SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1039

POSTUPCI GENERIRANJA MODELA KRAJOLIKA

Niko Mikuličić

Zagreb, lipanj 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 2. ožujka 2015.

Predmet: **Diplomski rad**

DIPLOMSKI ZADATAK br. 1039

Pristupnik: **Niko Mikuličić (0036461325)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Postupci generiranja modela krajolika**

Opis zadatka:

Proučiti karakteristike krajolika kao što su konfiguracija tla i vegetacijski pokrov. Proučiti postupke generiranja vegetacijskog pokrova a posebno obratiti pažnju na proceduralne tehnike. Razraditi aplikaciju koja će omogućiti stvaranje modela konfiguracije tla i pripadnog vegetacijskog pokrova za odabrano vegetacijsko područje. Po mogućnosti obraditi područje Mediterana. Razraditi utjecaj parametara kao što su količina padalina i godišnje doba u izradi krajolika.
Izraditi odgovarajući programski proizvod. Koristiti programski jezik C# i grafičko programsko sučelje OpenGL. 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.

Zadatak uručen pristupniku: 2. ožujka 2015.
Rok za predaju rada: 30. lipnja 2015.

Mentor:

Prof. dr. sc. Željka Mihajlović

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Sadržaj

1. Uvod

Realističan prikaz krajolika oduvijek je bio izazov u računalnoj grafici prvenstveno zbog količine podataka, ali i zbog inherentne kompleksnosti prirodnih pojava koje u stvarnosti podrazumijevamo, a u grafici nije lako reproducirati. Otvoreni krajolici vrlo detaljnog terena s bujnom i realističnom vegetacijom te pogleda koji seže do obzora tek su nedavno postali ostvarivi u stvarnom vremenu. U prikazu krajolika, realizam je mnogo više od realističnih 3D modela. Kompleksna interakcija svjetla i sjene svake pojedine travke, kamena i stabla značajno utječe na način koliko realno scena izgleda. Udaljena šumovita planina ima vrlo definiran volumen upravo zbog igre svjetla i sjene svakog stabla koje se na njoj nalazi. Vlažan šumski zrak ima vizualnu reprezentaciju raspršene svjetlosti i volumnog osvjetljenja. Pored toga, treba imati na umu da je riječ o jako velikim prostorima, dakle o jako mnogo objekata koji istovremeno moraju biti realistični i skromni po pitanju računalnih resursa. Iz svih tih razloga, područje prikaza krajolika vrlo je opsežno i spaja mnoga dostignuća računalne grafike.

Krajolici su svoju primjenu našli ponajviše u filmskoj industriji, industriji igara te u vojnim simulatorima. Realistični krajolici prvu su primjenu našli u filmskoj industriji za prikaz izmišljenih realističnih ili animiranih krajolika budući da se scena nije morala prikazivati u stvarnom vremenu. S napretkom računalne moći, danas su vrlo realistični krajolici široko prisutni i u računalnim igrama.

Cilj ovog rada je ostvariti programsku podršku za proceduralno generiranje vjerodostojnih prirodnih krajolika. Pokazat će se da se uz osnovni skup parametara te ulazne modele i teksture mogu vrlo brzo stvoriti vjerodostojni prirodni krajolici različitih klima, vegetacijskih zona i u različitim godišnjim dobima.

Osnovne komponente u vizualnom prikazu krajolika su teren i vegetacija. U poglavlju 2 razmatramo tehnike stvaranja geometrije terena i njegovog teksturiranja. Predstavljene su različite metode dobivanja visinske mape te je dan pregled različitih postupaka razina detalja specijaliziranih za teren uz detaljni osvrt na tehniku implementiranu u ovom radu. Poglavlje završava opisom različitih metoda preslikavanja tekstura na teren i njihovog proceduralnog generiranja. Nakon toga, u poglavlju 3, opisani su postupci prikaza vegetacijskog pokrova i njegovog prostornog razmještaja na terenu. Predstavljeno je učinkovito generiranje trave te su detaljnije objašnjeni različiti algoritmi za prostorno

raspoređivanje kamenja i više vegetacije. U poglavlju 4 slijedi detaljni opis implementacije opisanih postupaka čiji su rezultati potom prikazani u poglavlju 5.

2. Teren

Osnovni izgled svakog krajolika definiran je njegovim reljefom. Reljef se tipično prikazuje elevacijskim podatkom u svakoj njegovoj točki. Elevacijski podatci dobiveni satelitskim skeniranjem tipično se pohranjuju u DEM (engl. *Digital Elevation Map*) ili TIN (engl. *Triangular Irregular Network*) formatu dok se pri stvaranju 3D modela terena takvi podatci najčešće zapisuju u visinsku mapu.

2.1. Visinska mapa

Visinska mapa (engl. *heightmap*) je tekstura u kojoj svaki piksel predstavlja informaciju o nadmorskoj visini. Budući da je svaki piksel samo jedan podatak nadmorske visine, jasno je da se takvim zapisom ne mogu prikazati strukture poput nagnutih klisura jer jedan podatak o nadmorskoj visini preuzet s vrha klisure ne sadrži informaciju o oblicima koji su sakriveni ispod.

Visinska mapa je tipično siva slika, odnosno sve tri komponente boje sadrže isti podatak o nadmorskoj visini. Ako je komponenta boje predstavljena s 8 bitova, to znači da takvim zapisom možemo imati samo 256 nadmorskih visina. To je uglavnom dovoljno ako visinska mapa predstavlja manje područje unutar kojeg nema velikih razlika u nadmorskoj visini. Ako je potrebna veća razlučivost, u crvenu komponentu teksture mogu se zapisati podatci koji predstavljaju osnovnu topografiju, a primjerice zelena komponenta može sadržavati relativne pomake od crvene komponente. Plava komponenta slike kao još jedna razina razlučivosti uglavnom nije potrebna.

Glavna prednost visinske mape nad drugim formatima je mogućnost da se pohrani u memoriju grafičke procesorske jedinice. To je posebno korisno za tehnike koje geometriju terena stvaraju dinamički u procesoru geometrije (engl. *geometry shader*) i pritom koriste tehnike kontinuiranih razina detalja da bi broj poligona bio što manji.

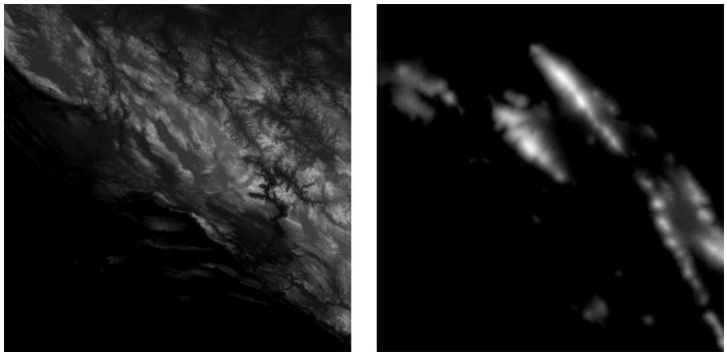
2.1.1. Podatci iz stvarnog svijeta

Na internetu postoje baze elevacijskih podataka gotovo cijele zemaljske kugle. DEM podatci agencije U.S. Geological Survey (USGS) dostupni su na stranici http://earthexplorer.usgs.gov/ gdje se mogu preuzeti elevacijski podatci u niskoj i srednjoj

razlučivosti (do preciznosti od 1 kutne sekunde). Postoje podatci i u većoj razlučivosti (do 1/3 kutne sekunde), ali su najčešće komercijalni ili ograničeni za vojnu uporabu. Detaljne upute kako preuzeti podatke kao i pregled mnogih tehnika i znanstvenih radova u području generiranja terena mogu se pronaći na stranici www.vterrain.org.

Postoje i mnogi programski alati za manipulaciju geografskim podatcima. Jedan od besplatnih je MICRODEM koji se može besplatno preuzeti na stranici http://www.usna.edu/Users/oceano/pguth/website/microdem/microdem.htm.

Slika 1 prikazuje visinske mape Dalmacije i dijela Bosne i Hercegovine te okolicu otoka Ista pored Zadra. Riječ je o USGS DEM podatcima u razlučivosti od 1 kutne sekunde koji su pomoću alata MICRODEM iz DEM formata pretvoreni u sivu sliku.



Slika 1: Visinske mape dobivene iz USGS podataka Dalmacije i dijela Bosne i Hercegovine (lijevo) te okolice otoka Ista (desno)

2.1.2. Proceduralno generiranje visinske mape

Ako želimo prikazati izmišljene krajolike, prvo je potrebno generirati vlastitu visinsku mapu. U nastavku ukratko opisujemo neke od brojnih algoritama za njeno proceduralno generiranje.

2.1.2.1 Fraktalni algoritmi

Fraktalni algoritmi temelje se na ideji statističke samosličnosti. Rekurzivnim ponavljanjem geometrijskog pomaka može se postići fraktalni izgled terena, odnosno da njegov najmanji dio ima sličnost s cjelinom. Ilustracija osnovne ideje može se prikazati ovako: neka algoritam započne s jednim kvadratom, odnosno s četiri točke. U centar kvadrata postavi se nova točka i pomakne za određeni pomak. Zatim se kvadrat podijeli na četiri manja kvadrata za koje se rekurzivno ponavlja postupak pritom smanjujući iznos pomaka sa svakom dubinom rekurzije. Predstavnici ovih algoritama su algoritam pomaka srednje

točke (engl. *Midpoint Displacement*) te algoritam podjele dijamant-kvadrat (engl. *Diamond-Square Subdivision*) koji je nastao kao njegovo poboljšanje (Miller, 1986).

2.1.2.2 Šum

Šum se u računalnoj grafici često koristi za stvaranje kontrolirane nasumičnosti i mogućnosti njegove primjene su praktični neograničene. Najčešće se koristi za generiranje terena, oblaka, vatre, dima, detalja na teksturama, za razbijanje ponavljajućih uzoraka na teksturi i sl.

Šum se razvio kao posebno područje jer potpuna nasumičnost, čiju aproksimaciju možemo dobiti običnim generatorom pseudoslučajnih brojeva, nije poželjna. Ekvivalent potpune nasumičnosti bi bio bijeli šum čija je energija jednoliko distribuirana na svim frekvencijama. Uzorkovanje teksture bijelog šuma bilo bi podložno aliasingu upravo zbog prisutnosti visokih frekvencija pa je bilo potrebno smisliti mehanizme za kontrolu nasumičnosti.

