 4.2 Modeli dobiveni algoritmom grupiranja vrhova, redom 75, 50, 25, 10 i 5 posto originalnog broja vrhova

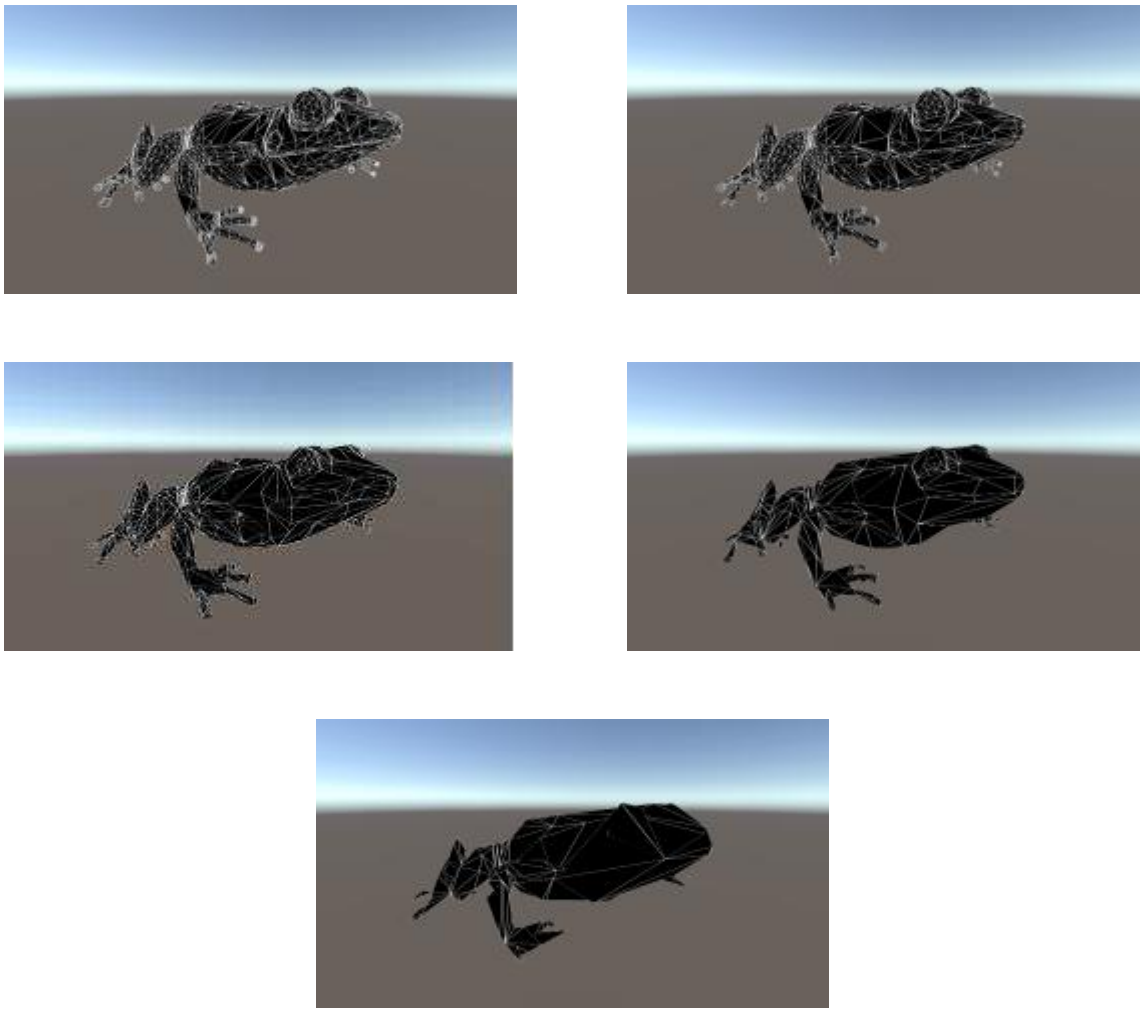
Algoritam grupiranja vrhova funkcionira prihvatljivo za pojednostavljivanja do 50% iako već tada možemo primijetiti grubo brisanje detalja (Sl 4.22) Do toga dolazi zbog forsiranja ravnomjernog rasporeda točaka. Algoritam postavlja točno jednu točku po kvadru u mreži neovisno o kompleksnosti tog dijela mreže. Primjer tog problema možemo vidjeti već na modelu generiranom sa 50% manjom složenosti ako se usredotočimo na žabine prste. Izgubili smo zadebljanja na

vrhovima prstiju. Cijela je grupa vrhova koja je tvorila zadebljanje pretvorena u svega njih nekoliko.

Dodatni problem je degeneracija segmenata mreže u 2D oblike pri većim smanjivanjima broja vrhova mreže. Ako pogledamo sliku gdje je kompleksnost smanjena za 95% možemo vidjeti kako su noge „nestale“. Ono što se zapravo dogodilo jest da su se pretvorile u dvodimenzionalne strukture. Razlog zašto se iz ovog kuta ne vide jest to što redoslijed vrhova poligona, ono što određuje u kojem smjeru gleda, računamo koristeći normale vrhova. Kako je noga reducirana na 2d plohu koriste se isti vrhovi za obje strane noge. Isti vrhovi uvijek će imati iste normale pa će stoga trokuti biti vidljivi samo iz jednog smjera, u ovom slučaju ne onog iz kojeg gledamo.

Velika prednost ovog algoritma je njegova brzina izvođenja. Algoritam sam po sebi već radi najbrže od svih isprobanih algoritama, a uz to povećanjem redukcije složenosti algoritam ubrzava. Naime ima manje kvadrova u koje dijeli prostor modela pa samim time i manje vrhova za koje mora generirati trokute što je operacijski daleko najskuplji dio ovog algoritma.

4.2. Modeli stvoreni algoritmom uklanjanja vrhova



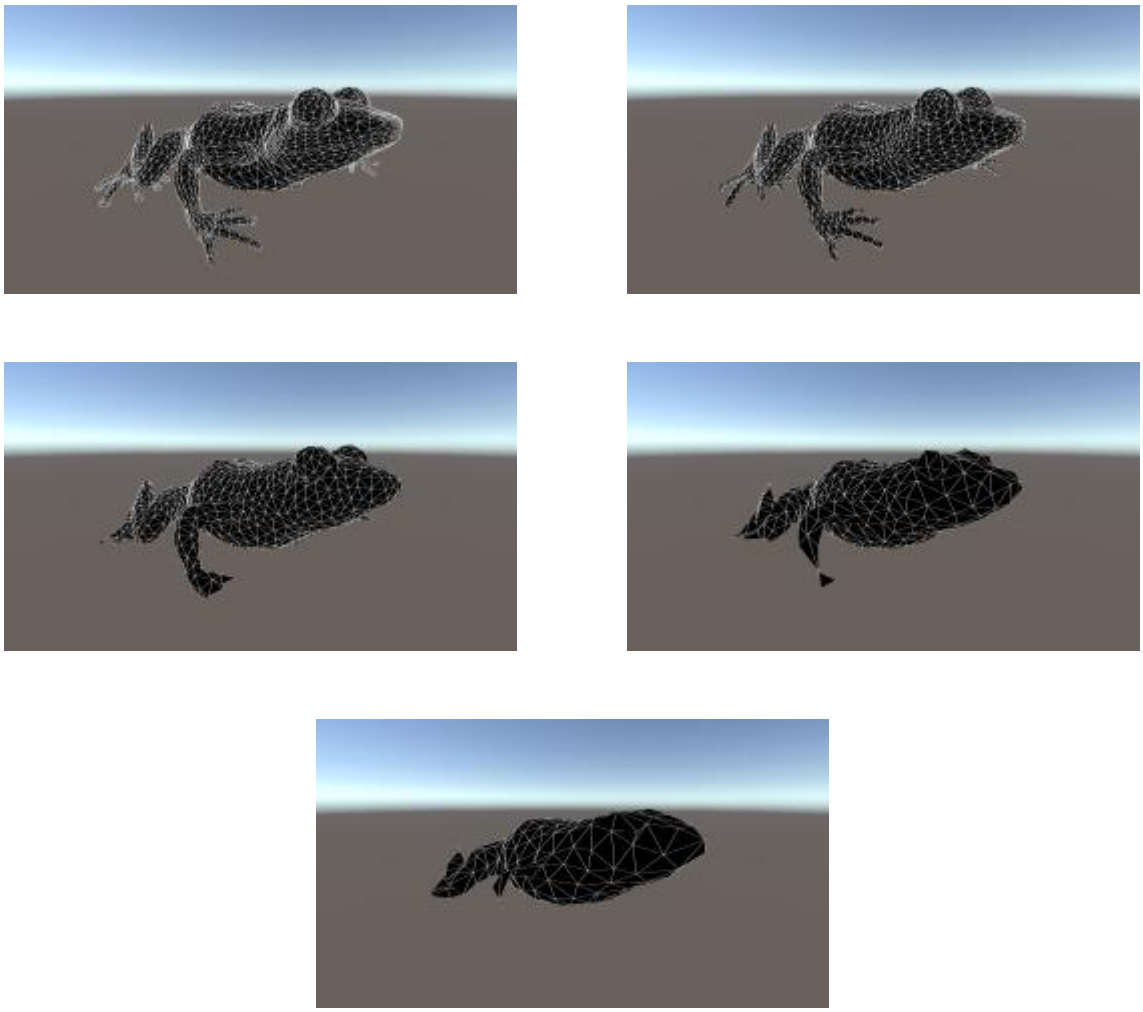
Sl 4.3 Modeli dobiveni algoritmom uklanjanja vrhova, redom 75, 50, 25, 10 i 5 posto originalnog broja vrhova