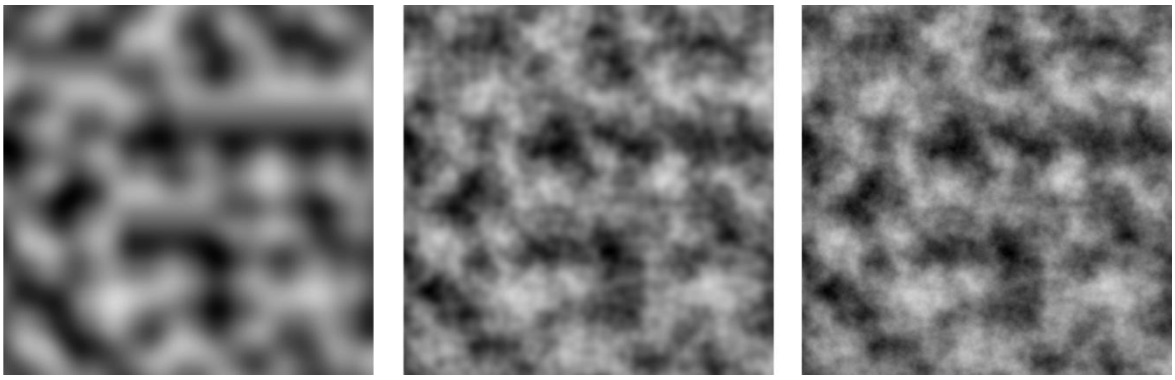
Postoje različite vrste šuma, ali najčešće se koriste varijante šuma na rešetci (engl. *Lattice Noise*) koji nasumičnu vrijednost dodjeljuje točkama s cjelobrojnim koordinatama. Vrijednosni šum (engl. *Value* Noise) svim točkama rešetke dodjeljuje nasumične vrijednosti koje se potom interpoliraju što ima efekt niskopropusnog filtra. Pritom je vrlo bitno koja se interpolacijska funkcija koristi. Primjerice, linearna interpolacija uzrokuje vidljive blokove na slici i vrlo je jasno da se u pozadini algoritma nalazi rešetka. Za vrijednosni šum najčešće se koristi kubna interpolacija kojom takvi efekti nisu uočljivi (Ebert, 2003). Gradijentni šum (engl. *Gradient Noise*) umjesto konkretne vrijednosti, svakoj točki rešetke dodjeljuje nasumičan vektor smjera iz kojih se poslije računa vrijednost i potom interpolira. U praksi se za generiranje visinske mape najčešće koriste Perlinov šum (engl. *Perlin Noise*) i Simpleks šum (engl. *Simplex Noise)* kao njegovo poboljšanje.

Šum generiran spomenutim algoritmima nije fraktalan. Da bi konačna visinska mapa imala svojstvo samosličnosti koristi se frakcijsko, odnosno fraktalno Brownovo gibanje (engl. *fractional/fractal Brownian motion, fBm)* koje se može zapisati kao (Dachsbacher, 2006):

$$fBm(\boldsymbol{x}) = \sum_{i=0}^{n-1} G^i N(\boldsymbol{x} \cdot L^i) \tag{1}$$

gdje je $N(\boldsymbol{x})$ vrijednost funkcije šuma za točku $\boldsymbol{x}$, n broj oktava, L razmak u frekvenciji između dvije susjedne oktave (engl. *lacunarity*) te G faktor smanjenja amplitude (engl. *gain*). Riječ je o iterativnom sumiranju šuma s njegovim oktavama gdje svaka sljedeća oktava ima višu frekvenciju i manju amplitudu. Time se postiže samosličnost dijelova s cjelinom i veća oštrina slike. Šum s vrijednostima parametara $L = 2.0$ i $G = 0.5$ zove se $1/f$ šum i predstavlja industrijski standard (Archer, 2011).

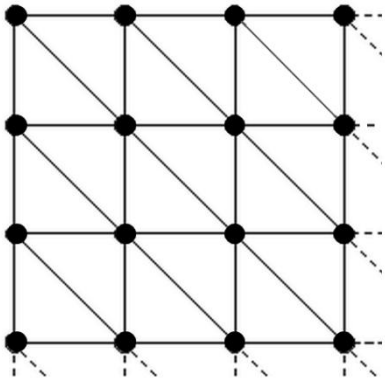
Slika 2 prikazuje Perlinov šum i generirane fraktalne šumove za različiti broj oktava. Korištene vrijednosti parametara su upravo vrijednosti navedenog industrijskog standarda. Može se primijetiti da fraktalni šum zadržava osnovni oblik Perlinovog šuma budući da je nulta oktava upravo jednaka Perlinovom šumu, ali sa svakom sljedećom oktavom dodaje se količina detalja i povećava oštrina slike. Tako generirana tekstura može poslužiti kao visinska mapa za generiranje proizvoljnih terena ili kao mapa detalja koja će postojećoj visinskoj mapi dodati raznolikost.



Slika 2: Perlinov šum (lijevo) te fraktalni šum s brojem oktava n = 3 (sredina) i n = 6 (desno)

2.2. Mreža trokuta

Nakon što imamo visinsku mapu potrebno je iz nje ostvariti trodimenzionalni model terena, odnosno mrežu trokuta (engl. *mesh*). Najjednostavniji način bi bio kreirati jedan vrh za svaki piksel visinske mape i dijagonalno povezati trokute kao na slici 3. No, za visinsku mapu veličine 512x512 piksela taj pristup bi stvorio preko 250 tisuća vrhova i preko 500 tisuća trokuta. Iako to nije nikakav problem za današnje grafičke procesorske jedinice, činjenica je da je takav pristup vrlo neefikasan i problemi nastaju vrlo brzo ako želimo prikazati veća područja ili koristiti veće visinske mape.



Slika 3: Naivni pristup generiranja mreže trokuta iz visinske mape

Bitno je primijetiti da dio terena koji se nalazi u daljini nije potrebno prikazati sa svim detaljima jer se ti detalji ionako ne vide. Moguće bi bilo udaljenim područjima značajno smanjiti broj poligona kojima se iscrtavaju bez da to utječe na vizualni dojam scene. Također, područja koja su gotovo ravna mogu biti prikazana s puno manje poligona bez unošenja greške. S tim idejama razvijeni su brojni postupci razina detalja.

2.2.1. Razine detalja

Postupci razina detalja (engl. *Level of Detail,* LOD) nastoje ostvariti optimizaciju broja poligona s kojima se teren prikazuje bez da se smanji količina detalja koju je moguće percipirati. Razine detalja mogu se podijeliti na diskretne i kontinuirane.

Diskretne, odnosno statičke razine detalja (engl. *Static Level of Detail*, SLOD) generiraju se prije izvođenja programa i ne mijenjaju se tijekom izvođenja. Takav pristup smanjuje računske zahtjeve, ali povećava memorijske budući da se informacije o svakoj razini detalja moraju prethodno pohraniti. Teren se tipično simplificira u nekoliko razina i zatim podijeli u zone kako bi se omogućilo jednostavnije odbacivanje poligona po projekcijskom volumenu (engl. *view frustum culling*). Odluka koja razina detalja se koristi donosi se nad cijelom zonom. Ako se za zonu odredi da je dovoljno daleko od promatrača ili da je njena projekcija na ekran dovoljno mala, zona se prikazuje u nižoj razini detalja. Takav pristup ima nekoliko problema. Prvi problem je efekt iskakanja (engl. *popping*). Prilikom mijenjanja iz jedne razine detalja u drugu desi se nagli prijelaz za veliki broj vrhova što ljudsko oko može primijetiti. To se tipično rješava interpolacijom vrhova (engl. *vertex morphing*) gdje se kroz neko tranzicijsko područje vrhovi iz jedne razine detalja postepeno translatiraju prema vrhovima sljedeće razine detalja dok se ne preklope (Ulrich, 2002). Takvim pristupom moguće je u potpunosti riješiti efekt iskakanja. Drugi problem ovakvog pristupa su pukotine u terenu na granicama između dvije zone koje su prikazane u

različitim razinama detalja. Za taj problem nema elegantnog rješenja već se svi pristupi uglavnom zasnivaju na stavljanju zakrpi između dvije zone. Predložene su različite vrste zakrpi (Ulrich, 2002) s različitim svojstvima te se pokazalo da mogu biti neprimjetne ako razlike između razina detalja nisu prevelike. Diskretne razine detalja su brze i jednostavne, a glavni nedostatak ovih tehnika ostaje iscrpljujuća pripremna faza koja je dosta rigidna i specifična za aplikaciju. Algoritama diskretnih razina detalja nema mnogo i baziraju se na vrlo sličnim principima. Jedan od popularnijih je *ChunkedLOD* (Ulrich, 2002).

Kontinuirane razine detalja (engl. *Continuous Level of Detail*, CLOD) mijenjaju se dinamički prilikom svakog prolaza kroz grafički protočni sustav. To omogućuje neograničen broj razina detalja bez potrebe za memorijskim prostorom da se pohrane. Budući da i ovdje želimo omogućiti jednostavno odbacivanje poligona po projekcijskom volumenu, teren se obično dijeli u zone što nosi iste probleme pukotina kao i kod diskretnih razina detalja. Za rješavanje problema pukotina koriste se slične tehnike, ali su razvijeni i neki algoritmi koji ne stvaraju pukotine i stoga nemaju potrebe za zakrpama (Strugar, 2009). Loša strana ovih metoda je što su prilično kompleksne i računski dosta zahtjevne te mogu imati značajan utjecaj na brzinu iscrtavanja. Unatoč tome, većina razvijenih tehnika baziraju se na kontinuiranim razinama detalja upravo zbog fleksibilnosti i skalabilnosti bez potrebe za pripremnom fazom. Razvijene su mnoge metode kontinuiranih razina detalja od kojih bismo mogli izdvojiti *Progressive meshes* (Hoppe, 1996), ROAM (Duchaineau, 1997) i SOAR (Lindstrom, 1996, 2001, 2002) koji su postavili temelje za algoritme koji su slijedili, *GeoMipMapping* (de Boer, 2000) čija glavna prednost je jednostavnost implementacije te CDLOD (Strugar, 2009) kao novija i naprednija tehnika koja ne stvara pukotine između razina detalja.

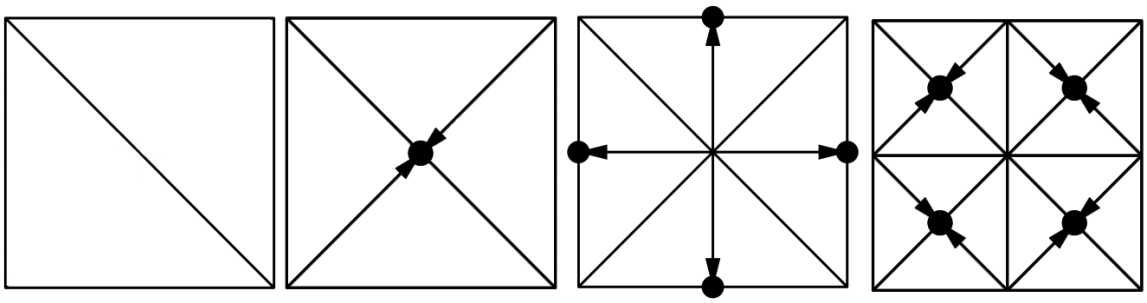
2.2.2. Lindstrom-Koller metoda simplifikacije

Lindstrom-Koller metoda simplifikacije terena predložena je kao sastavni dio SOAR algoritma (Lindstrom, 1996, 2001, 2002). Osim za SOAR algoritam, metoda je našla primjene i u mnogim drugim CLOD algoritmima, ali i kao metoda generiranja statičkih, diskretnih razina detalja. U ovom radu navedena metoda je korištena za generiranje statičke mreže trokuta s optimiziranim brojem poligona.

Algoritam radi s visinskom mapom veličine $(2^n + 1)$ x $(2^n + 1)$ piksela i bazira se na rekurzivnoj podjeli najduljeg brida trokuta (engl. *longest edge bisection*). Temeljem

određenih pravila, za svaku točku visinske mape određuje se je li aktivna. Aktivne točke se na kraju algoritma koriste za formiranje konačne, simplificirane mreže trokuta.

Algoritam započinje kvadratom tj. s dva pravokutna trokuta (slika 4). Svakom trokutu se na polovištu hipotenuze odredi nova točka i taj postupak se ponavlja dok se ne dođe do najdublje razine. Smjer strelice označava relaciju roditelj-dijete. Možemo primijetiti da će svaka točka koja nije rubna imati četvero djece i dva roditelja.