Algoritam uklanjanja vrhova ispao je najbolji za potrebe dinamičke generacije pojednostavljenih modela. Čak i modeli sa samo 5 posto originalnog broja vrhova (Sl 4.33) topološki dobro predstavljaju originalni model i mogu biti korišteni kao jako udaljeni elementi sustava razine detalja dok modeli od 10 posto originalnog broja vrhova uspijevaju zadržati većinu detalja originalnog modela.

Najveći problem ovog algoritma možemo uočiti usredotočimo li se na oči žabe. Kako su točke oko očne šupljine skoro u ravnini algoritam ih karakterizira kao jeftino uklonjive. Budući da ne pratimo svojstva bridova ne prepoznajemo da se točke nalaze na oštrim rubovima konkavnog dijela mreže i ne bi trebale biti uklonjene. To rezultira naglim smanjivanjem kvalitete unutrašnje strane očiju i malom rezolucijom rupe.

Algoritam u trenutnom stanju ima prihvatljivo vrijeme izvođenja. Dodatkom praćenja svojstava bridova koje bi riješilo problem spomenut u prošlom odlomku to vrijeme povećava na više od sekunde po mreži što u bilo kakvoj realnoj sceni nije prihvatljiva brojka.

4.3. Modeli stvoreni algoritmom uklanjanja bridova



Sl 4.4 Modeli dobiveni algoritmom uklanjanja bridova, redom 75, 50, 25, 10 i 5 posto originalnog broja vrhova