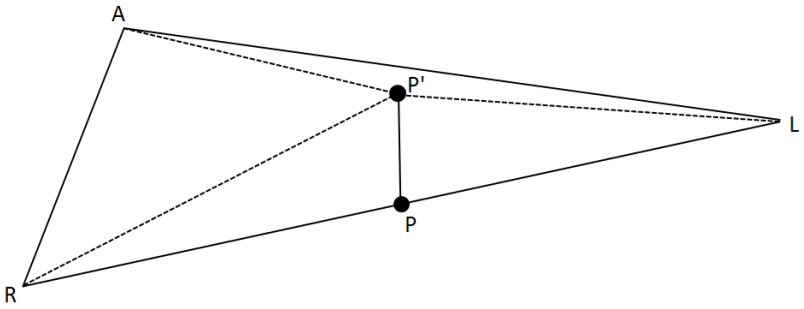
Slika 4: Rekurzivna podjela hipotenuze trokuta (desne dvije slike preuzete iz Lindstrom, 2002)

Kad smo se spustili do najdublje razine, potrebno je za svaku točku provjeriti kriterij aktivacije, odnosno hoće li se točka pojaviti u konačnoj mreži trokuta ili ne. Provjeru vršimo od dolje prema gore, odnosno jednom kad završimo s trenutnom razinom, nastavljamo provjeru na razini iznad. Početne četiri točke na najvišoj razini uvijek su aktivne.

Da bismo definirali kriterij aktivacije prvo je potrebno uvesti mjerilo za veličinu pogreške koju unosimo ako neki vrh izostavimo iz mreže trokuta. Slika 5 prikazuje postupak izračunavanja pogreške. Točka P je polovište dužine $\overline{LR}$ trokuta $\triangle ARL$. Točka P' je stvarna pozicija točke P ako ju uključimo u konačnu mrežu trokuta. Grešku koju unosimo ako točku P ne uključimo u konačnu mrežu trokuta možemo definirati kao maksimalnu vertikalnu udaljenost između trokuta $\triangle ALR$ i para trokuta $(\triangle ARP', \triangle AP'L)$ (Lindstrom, 1996):

$$\delta = \left| P_z + \frac{L_z + R_z}{2} \right|, \tag{2}$$

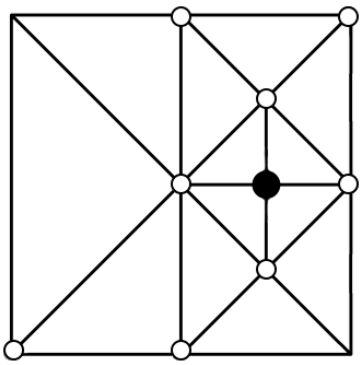
gdje indeks z označava elevaciju očitanu iz visinske mape za pojedinu točku. Za neku zadanu dozvoljenu grešku τ, možemo reći da je točku P potrebno aktivirati ako vrijedi $\delta > \tau$, odnosno ako njenim izostankom unosimo veću grešku nego što je dozvoljeno.



Slika 5: Pogreška koja se unosi ako se točka P izostavi iz konačne mreže trokuta

Već smo ustanovili da sve točke osim rubnih imaju po dva roditelja što znači da se rekurzivnim raspolavljanjem najduljeg brida do njih može doći na dva načina. Moguće je doći do točke P s jedne strane i zaključiti da ju nije potrebno aktivirati te potom doći s druge strane i zaključiti da ju je unutar tog trokuta potrebno aktivirati. Ako točku aktiviramo samo s jedne strane dolazi do T-spoja (engl. *T-junction*) koji uzrokuje pukotinu u mreži trokuta. Nazovimo A' točku koja se nalazi na pravom kutu trokuta koji s trokutom $\triangle ARL$ dijeli hipotenuzu $\overline{RL}$. Pukotina će nastati jer je s jedne strane stvoren trokut $\triangle A'LR$ (jer se zaključilo da točka P nije potrebna), a s druge strane trokuti $\triangle ARP'$ i $\triangle AP'L$ (jer se zaključilo da je točka P potrebna). Pukotina će zapravo biti nepostojeći trokut $\triangle RLP'$.

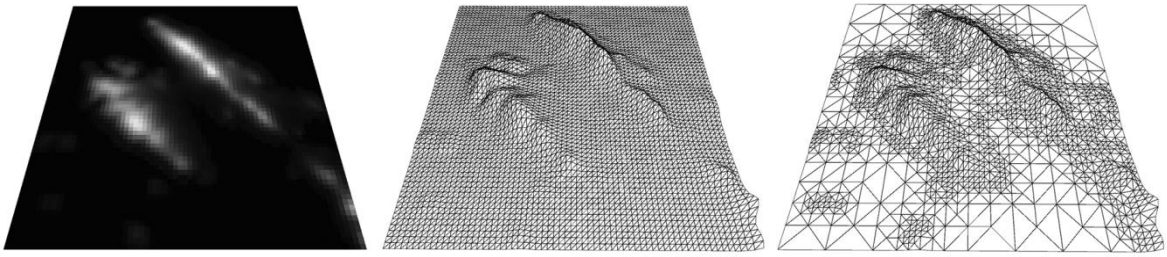
Da bismo osigurali nepostojanje pukotina u konačnoj mreži trokuta, prilikom aktivacije pojedine točke potrebno je također aktivirati i sve njene prethodnike. Tako se osigurava da je točka aktivirana s obje strane i ne nastaju T-spojevi. Slika 6 pokazuje punu točku koju je potrebno aktivirati i prazne točke koje su njeni prethodnici. Ako aktiviramo punu točku, tada moramo aktivirati i sve prazne.



Slika 6: Aktivacija pune točke povlači aktivaciju praznih točaka

Nakon što je za svaki vrh određeno je li aktivan ili nije, iz aktivnih vrhova stvara se mreža trokuta. Slika 6 istovremeno prikazuje i konačni izgled mreže trokuta ako su aktivne samo označene točke.

Rezultati korištenja ove tehnike prikazani su na slici 7. Lijevo je prikazana visinska mapa veličine 65x65 piksela za koju stvaramo mrežu trokuta, u sredini se nalazi mreža trokuta generirana naivnom metodom opisanom na početku poglavlja 2.2, a desno mreža trokuta simplificirana metodom Lindstrom-Koller. Može se primijetiti da metoda Lindstrom-Koller koristi veće trokute tamo gdje je površina relativno ravna, a ostavlja manje trokute tamo gdje površina sadrži detalje. Na taj način mogu se postići značajne uštede bez smanjenja kvalitete. Budući da navedena tehnika stvara mrežu trokuta ovisno o dozvoljenoj grešci, vrlo lako je stvoriti diskretne razine detalja na način da svaka sljedeća razina dozvoljava veću grešku. Već pri malim iznosima greške, uštede su velike, a razlika s originalom neprimjetna.



Slika 7: Visinska mapa (lijevo) i mreže trokuta generirane naivnom metodom (sredina) i simplifikacijom Lindstrom-Koller (desno)

2.3. Teksturiranje

Nakon što smo ostvarili trodimenzionalni model terena, možemo teksturama u njega unijeti život. Teksturne koordinate vrhova lako se izračunaju planarnom projekcijom koordinata tekstura iz pozitivne y-osi na mrežu trokuta. Jednostavnije govoreći, vrh koji se nalazi u donjem lijevom kutu dobit će teksturne koordinate (0,0), a vrh u gornjem desnom kutu koordinate (1,1). Ostalim vrhovima se teksturne koordinate računaju linearnom interpolacijom s obzirom na to gdje se između donjeg lijevog i gornjeg desnog kuta nalaze.

Na velikim nagibima terena, planarnom projekcijom će dva susjedna vrha dobiti vrlo bliske teksturne koordinate što za posljedicu ima rastezanje teksture. Taj problem može se riješiti troplanarnom projekcijom gdje se projekcija ne obavlja nužno iz smjera pozitivne y-osi već iz onog smjera iz kojeg dominira normala na površinu. Taj postupak je računski skuplji, ali daje bolje rezultate.

2.3.1. Tekstura boje

Najjednostavniji postupak teksturiranja terena je korištenje jedne teksture boje (engl. *color map*) koja se preslika na čitavu površinu terena. Takav postupak je vrlo brz i

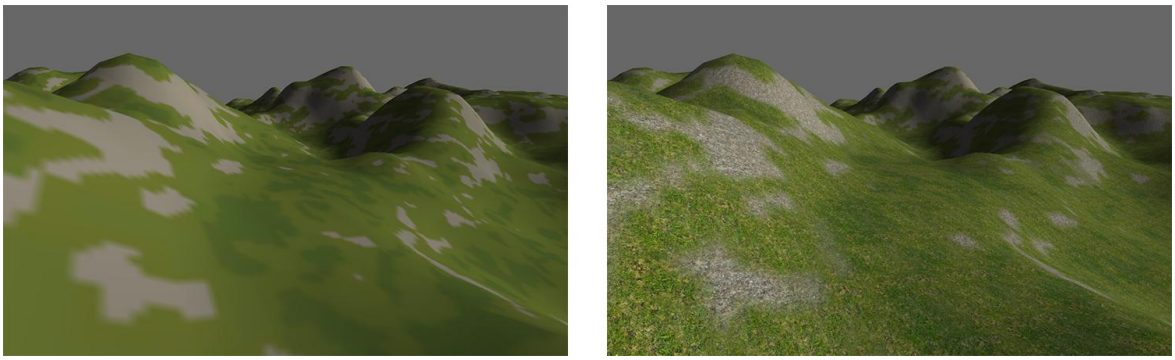
dovoljno dobar ako se teren gleda iz velike udaljenosti. Približavanjem površini terena, tekstura će postajati monotona i mutna. Mogli bismo probati povećati veličinu teksture boje s nadom da će veća tekstura imati dovoljnu količinu detalja, ali ispostavlja se da bismo vrlo brzo dosegnuli memorijska ograničenja grafičke kartice, a još uvijek imali vrlo monotonu površinu terena.

2.3.2. Miješanje tekstura detalja

Miješanje tekstura detalja (engl. *texture splatting*) je tehnika kojom se rješava problem monotonih površina. Umjesto jedne velike teksture boje za cijeli teren, koristi se više manjih i detaljnijih tekstura koje se preslikavaju na manji dio terena i potom ponavljaju po ostatku (engl. *tiling*). Uvodi se još jedna tekstura težina (engl. *splat map*) koja se preslikava na cijeli teren i opisuje s kolikim udjelima se teksture detalja miješaju. Svaki piksel mape težina sadrži četiri vrijednosti (RGBA) gdje svaka vrijednost nosi informaciju o udjelu kojeg ima pojedina tekstura detalja. Prilikom određivanja boje fragmenta terena, uzorkuju se sve teksture detalja i tekstura težina. Zatim se boje uzorkovane iz tekstura detalja miješaju u skladu s udjelima uzorkovanim iz teksture težina i dobivena boja postaje osnovna boja fragmenta.

Jedna tekstura težina može pohraniti težine za četiri teksture detalja što je uglavnom dovoljno. Naravno, uvijek je moguće koristiti još jednu mapu težina za četiri nove teksture detalja, ali to se može pokazati vrlo neefikasnim. Poželjno je koristiti do četiri teksture detalja, a ako ih treba biti više onda je poželjno podijeliti teren na zone tako da ih svaka zona ima maksimalno četiri.

Na slici 8 prikazani su rezultati dviju navedenih tehnika za teksturiranje terena. Miješanje tekstura detalja očito je vizualno prihvatljivije, ali također valja primijetiti kako vizualna razlika postepeno nestaje u daljini.



Slika 8: Tekstura boje preslikana na teren (lijevo) i rezultat miješanja tekstura detalja (desno)

Tehnike razine detalja primjenjuju se i na teksture. S obzirom na to da je miješanje tekstura detalja računski i memorijski višestruko zahtjevnije od korištenja samo jedne teksture boje, tipično se teksture detalja koriste samo u blizini promatrača, a tekstura boje za udaljenija područja. Osim toga, moguće je koristiti i teksture u različitim rezolucijama.

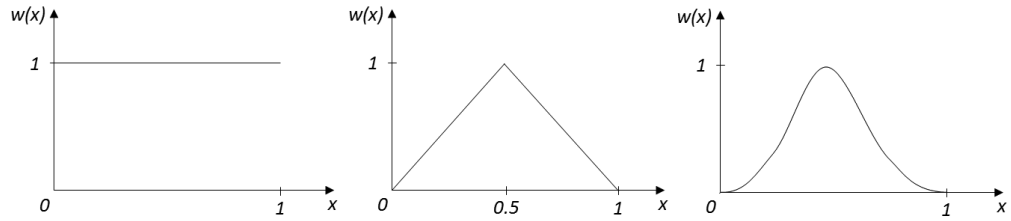
2.3.3. Proceduralno teksturiranje

Do sada smo razmotrili dvije metode za teksturiranje terena i pritom smo govorili o teksturi boje, teksturama detalja i mapi težina. Ostaje nam razmotriti kako možemo dobiti navedene teksture.

Teksture detalja potrebno je ručno pripremiti korištenjem alata za uređivanje slike i stoga ih nećemo ovdje detaljnije obrađivati. Slika 9 prikazuje primjere tekstura detalja.

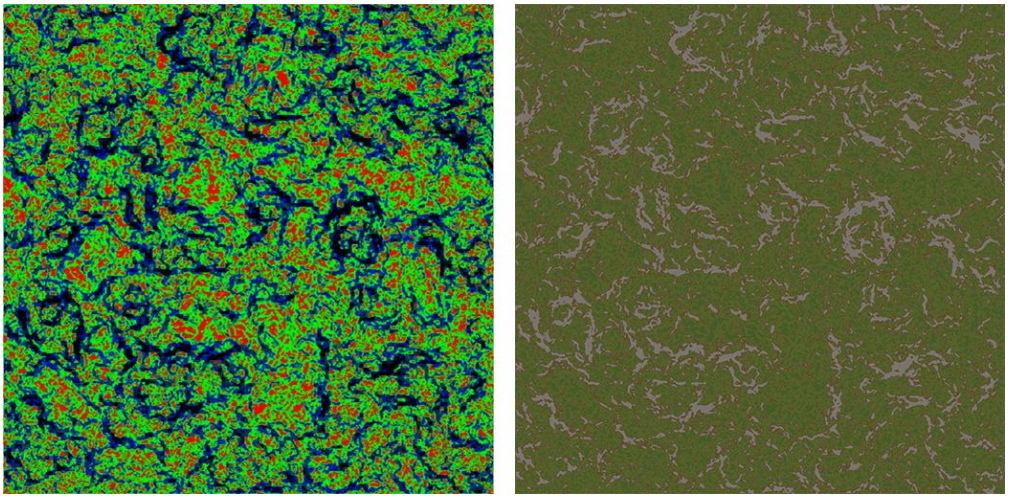
Slika 9: Teksture detalja (prve tri teksture su preuzete s www.assetstore.unity3d.com, a zadnje dvije s www.cgtextures.com)

Da bismo proceduralno generirali mapu težina opisanu u poglavlju 2.3.2 prvo je potrebno svakoj teksturi detalja odrediti parametre tla kojem pripada. Najčešće se uzimaju u obzir nagib i nadmorska visina terena, ali bitni faktori mogu biti i blizina vode, izloženost vjetru, količina svjetla itd. Kad je definirana donja i gornja granica za svaki odabrani parametar, možemo reći da je odabrano područje koje tekstura preferira. Ostaje pitanje na koji način pojedina tekstura preferira definirano područje? Preferira li više područja uz donju granicu ili možda ima jednak utjecaj unutar cijelog područja? Nazovimo funkciju koja odgovara na ta pitanja težinska funkcija. Težinsku funkciju potrebno je definirati za svaki odabrani parametar tla. Slika 10 prikazuje primjere različitih težinskih funkcija. Najčešće se koristi trokutasta funkcija (sredina), ali moguće je definirati proizvoljnu krivulju. Težinska funkcija je definirana na intervalu $[0, 1]$ koji se potom preslikava na dozvoljeno područje za pojedini parametar. Izvan dozvoljenog područja uzima se da je težinska funkcija jednaka nuli.



Slika 10: Primjeri mogućih težinskih funkcija

Kad je svakoj teksturi detalja određeno dozvoljeno područje parametara i definirane težinske funkcije, moguće je izračunati njihove težine i zapisati ih u mapu težina. Za svaki piksel mape težina potrebno je provjeriti na koji dio terena se preslikava te dohvatiti vrijednosti definiranih parametara na tom mjestu, odnosno iznos nagiba, nadmorske visine itd. Zatim se koriste težinske funkcije da bi se svakoj teksturi odredila težina za dohvaćene vrijednosti parametara. Nakon toga, dobivene vrijednosti se skaliraju tako da u sumi daju 1 i zapišu u mapu težina u promatrani piksel, svaka težina u svoj kanal teksture. Ako postoji više definiranih parametara tla, ukupne težine mogu se dobiti množenjem pojedinačnih. Rezultat tog postupka prikazan je na slici 11 lijevo. Slika prikazuje mapu težina u formatu RGBK generiranu za teren čija je visinska mapa fraktalni šum sa slike 2. Težine su izračunate s obzirom na nagib površine korištenjem trokutastih težinskih funkcija.

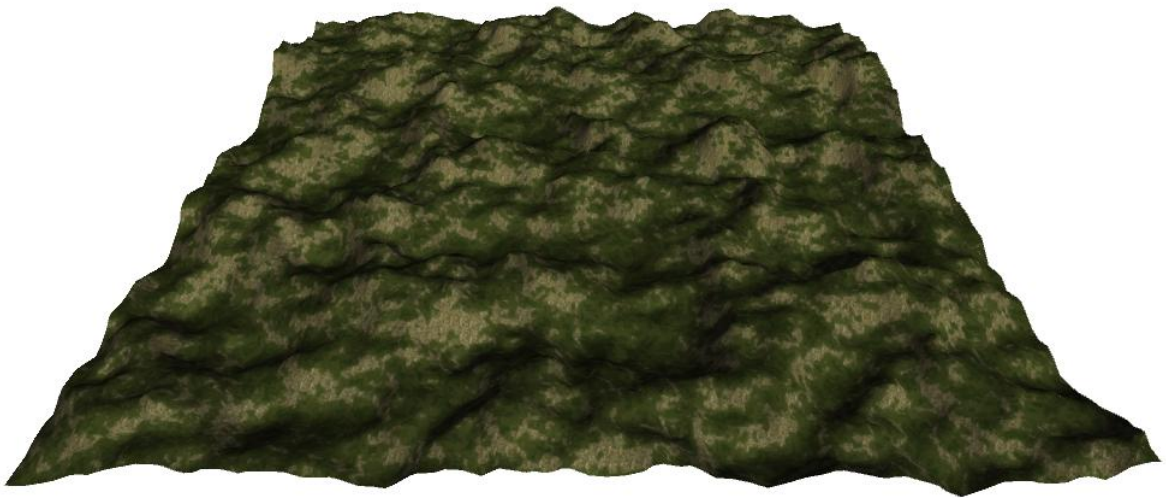


Slika 11: Mapa težina u formatu RGBK (lijevo) i tekstura boje (desno)

Postupak generiranja teksture boje vrlo je sličan postupku generiranja mape težina. Prvo se izračunaju težine svake pojedine teksture detalja, ali umjesto da se težine zapisuju u teksturu težina, boje se odmah pomiješaju sukladno težinama i konačna boja se zapisuje u teksturu boje. Bitna razlika je što se ne uzorkuju originalne teksture detalja već se koristi prosječna boja čitave teksture, odnosno boja njene posljednje mipmape. Teksture detalja tipično sadrže visoke frekvencije i kad bismo uzorkovali originalne teksture i potom ih

miješali, dobili bismo jednu visokofrekventnu teksturu s vrlo vidljivim razlikama između dva susjedna piksela. Korištenjem prosječne boje teksture uklanjaju se visoke frekvencije i omogućava se gladak prijelaz između prikaza teksturama detalja i prikaza teksturom boje. Rezultat ove metode prikazan je na slici 11 desno.

Konačan izgled terena s proceduralno preslikanim teksturama prikazan je na slici 12.



Slika 12: Teren s proceduralno preslikanim teksturama

3. Pokrov

Nakon što smo ostvarili trodimenzionalni prikaz terena i preslikali na njega teksture, potrebno je na teren postaviti vegetaciju i kamenje. Pri simulaciji okoliša odvojeno se pristupa detaljima, poput trave i sitnog kamenja, i većim objektima poput stabala, grmlja i većeg kamenja.

3.1. Detalji

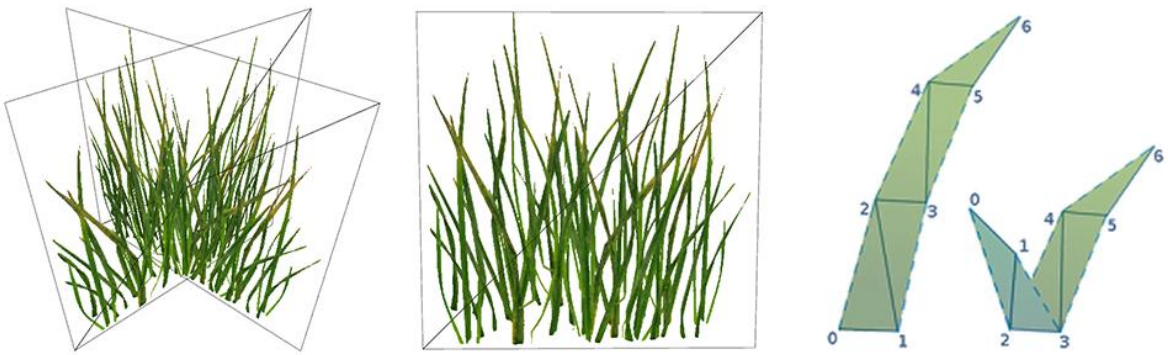
Kad govorimo o detaljima na površini terena, govorimo o svemu što je relativno malih dimenzija i uočljivo je samo kad se promatrač nalazi u neposrednoj blizini. U detalje površine najčešće ubrajamo travu i sitnije kamenje, ali može biti bilo što sličnih dimenzija. Mogli bismo reći da je trava ipak uočljiva iz daljine i da stoga ne bi trebala spadati u detalje, ali činjenica je da se iz daljine ne vide pojedine vlati trave već samo zelena površina. Takvu površinu uspješno povezujemo i prepoznajemo kao travu, ali travu kao takvu ipak ne vidimo dok se dovoljno ne približimo. Udaljene zelene površine moguće je simulirati teksturama, ali za bujnu travnatu livadu potrebne su posebne tehnike.