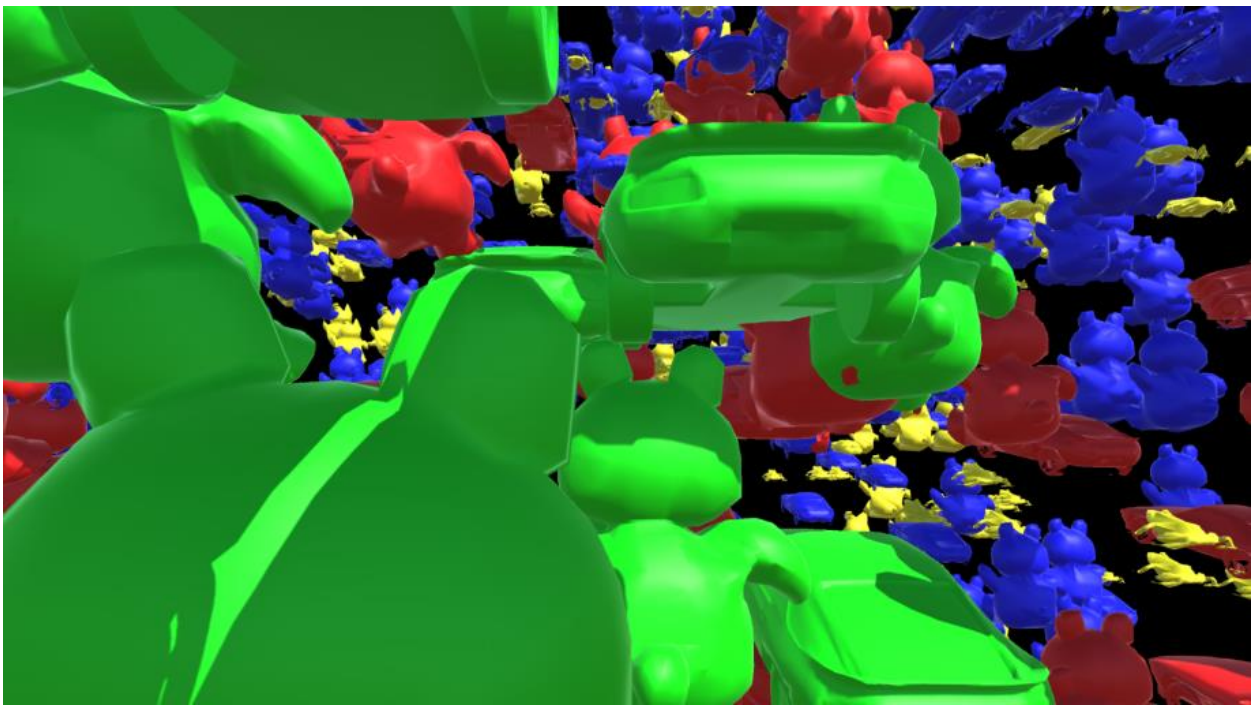
Algoritam uklanjanja bridova radi veoma dobro na višim razinama složenosti. Rezultati na 75 i 50 posto originalnog broja vrhova (Sl 4.4) mogu se ocijeniti kao najbolji do sad dobiveni. Za razliku od algoritma uklanjanja vrhova rezolucija očiju je još uvijek zadovoljavajuća dok su rezultati na ostalim detaljnijim dijelovima modela poput prstiju slične kvalitete. Rezultat na 25 posto vrhova

također je prihvatljiv no lošiji od rezultata dobivenom uklanjanjem vrhove jer je nedostatak prstiju na udaljenim objektima puno primjetljiviji od slabe rezolucije očiju.

Nažalost kvaliteta modela ubrzano opada pri prijelazu na manje brojeve vrhova. Uvjet duljine brida tu pokazuje svoju naivnost te počinje s eliminacijom ekstremiteta tako da na modelu s 10 posto originalnog broja vrhova gubimo dlanove i detalje u očima, a na modelu s 5 skoro pa ni nemamo prednje noge. Kada ne bismo bili ograničeni izvođenjem u skoro stvarnom vremenu mogli bismo uvesti dodatne zahtjeve pri biranju bridova za eliminaciju poput zabrane odabira rubnih bridova, no kao i u prošlom algoritmu zbog specifičnih zahtjeva projekta to nije moguće.

Algoritam se izvodi nešto brže nego algoritam uklanjanja vrhova no i dalje sporije od algoritma grupiranja vrhova.