3.1.1. Geometrijska reprezentacija trave

Mogli bismo se zapitati zašto je potreban drugačiji pristup geometriji trave, odnosno, zašto ne bismo svaku vlat trave modelirali vlastitom mrežom trokuta. Problem s takvim pristupom je što bismo i za relativno male travnate površine vrlo brzo dosegnuli maksimum vrhova i trokuta koje grafički protočni sustav može obraditi u stvarnom vremenu. Iz tog razloga osmišljene su različite efikasne metode za geometrijsku reprezentaciju trave.

Jedan od čestih pristupa je modeliranje busena trave s dva prekrižena kvadrata koji tvore X-formu (slika 13 lijevo). Na svaki kvadrat se preslika tekstura cijelog busena i isključi se odbacivanje stražnjih poligona (engl. *backface culling*) kako bi busen bio vidljiv sa svih strana. Takav pristup uspješno prikazuje busen trave sa samo četiri trokuta što ga čini vrlo jeftinim i efikasnim. Jedna mana ovog pristupa je što trava nestaje dok se gleda odozgo.

Sljedeći pristup je korištenje osnog panoa (engl. *axial billboard*) za modeliranje jednog busena trave (slika 13 sredina). Osni pano je kvadrat, tj. dva trokuta na koje je preslikana tekstura busena trave i koji se stalno okreću prema promatraču. Pogrešno bi bilo usmjeriti pano prema samoj poziciji promatrača jer bi panoi s njegove lijeve strane bili vidljivo drugačije usmjereni od panoa s njegove desne strane. Zato se pano obično postavlja tako da gleda u smjeru normale na ravninu projekcije. Iako nije intuitivno, takvim pristupom se omogućava da svi panoi imaju jednaku orijentaciju te im se orijentacija ne mijenja dok se promatrač kreće već samo dok se promatrač rotira što dodatno sakriva činjenicu da je busen trave predstavljen običnim kvadratom. Ovaj pristup je još jeftiniji od prethodnog te je moguće dozvoliti i djelomičnu nagnutost trave kako trava ne bi posve nestala dok se gleda odozgo.



Slika 13: Prikaz geometrijskog modela trave u X-formi (lijevo), kao pano (sredina) i kao lista trokuta za svaku vlat trave (desno). Desna slika preuzeta iz Kemen, 2012.

Razvijeni su i složeniji pristupi poput pristupa korištenog u projektu *Outerra* (Kemen, 2012). Svaka travka modelirana je zasebno s pet trokuta (slika 13 desno). Zaključilo se da je šteta koristiti pet trokuta za manje travke pa se takve travke dijele u V-formu. Time se postiže veća gustoća trave i bolja pokrivenost površine terena.

Iako je riječ o vrlo jednostavnim pristupima, konačni rezultat je vrlo uvjerljiv. Slika 14 prikazuje područje na kojoj je svaki busen trave predstavljen osnim panoom.

Slika 14: Područje prekriveno travom. Svaki busen trave je jedan osni pano.

Ostale vrste detalja poput sitnijeg kamenja najčešće se prikazuju korištenjem mreže trokuta. Treba napomenuti da je mreža trokuta drugih detalja značajno jednostavnija od mreže trokuta busena trave u kojem je svaka travka zasebno modelirana. Iz tog razloga moguće je koristiti mrežu trokuta za ostale detalje, ali ipak s oprezom da se ne prekorači propusnost grafičkog protočnog sustava.

3.1.2. Ubrzavanje iscrtavanja

Detalji su sveprisutni na terenu i budući da popunjavaju vrlo malo vizualnog prostora potrebno ih je jako mnogo da bi se postigao vjerodostojan izgled. Iz tog razloga, detalji zauzimaju naviše računalnih resursa i potrebno je koristiti različite optimizacijske metode da bi trajanje njihovog iscrtavanja ostalo unutar dozvoljenih granica.

3.1.2.1 Grupiranje detalja

Mogli bismo svaki detalj jedan po jedan slati grafičkom protočnom sustavu, ali komunikacija između CPU-a i GPU-a vrlo je skupa i poželjno je imati što manje poziva iscrtavanja (engl. *draw call*). Efikasno rješenje je grupiranje detalja u zone tako da detalji koji imaju isti materijal i prostorno se nalaze blizu jedan drugog budu iscrtani istovremeno

kao jedan objekt (engl. *batching*). Budući da su detalji grupirani i prostorno, takav pristup omogućava i efikasno odbacivanje velikog broja detalja po projekcijskom volumenu. Unatoč tome, scene krajolika najčešće pogledom sežu do obzora i količina detalja koja se prikazuje u takvoj sceni još uvijek je prevelika za iscrtavanje u stvarnom vremenu.

3.1.2.2 Odbacivanje detalja s obzirom na udaljenost

Već smo napomenuli da je definicija detalja upravo to da ih se vidi samo iz neposredne blizine što znači da iscrtavanje detalja ima smisla ograničiti i s obzirom na udaljenost od promatrača. Vrlo je jednostavno postaviti granicu udaljenosti do koje se detalji iscrtavaju i potom na aplikacijskoj razini odbaciti sve zone detalja koje se nalaze dalje od dozvoljene granice. Nažalost, takvo rješenje uvodi novi problem. Nakon što dođe u dozvoljeno područje iscrtavanja, zona će odjednom iz potpuno nevidljive postati potpuno vidljiva što će uzrokovati već spomenuti efekt iskakanja. Mogli bismo pomisliti da taj efekt ne bi bio vidljiv kad bi se vidljivost određivala na razini busena umjesto na razini cijele zone, ali ljudsko oko jako dobro primjećuje nagle promjene te je za navedeni problem potrebno koristiti neko naprednije rješenje.

3.1.2.3 Razine vidljivosti

Osnovna ideja je umjesto dvije razine vidljivosti (potpuna vidljivost i potpuna nevidljivost) uvesti postepeni prijelaz iz nevidljivog u vidljivo kroz određeni tranzicijski pojas. Dva najčešća pristupa postepenoj promjeni prozirnosti su blijeđenje (engl. *fadeout*), gdje se objekt postepeno miješa s pozadinom dok ne postane nevidljiv, te izrezivanje (engl. *cutout*) gdje se objekt postepeno izrezuje i iscrtava sa sve manjom površinom. Glavna razlika između dva navedena pristupa je što izrezivanje za metodu preslikavanja prozirnosti (engl. *alpha mapping*) koristi provjeru prozirnosti (engl. *alpha testing*), a blijeđenje koristi miješanje po prozirnosti (engl. *alpha blending*).

Provjera prozirnosti iscrtava objekt tako da odbacuje sve fragmente čija je α vrijednost niža od nekog postavljenog praga. Postepeno pojavljivanje objekta može se ostvariti tako da se postavljeni α prag postepeno smanjuje od 1 (potpuna nevidljivost) prema 0 (potpuna vidljivost) pri čemu se sukladno povećava i vidljiva površina objekta, odnosno smanjuje površina koja se izrezuje. Ovaj pristup je vrlo brz, nema dodatnih zahtjeva na računalne resurse te jako dobro funkcionira s objektima koji su prikazani teksturom koja je već djelomično prozirna (poput busena trave).

Drugi pristup koristi miješanje po prozirnosti. Umjesto da se fragmenti odbacuju s obzirom na definirani prag, fragmentima se određuje u kojoj mjeri su prozirni, odnosno u kojoj mjeri su vidljivi objekti koji se nalaze iza. Za ovaj pristup je uobičajena vrijednost $\alpha = 0$ za potpunu prozirnost i $\alpha = 1$ za potpunu neprozirnost. Ovaj način je fleksibilan i omogućuje kontinuirane razine prozirnosti pojedinog fragmenta, ali je sporiji od prethodne metode jer se obrađuju i fragmenti koji su potpuno prozirni (Pandžić, 2011). Osim toga, takve objekte potrebno je iscrtavati od nazad prema naprijed što znači da ih je prije iscrtavanja potrebno sortirati. Ovaj pristup se najčešće koristi za potpuno neprozirne objekte predstavljene 3D modelom (poput kamenja) te općenito daje ugodnije i blaže prijelaze od prethodne metode.

3.1.2.4 Iscrtavanje objekata s istom geometrijom

Na kraju je potrebno još razmotriti različite tehnike iscrtavanja velikog broja objekata koji imaju istu geometriju, ali različite vrijednosti drugih atributa poput pozicije, orijentacije, veličine, boje i sl. U takvom slučaju nema smisla slati geometriju grafičkom protočnom sustavu za svaki objekt odvojeno već je moguće ostvariti uštede tako da se geometrija šalje samo jednom.

Osni pano objašnjen u prethodnom poglavlju u svojoj definiciji sadrži orijentaciju i opis vlastite mreže trokuta. Iz tog razloga, jedan osni pano moguće je u potpunosti generirati dinamički u procesoru geometrije (engl. *geometry shader*) bez potrebe da se informacija o njegovoj geometriji šalje grafičkom protočnom sustavu i pohranjuje u memoriji. Za generiranje velikog broja osnih panoa moguće je definirati mrežu trokuta u kojoj svaki vrh predstavlja poziciju jednog osnog panoa. Potom se pojedini panoi opetovano instanciraju i orijentiraju u procesoru geometrije svakim prolaskom kroz grafički protočni sustav. Svaki vrh takve mreže trokuta osim pozicije može sadržavati i vrijednosti ostalih atributa specifičnih za pojedini pano. Na taj način omogućava se efikasno generiranje velikog broja panoa koji su geometrijom isti, ali mogu imati različite vrijednosti ostalih atributa.

Za prikaz ostalih objekata čija geometrija i orijentacija nisu strogo definirane najčešće se koristi metoda instanciranja (engl. *instancing*) (Carucci, 2005) koja koristi jedan spremnik vrhova (engl. *vertex buffer*) za geometriju i drugi spremnik instanci (engl. *instance buffer*) koji sadrži informacije specifične za pojedinu instancu objekta poput veličine, orijentacije i sl. Spremnik vrhova se pohranjuje u priručnu memoriju grafičke procesorske jedinice te se

potom iscrtava za svaku instancu u spremniku instanci. Ako je podržan, ovaj pristup je uobičajeni način generiranja velikog broja objekata s istom geometrijom.

3.2. Viši pokrov

Kad se govori o višem pokrovu najčešće se misli na stabla i veće kamenje. To su objekti koji su vidljivi i s velikih udaljenosti te je stoga potrebno koristiti drugačije metode za njihovo efikasno prikazivanje. Poseban naglasak stavlja se na stabla zbog njihove vizualne i geometrijske složenosti.

3.2.1. Generiranje stabala

Stabla su u računalnoj grafici oduvijek predstavljala izazov. Ručno modeliranje svake grane i svakog pojedinog lista mukotrpan je posao i zato su ostvareni mnogi alati za proceduralno generiranje stabala.

U osnovi svakog alata za generiranje biljaka stoje Lindenmayerovi sustavi ili skraćeno L-sustavi (engl. *L-system*). L-sustav je formalni jezik koji se temelji na sustavima prepisivanja (engl. *rewriting systems*). Sastoji se od početnog skupa simbola koji se zove alfabet i skupa produkcijskih pravila koja govore kako se pojedini nizovi zamjenjuju novima. Počevši s nekim početnim uzorkom, koji se još zove aksiom, u svakom koraku se paralelno primjenjuju produkcijska pravila mijenjajući stare nizove novima. Sustav je inicijalno razvijen za opis ponašanja živih biljnih stanica s ciljem modeliranja procesa rasta i razvoja biljki. Budući da se diobe stanica događaju paralelno, upravo je to bila inspiracija za paralelnu primjenu produkcijskih pravila. L-sustavima je također moguće modelirati fraktalnu samosličnost koju mnoge biljke posjeduju i zato je najčešće korišten sustav za proceduralno generiranje stabala.