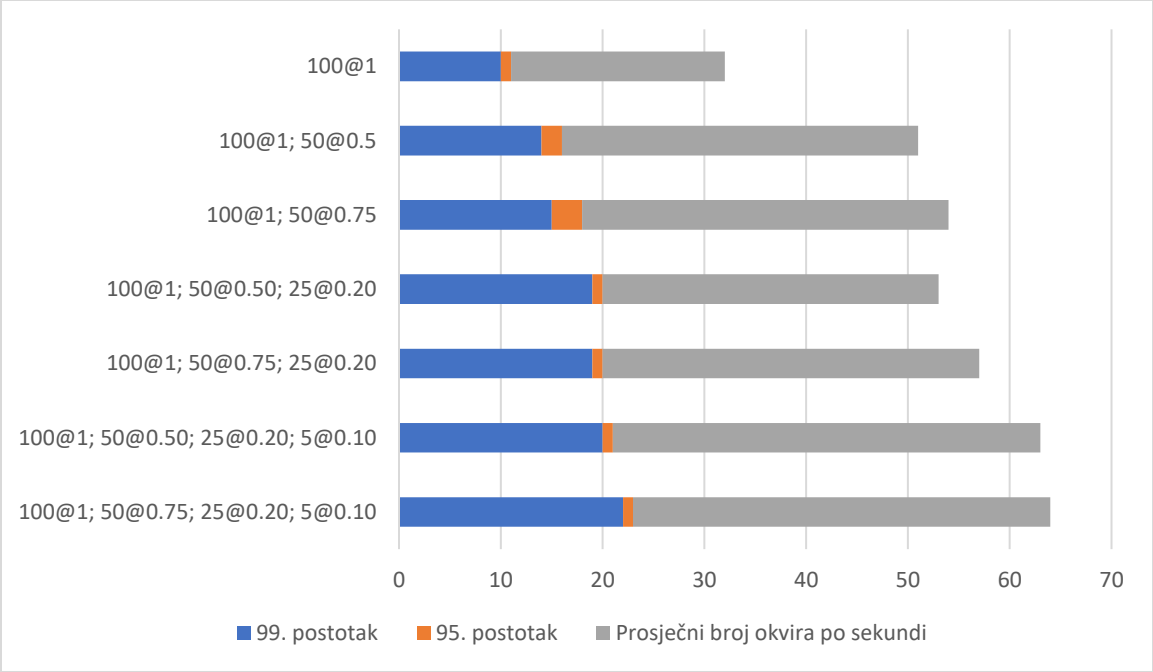
4.4. Utjecaj pojednostavljivanja objekata na performanse



Sl 4.5 Prikaz jednog okvira tijekom izvođenja testa

Specifikacije računala na kojem je test izveden su AMD Ryzen 7 4800HS procesor, Nvidia RTX 2060 6GB grafički procesor, 16GB radne memorije na frekvenciji 3.2GHz.

Testiranje performansi izvodimo na sceni s 1000 objekata (Sl 4.5). Objekti su mješavina instanci tri različita modela. Programski pomičemo kameru kroz prostor predodređenom putanjom te brojimo iscrtane okvire. Rezultat svakog testa je statistika o prosječnom broju okvira iscrtanih u sekundi, 95. postotak broja okvira iscrtanih u sekundi i 99. postotak broja okvira iscrtanih u sekundi. Za svaki set postavki testiranje je ponovljeno 20 puta. Odbačeno je 5 najboljih i 5 najgorih rezultata te je od preostalih 10 uzet prosjek. Ovo je napravljeno kako bi se izbjegle nasumične devijacije u rezultatima uzrokovane pozadinskim procesima na računalu na kojem je test izveden.



Sl 4.6 Iscrtani okviri ovisno o postavkama LOD-a

Na grafu (Sl 4.6) možemo vidjeti zabilježene performanse izražene u okvirima po sekundi ovisne o postavkama LOD-a. 99. odnosno 95. postotak je broj okvira iscrtanih u sekundi od kojeg je u 99 odnosno u 95 posto sekundi bilo iscrtano više okvira. Te nam mjere predstavljaju količinu trzanja koja se događa. Na vertikalnoj osi su izražene postavke LOD sustava u obliku <složenost modela u postotcima>@<omjer visine objekta na kameri i visine ekrana>. Na primjer 100@1; 50@0.75; 25@0.20 predstavlja slučaj kada od omjera visina 1 do omjera visina 0.75 iscrtavamo model pune kvalitete, od 0.75 do 0.2 mreža je pojednostavljena za 50 posto, a od 0.2 do 0 iscrtavamo mrežu pojednostavljenu na 25 posto vrhova originalne mreže. Podatci koje smo dobili podudaraju se s našim pretpostavkama. Što ranije postavljamo modele manje kvalitete to su performanse bolje. Jedan potencijalno neočekivan rezultat je činjenica da smo koristeći postavke 100@1; 50@0.75 dobili bolje rezultate nego koristeći 100@1; 50@0.5; 0.25@0.2. S obzirom na udio modela koji su na kameri iscrtani kao manji od petine visine ekrana očekivano je bilo da će, usporedno, 100@1; 50@0.5; 0.25@0.2 ostvari bolje performanse, no to nije slučaj. Postoji mogućnost da je to uzrokovano dodatnim troškovima baratanja trećom mrežom poligona kao dijelom sustava razine detalja.