Najpoznatiji alati za računalnom potpomognuto generiranje stabala su komercijalni alati *Xfrog* (www.xfrog.com) i *SpeedTree* (www.speedtree.com) i temelje se upravo na L-sustavima.

3.2.2. Razine detalja

Slično kao pri generiranju mreže trokuta terena, možemo reći da su objekti gledani izbliza vidljivi sa svim svojim detaljima, ali ti detalji se ne primjećuju ako objekt gledamo iz veće udaljenosti. Iz tog razloga, za prikaz stabla koje se nalazi u neposrednoj blizini potrebno je koristiti model s više detalja, ali za udaljeno stablo moguće je koristiti jednostavniji model s mnogo manje poligona bez da se primijeti vizualna razlika. S tom idejom razvile su se

mnoge metode razina detalja za različite primjene, a u ovom poglavlju razmatramo njihovu primjenu na viši pokrov.

Objekti se često dijele u četiri razine detalja gdje su prve tri razine 3D modeli sa sve manjim brojem poligona dok je posljednja razina najčešće osni pano. Jedna slika katkad nije dovoljna za vjerodostojan prikaz udaljenog stabla, pogotovo ako je stablo asimetrično. Dodatan problem može biti i osvjetljenje. Kruženjem promatrača oko stabla mijenja se strana stabla koja je vidljiva pa tako i količina svjetlosti kojom je obasjana. Ta promjena u osvjetljenju ne može se odraziti na statički prikaz stabla osnim panoom. Prijelaz s osnog panoa na 3D model uzrokovao bi vidljivu promjenu zbog čega se osni pano kao posljednja razina detalja najčešće koristi za najudaljenija područja kad se stablo gotovo ni ne vidi pa tako ni navedene netočnosti. Moguće je unaprijediti osni pano tako da umjesto jedne slike koristi više slika iz različitih kutova gledanja i pritom odabire odgovarajuću sliku s obzirom na kut pod kojim se nalazi promatrač. Takvu metodu koristi *SpeedTree* za prikaz najniže razine detalja, ali vidljive su nagle promjene pri zamjeni slike koju pano koristi. Također, umjesto osnog panoa moguće je koristiti i varalice (engl. *impostors*) koje su u suštini jednake osnom panou osim po tome što se slika objekta ne generira statički već dinamički iz pozicije promatrača (Pandžić, 2011). Za male pomake kamere tako stvorena slika dobro reprezentira objekt, ali za veće pomake potrebno je ponoviti postupak. Moguće su i druge, složenije, ali vizualno kvalitetnije reprezentacije objekta u nižim razinama detalja. Umjesto jednog panoa moguće je koristiti oblake panoa (engl. *billboard clouds*) gdje se složeni objekt aproksimira nizom preklapajućih ravnina na koje su preslikani pojedini dijelovi objekta. Prednost takvog prikaza je što promatrani predmet izgleda dobro iz svih kutova gledanja te nije potrebno ponavljati postupak stvaranja pojedinih slika.

Slika 15 prikazuje razine detalja stabla generiranog alatom *SpeedTree*. Riječ je o besplatnim modelima preuzetima sa stranice www.assetstore.unity3d.com. Modeli prikazani na gornjoj slici redom se sastoje od 7188, 4420, 879 i 4 trokuta. Donja slika prikazuje iste te modele prikazane u perspektivi. Možemo primijetiti da su razlike između broja trokuta velike, ali u perspektivi se razlika gotovo ni ne vidi.

Kao i kod ostalih metoda diskretnih razina detalja, ako se ne koristi napredniji prijelaz između dvije razine, česta je pojava efekta iskakanja. Efekt iskakanja moguće je ublažiti korištenjem miješanja dviju razina detalja. Kroz neko tranzicijsko područje iscrtavaju se obje razine pri čemu jedna postepeno postaje nevidljiva, a druga potpuno vidljiva. Za

iscrtavanje prijelaznih modela koristi se miješanje po prozirnosti o kojem je već bilo riječ u poglavlju 3.1.2.3. Tranzicijsko područje unutar kojeg detalji postepeno blijede bilo je određeno udaljenošću od promatrača što je imalo smisla budući da detalji uvijek zauzimaju mali postotak ekrana. Korištenjem istog principa za prikaz većih objekata moglo bi se dogoditi da dva objekta značajno različitih veličina koja se nalaze jedan pored drugog budu prikazana u istoj razini detalja. Iz tog razloga se kod većih objekata za određivanje razine detalja ne koristi udaljenost od promatrača već površina projekcije obujmice objekta na ekran. Takav pristup je precizniji jer se objekt prikazuje u razini detalja sukladnoj postotku ekrana kojeg zauzima.

Osim pojednostavljivanja geometrije, princip razina detalja primjenjuje se i na ostale tehnike iscrtavanja. Modeli prikazani u nižoj razini detalja tipično imaju i teksture niže rezolucije ili mogu biti prikazani jednostavnijom inačicom programa za sjenčanje.

Slika 15: Usporedba razina detalja (gore) i njihov prikaz u perspektivi (dolje)

3.3. Algoritmi razmještanja

U prethodna dva poglavlja razmatrali smo geometrijski prikaz detalja i višeg pokrova te optimizaciju vezanu za njihovo efikasno iscrtavanje. Sada je potrebno postaviti vegetaciju na teren na način da to izgleda prirodno. Ručno postavljanje jednog po jednog objekta na teren bio bi vrlo mukotrpan posao te su iz tog razloga ostvarene različite metode za njihovo proceduralno postavljanje.

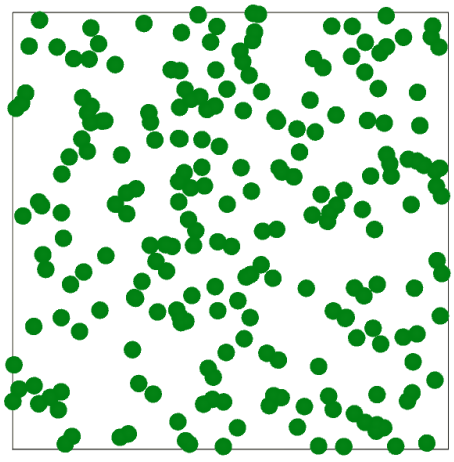
Postoje dva glavna pristupa simulaciji velikih ekosustava (Lane, 2002). Od lokalnog prema globalnom (engl. *local-to-global*), gdje se ekosustav opisuje na razini pojedine biljke te potom ostvaruje kroz interakciju i kompeticiju među jedinkama, te od globalnog prema lokalnom (engl. *global-to-local*) gdje se parametri definiraju na razini cijelog ekosustava, a distribucija biljaka ostvaruje nekim globalnim algoritmom razmještanja uz moguću interakciju s korisnikom.

U praksi su najzastupljeniji pristupi od globalnog prema lokalnom. Korisnik najčešće interaktivno odabire vrstu stabla ili trave koju želi postaviti na teren te zatim "boji" odabrani pokrov po željenim područjima na terenu. Pokrov se unutar obojenog područja automatski raspoređuje sukladno definiranim parametrima za odabranu vrstu pokrova te se unosi varijacija u boji, rotaciji i veličini tako da pojedine instance iste vrste pokrova ne izgledaju posve jednako. Takav pristup omogućava vrlo veliki stupanj kontrole gdje će se pojedina vrsta pokrova nalaziti i s kojom gustoćom te olakšava korisniku posao dozvoljavajući mu da istovremeno postavi veliki broj objekata.

U nastavku razmatramo različite metode razmještanja pokrova po terenu. Navedene metode mogu se koristiti kao pomoć pri interaktivnom postavljanju pokrova, ali i kao potpuno nezavisne metode za proceduralno postavljanje pokrova bez interakcije s korisnikom.

3.3.1. Potpuno nasumično razmještanje

Najjednostavniji algoritam razmještanja objekata je algoritam potpune nasumičnosti. Pozicija na koju ćemo postaviti naš objekt nasumično se odabire unutar definiranog područja. Slika 16 prikazuje rezultat ove metode na skupu od 255 objekata.



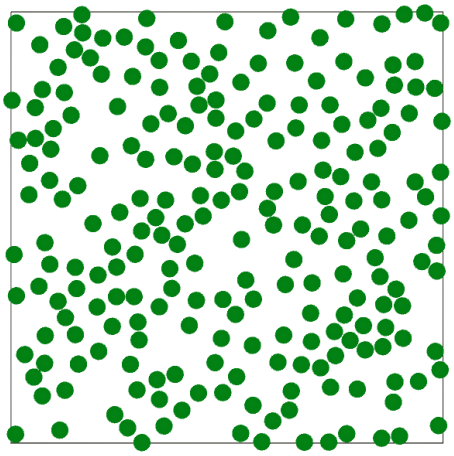
Slika 16: Nasumično razmještanje objekata

Mane ovog pristupa odmah su uočljive. Prvi problem je preklapanje točaka što bi se u 3D prostoru manifestiralo kao presijecanje geometrija dvaju različitih objekata. Osim toga, postoje područja koja uopće nisu prekrivena objektima dok su neka druga prekrivena previše gusto. Šuma generirana ovim algoritmom sadržavala bi prazne prostore i osamljena stabla što iako katkad može biti korisno, nije smisao algoritama koje razmatramo.

3.3.2. Nasumično razmještanje s provjerom kolizije

Problem preklapanja točaka možemo riješiti provjerom kolizije. Svakom objektu možemo pridijeliti radijus unutar kojeg nijedan drugi objekt ne može postojati. Prilikom postavljanja novog objekta potrebno je provjeriti hoće li na toj poziciji objekt biti u koliziji s nekim već postavljenim objektom. Ako postoji kolizija onda se objekt ne postavlja na dodijeljenu poziciju već mu se određuje nova nasumična pozicija. Algoritam se zaustavlja kad su postavljeni svi objekti ili je dosegnut maksimalni dozvoljeni broj iteracija.

Slika 17 prikazuje rezultat razmještanja s provjerom kolizije na skupu od 255 objekata. Možemo primijetiti da su nestala preklapanja između točaka, ali još uvijek postoje prazni prostori i osamljena stabla.



Slika 17: Nasumično razmještanje objekata s provjerom kolizije

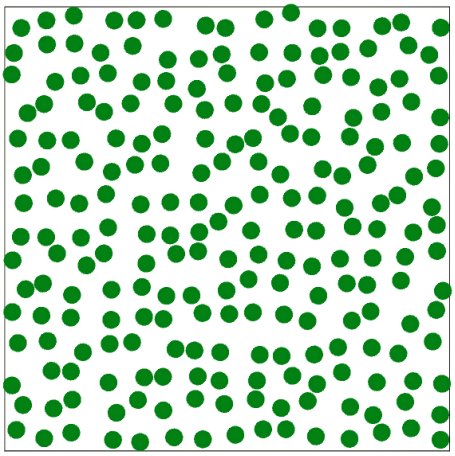
Glavni nedostatak ovog algoritma su performanse. Provjera kolizije koju je potrebno obaviti je složenosti $O(n^2)$ gdje je n broj objekata koji sudjeluju u razmještanju što može postati dosta skupo za veliki n. Algoritam je moguće ubrzati podjelom područja u zone tako da pri provjeri kolizije nije potrebno provjeravati koliziju sa svim objektima već samo s onima koji se nalaze u trenutnoj i susjednim zonama. Takva optimizacija prirodno bi se nadovezala na već spomenuto dijeljenje geometrije terena na zone.

3.3.3. Nasumični pomak s rešetke

Prethodnim algoritmima nismo uspjeli postići jednoliku distribuciju objekata već su uvijek postojala gušća i rjeđa područja. Ideja ovog algoritma je započeti od jednolike distribucije te naknadno uvesti nasumičnost.

Algoritam započinje postavljanjem objekata na pravilnu rešetku tako da se svaki objekt nalazi u svojoj ćeliji. U sljedećom koraku svaki se objekt pomakne u nasumičnom smjeru za nasumičan iznos. Ako je veličina dodijeljene ćelije veća od kruga kojim reprezentiramo pojedini objekt uvijek je moguće unijeti nasumičan pomak kojim će objekt čitavom svojom površinom ostati unutar dodijeljene ćelije što znači da neće doći do preklapanja geometrija. Dozvoljeni pomak unutar ćelije ovisit će o tome koliko je ćelija veća od objekta koji se u njoj nalazi.

Slika 18 prikazuje rezultat ovog algoritma na skupu od 255 objekata. Možemo primijetiti da su nestala velika prazna područja i da je uspješno ostvarena jednolika distribucija objekata.



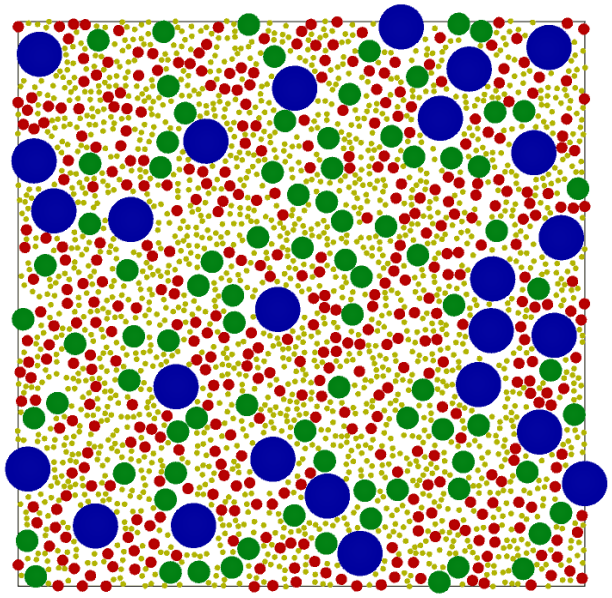
Slika 18: Nasumični pomak objekata s pravilne rešetke

Dodatna prednost ovog algoritma je njegovima brzina. Za razliku od prethodnog, složenost ovog algoritma je $O(n)$.

3.3.4. Proširenje nasumičnog razmještanja na različite vrste objekata

Do sada smo razmatrali nasumično razmještanje jedne vrste objekta po zadanom području. Različite vrste mogu imati različite radijuse te posložiti objekte različitih radijusa u pravilnu rešetku ne bi bio trivijalan posao. Za različite vrste objekata povoljnije je koristiti već opisani algoritam nasumičnog razmještanja s provjerom kolizije (poglavlje 3.3.2).

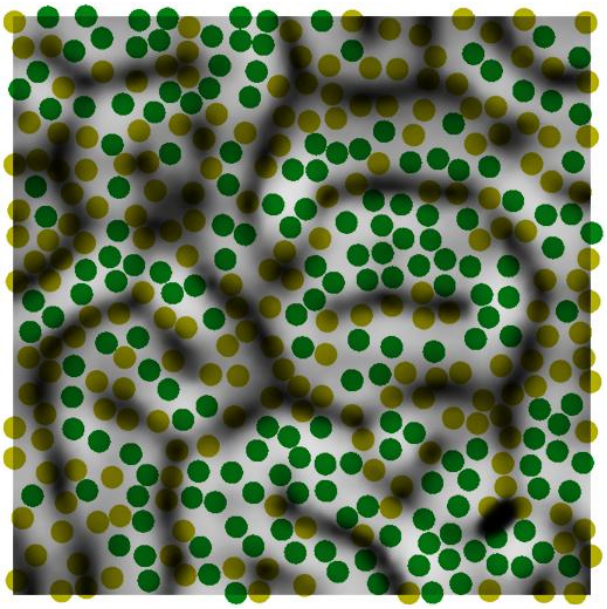
Slika 19 prikazuje rezultat takvog algoritma za četiri vrste objekata različitih radijusa. Ukupno je postavljeno 2388 objekata u 10000 iteracija.



Slika 19: Nasumično razmještanje s provjerom kolizije za četiri različite vrste objekata

Različite vrste objekata mogu imati i različite preferencije područja na kojem obitavaju. Kao što smo već razmatrali u poglavlju 2.3.3 za teksture, svakoj vrsti mogu se dodijeliti granične vrijednosti parametara poput nagiba, nadmorske visine, udaljenosti od vode i sl. Tako definirane granične vrijednosti zajedno sa zadanim težinskim funkcijama definiraju koliko pojedina vrsta preferira promatranu poziciju. Prilikom nasumičnog odabira pozicije, za svaku vrstu objekta izračuna se iznos težinske funkcije, odnosno vrijednost koliko pojedina vrsta preferira tu poziciju. Vrsta koja će biti postavljena na poziciju izabire se nasumično gdje je vjerojatnost odabira proporcionalna iznosu težinske funkcije što znači da vrste koje više preferiraju odabranu poziciju imaju veću šansu da budu izabrane. Za objekt izabrane vrste provjerava se hoće li biti u koliziji s nekim drugim objektom ako se postavi na promatranu poziciju. Ako nema kolizije, možemo postaviti objekt izabrane vrste. U suprotnom, ako postoji kolizija, odbacuje se izbrana vrste te se izbor vrste ponavlja. Ako nijedna vrsta ne može opstati na toj poziciji, bilo zbog kolizije, bilo zbog toga što pozicija nije unutar dozvoljenih područja parametara, postupak se ponavlja s nekom novom nasumičnom pozicijom.

Slika 20 prikazuje razmještaj dviju vrsta s različitim dozvoljenim područjima. Nagib terena prikazan je nijansama sive boje tako da svjetlija područja označavaju veći nagib. Žuta vrsta je definirana tako da preferira manje, a zelena veće nagibe.



Slika 20: Razmještaj dvije vrste objekata sukladno njihovim preferencijama

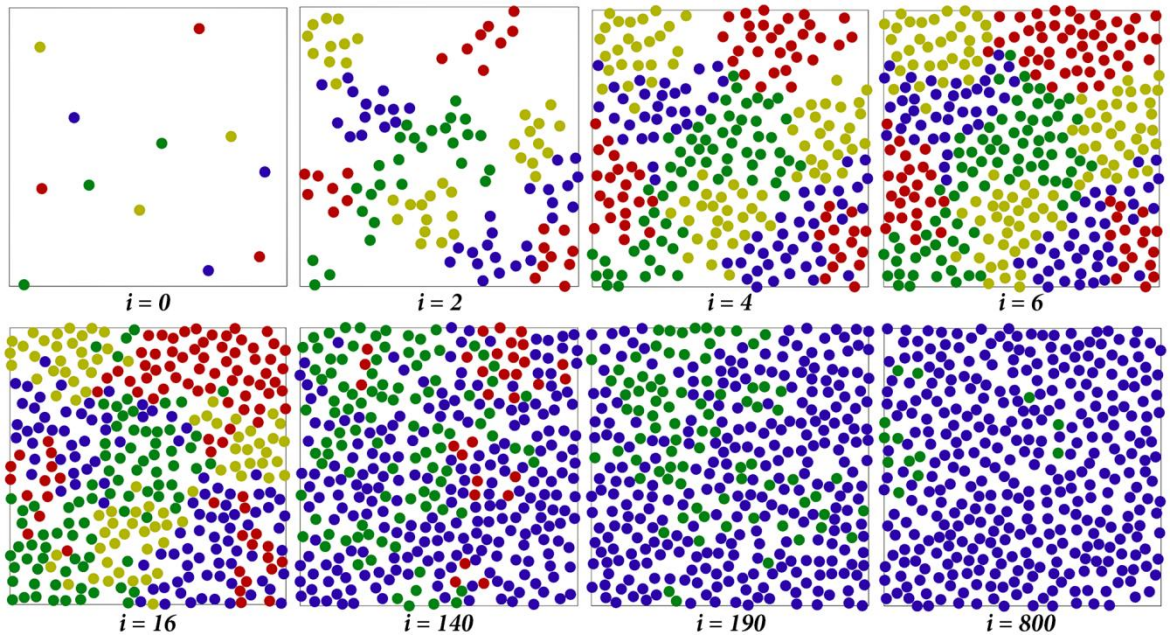
3.3.5. Algoritam preživljavanja i borbe za resurse

Algoritmima razmještanja pokušavamo postići nasumičnu razdiobu objekata koja izgleda prirodno. U stvarnosti, prirodna nasumičnost postiže se dugogodišnjim rastom i međusobnom interakcijom između biljaka. Biljke rastu paralelno i bore se za resurse. Starije biljke umiru te se svake godine stvaraju nove biljke koje zauzimaju njihova mjesta. Prirodna nasumičnost koju mi percipiramo posljedica je vrlo kompleksnog procesa. Prethodni algoritmi su pokušaji ostvarivanja nasumične razdiobe na način da se zanemari prirodni proces i direktno modelira konačni rezultat. Mogli bismo problemu pristupiti drugačije i umjesto konačnog rezultata modelirati proces rasta i razvoja biljaka te očekivati da će se prirodna razdioba automatski postići.

Cjelokupni proces preživljavanja i borbe za resurse može se aproksimirati s nekoliko koraka. Stanje sustava se inicijalizira tako da se područje rijetko naseli nekim od prethodnih algoritama nasumičnog razmještanja. Na početku svake iteracije sve biljke koje trebaju stvoriti sjeme, stvore ga unutar određenog radijusa oko svoje pozicije. Sjeme koje padne izvan granica ekosustava ili na tlo čije vrijednosti parametara ne upadaju u dozvoljeno područje za promatranu vrstu, potrebno je ukloniti. Zatim se obavlja provjera kolizije. Biljke koje su u koliziji bore se za opstanak i slabija biljka uklanja se iz ekosustava. Na kraju iteracije svakoj biljci se poveća starost te se eliminiraju biljke koje su dosegnule svoju maksimalnu dob. Postupak se ponavlja za definirani broj iteracija.

3.3.5.1 Preživljavanje

Svaka biljka predstavljena je radijusom ekološkog susjedstva (engl. *ecological neighbourhood*). Unutar tog radijusa ne smije postojati nijedna druga biljka. Budući da se svakom iteracijom stvaraju nove biljke, ekološka susjedstva često se presijecaju i potrebno je odrediti koja biljka će preživjeti. Sposobnost preživljavanja biljke određuje se funkcijom preživljavanja (engl. *viability function*) koju možemo definirati trokutastom funkcijom (slika 10 sredina) gdje x-os predstavlja normaliziranu starost u intervalu $[0, 1]$, odnosno omjer trenutne i maksimalne starosti biljke, a y-os iznos funkcije preživljavanja (Beneš, 2002). Možemo primijetiti da ovako definirana funkcija preživljavanja uzima starost kao jedini relevantan parametar preživljavanja. Biljke imaju najveću sposobnost preživljavanja u svojoj srednjoj dobi, dok mlađe i starije biljke lakše ugibaju. Kad se dvije biljke nađu u koliziji, preživljava ona biljka koja ima veći iznos funkcije preživljavanja.