Zaključak

Rezultati koje smo postigli pokazuju da je doista moguće dinamički algoritamski u skoro-stvarnom vremenu stvoriti mreže niže složenosti koji zadržavaju dovoljnu razinu kvalitete da se mogu koristiti kao komponente LOD sustava. Naravno, tako stvorene mreže neće parirati ručno pojednostavljenoj mreži od strane profesionalnog 3D dizajnera pa niti mrežama stvorenima *offline* algoritmima kojima je u redu iskoristiti koju sekundu više da bi postigli optimalniji rezultat, ali to se nije ni očekivalo.

Što se potencijalnih budućih poboljšanja tiče postoje dva glavna smjera u kojima bi mogli krenuti. Prvo, kada bismo uspjeli stvoriti strukturu podataka koja može efikasnije spremati i ažurirati podatke potrebne Garland – Heckbertovu algoritmu mogli bi njegovom implementacijom drastično poboljšati kvalitetu pojednostavljenih mreža. Druga opcija koja bi se mogla istražiti je sastav same ulazne mreže. Ovaj je rad pisan pod pretpostavkom da smo kao ulazne podatke dobili osnovne podatke o mreži. Vrhovi, poligoni, potencijalno normale i druge stvari navedene u prvom poglavlju. Kada bismo uz te podatke u ulazne mreže zapisali još i podatke o oštrim rubovima, unutarnjim rupama, izbočenjima i drugim pojavama čije se segmente ne bi trebalo značajno mijenjati mogli bismo to znanje primijeniti i potencijalno eliminirati probleme na koje smo naišli s očima u algoritmu uklanjanja vrhova i s nogama u algoritmu uklanjanja bridova.

Literatura

[1] Mesh, Unity dokumentacija. https://docs.unity3d.com/ScriptReference/Mesh.html: pristupljeno 3. veljače 2022.

[2] LODGroup, Unity dokumentacija, https://docs.huihoo.com/unity/5.5/Documentation/ScriptReference/LODGroup.html: pristupljeno 3. veljače 2022.

[3] LOD, Unity dokumentacija, https://docs.huihoo.com/unity/5.5/Documentation/ScriptReference/LOD.html: pristupljeno 3. veljače 2022.

[4] Jarek Rossignac, Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. Modeling in Computer Graphics: Methods and Applications (1993.), str. 455–465.

[5] Schroeder, W. J., Zarge, J. A., & Lorensen, W. E., Decimation of triangle meshes. Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques – SIGGRAPH, Chicago, 1992.

[6] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques – SIGGRAPH, New York, 1993.

[7] Hugues Hoppe. Progressive meshes. Proceedings of the 23th Annual Conference on Computer Graphics and Interactive Techniques – SIGGRAPH, New Orleans, 1996.

[8] Michael Garland, Paul S. Heckbert, Surface Simplification Using Quadric Error Metrics. Annual Conference on Computer Graphics and Interactive Techniques – SIGGRAPH, Los Angeles, 1997.

Sažetak

Ovaj rad bavi se sustavom razine detalja i različitim tehnikama smanjivanja kompleksnosti mreža koje bi u njemu mogli koristiti. U radu su isprobani algoritmi grupiranja vrhova, uklanjanja vrhova i uklanjanja bridova. Ocjenjene su pojednostavljene mreže i učinkovitost rada sustava razine detalja. Sva programska podrška implementirana je u programskom jeziku C# uz potporu Unity programskog alata.

Summary

This paper examines level of details systems as well as several techniques dealing with reducing the complexity of meshes to be used with it. The algorithms used are vertex clustering, vertex decimation and edge decimation. We rate the resulting meshes and the improvements in efficiency due to the usage of the level of detail system. All software used in this paper was written in C# using the Unity game engine for help with rendering and mathematical operations.