Slika 21: Prikaz algoritma preživljavanja i borbe za resurse kroz iteracije

Rezultat ovog algoritma prikazan je na slici 21. Svaka točka predstavlja jednu biljku i njeno ekološko susjedstvo. Početno stanje inicijalizirano je s 12 objekata različite starosti gdje je svakoj vrsti pridijeljena maksimalna starost od deset iteracija. U svakoj iteraciji svaka biljka stvori četiri sjemenke u radijusu tri puta većem od radijusa ekološkog susjedstva. Prve četiri slike prikazuju razvoj ekosustava kroz šest iteracija algoritma. Zbog dovoljno praznog prostora biljke se nesmetano šire dok ne popune čitav prostor. Možemo primijetiti da za razliku od prethodnih algoritama, ovaj algoritam ima svojstvo grupiranja objekata iste vrste (engl. *clustering*) što je posljedica stvaranja sjemenja u neposrednoj

okolini matičnog objekta. Na sljedećoj slici (šesnaesta iteracija) mogu se vidjeti posljedice borbe između vrsta. Iako su još uvijek vidljive prethodno formirane grupe, granice između njih više nisu jednako jasne. Kako algoritam napreduje tako će jedne grupe nestajati, a druge se stvarati. Na preostale tri slike moguće je primijetiti da algoritam pokazuje svojstvo eksponencijalnog rasta. Nakon dovoljnog broja iteracija pojedine vrste ostanu istisnute i jedna vrsta postane dominantna. Nakon što je vrsta istisnuta, više ne postoji mogućnost da se njena populacija obnovi.

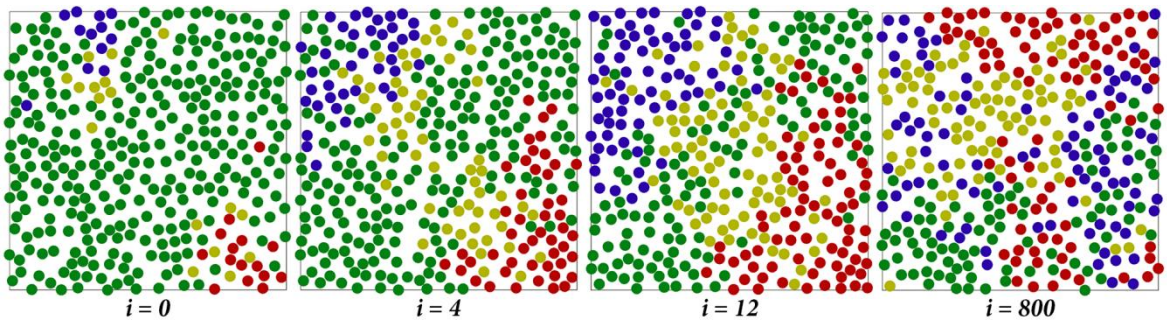
3.3.5.2 Utjecaj okoline

U prirodi, vrsta koja se previše proširila vrlo brzo potroši resurse potrebne za njen opstanak. Posljedično, slabije jedinke mnogo brže umiru te se njena populacija počinje smanjivati omogućavajući ostalim vrstama da obnove vlastitu populaciju.

Taj proces moguće je simulirati utjecajem okoline u obliku negativne povratne veze. Kad jedna vrsta zauzme previše teritorija, funkcija preživljavanja joj se smanji za određeni faktor što ostalim vrstama olakšava preživljavanje. Neka je zadano n vrsta te neka i-ta vrsta zauzima područje veličine a_i. Nova funkcija preživljavanja u koju je uključen utjecaj okoline može se definirati kao (Beneš, 2002):

$$v_k = \frac{\sum_{i=1, i\neq k}^{n} a_i}{\sum_{i=1}^{n} a_i} v_k' = \frac{a_1 + a_2 + \ \ldots + a_{k-1} + \ a_{k+1} + \ \ldots + a_n}{a_1 + a_2 + \ \ldots + a_n} v_k' \,, \tag{3}$$

gdje je $v_k{}'$ iznos inicijalno definirane funkcija preživljavanja. Utjecaj okoline simuliran je omjerom veličine područja kojeg zauzimaju ostale vrste i ukupnog područja koje je zauzeto svim vrstama. Ako vrsta k zauzima prevelik dio područja, znači da ostale vrste zauzimaju premalo i brojnik jednadžbe (3) za vrstu k bit će manji od brojnika za ostale vrste. Posljedično, tako definiran utjecaj okoline smanjit će vrsti k šansu za preživljavanje istovremeno povećavajući šanse ostalim vrstama te im tako dozvoljavajući da se obnove.



Slika 22: Utjecaj okoline potiče stabilnost sustava

Slika 22 prikazuje sustav koji je početno inicijaliziran tako da jedna vrsta u potpunosti dominira. Možemo vidjeti da sustav vrlo brzo dostiže stabilno stanje u kojem su sve vrste otprilike jednako zastupljene i uspješno ga zadržava.

4. Implementacija

Sustav za proceduralno generiranje krajolika implementiran je u programskom jeziku *C#* koristeći programsko okruženje *Visual Studio 2012* u integraciji s alatom za izradu igara *Unity3D*. Aplikacija je osmišljena kao editor gdje korisnik interaktivno podešava različite parametre i postepeno stvara željeni krajolik.

4.1. Grafičko sučelje

Sučelje aplikacije sastoji se od jednog glavnog i nekoliko pomoćnih sučelja. Pomoćna sučelja koriste se u pripremnoj fazi za definiranje tekstura i pokrova koji će se potom koristiti u glavnoj fazi za generiranje cjelokupnog krajolika. Sukladno tome, postupak generiranja krajolika možemo podijeliti u pripremnu fazu te fazu samog generiranja.

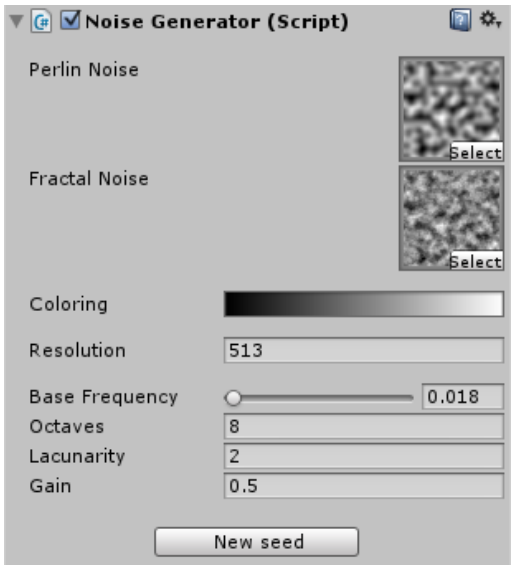
4.1.1. Pripremna faza

U pripremnoj fazi potrebno je pripremiti visinsku mapu, odabrati teksture detalja, vrste trave i višeg pokrova te svakom odabranom tipu definirati potrebne parametre.

4.1.1.1 Visinska mapa

Visinska mapa može biti preuzeta s interneta kao što je upisano u poglavlju 2.1.1 ili generirana nekim od algoritama opisanih u poglavlju 2.1.2. Aplikacija podržava generiranje visinske mape metodom fraktalnog Perlinovog šuma.

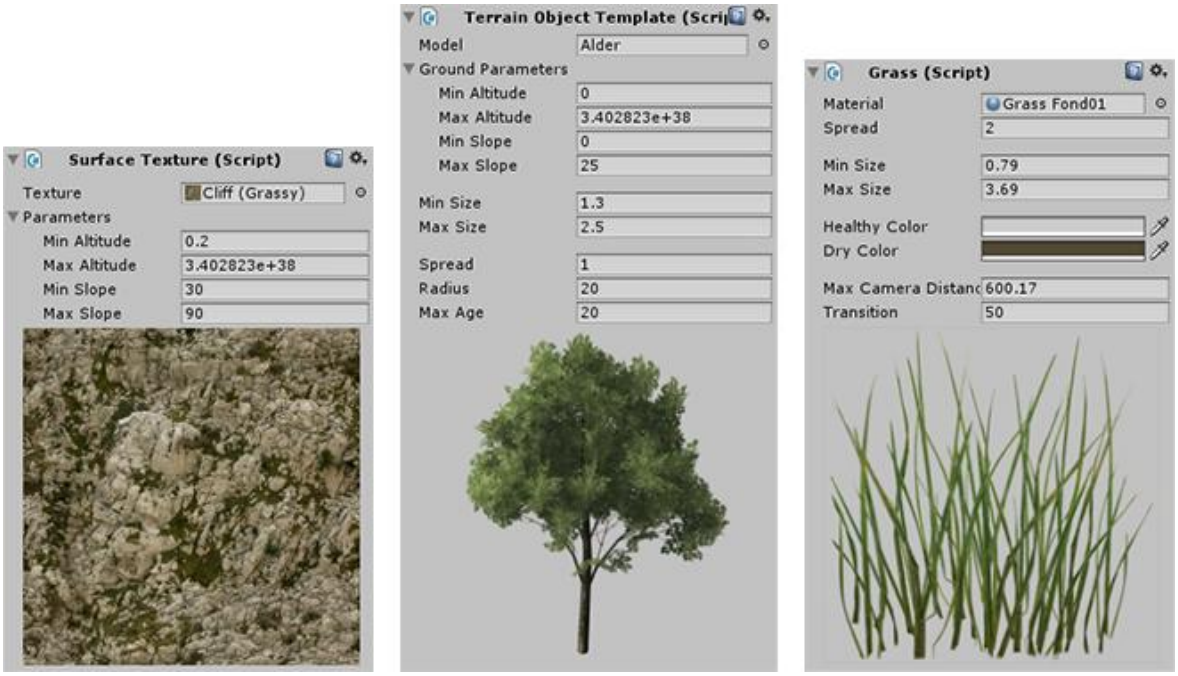
Slika 23 prikazuje korisničko sučelje za generiranje Perlinovog i fraktalnog šuma. Perlinov i fraktalni šum iscrtavaju se u teksture *Perlin Noise* i *Fractal Noise*. Parametar *Coloring* definira gradijent kojim će teksture biti obojane. Visinska mapa je siva slika i stoga se uzima gradijent od crne prema bijeloj boji, ali moguće je definirati proizvoljan spektar boja za korištenje teksture u druge svrhe. Parametar *Resolution* definira razlučivost izlaznih tekstura. *Octaves*, *Lacunarity* i *Gain* su parametri generiranja fraktalnog šuma kako je opisano u poglavlju 2.1.2.2. Dodatno je uveden parametar *Base Frequency* za kontrolu početne frekvencije. Nakon definiranja vrijednosti parametara, pritiskom na gumb *New seed* pokreće se generiranje novog Perlinovog i fraktalnog šuma.



Slika 23: Pomoćno korisničko sučelje za generiranje Perlinovog i fraktalnog šuma

4.1.1.2 Teksture detalja

Za teksturiranje terena prvo je potrebno pripremiti skup tekstura koje namjeravamo koristiti i svakoj teksturi definirati parametre tla kako je definirano u poglavlju 2.3.3. Slika 24 lijevo prikazuje sučelje za definiranje parametara teksture. Granica dozvoljenog područja nadmorske visine određena je parametrima *Min Altitude* i *Max Altitude*, a nagiba parametrima *Min Slope* i *Max Slope*.



Slika 24: Pomoćna sučelja za definiranje parametara tekstura (lijevo), višeg pokrova (sredina) i trave (desno)

4.1.1.3 Trava

Slično kao i za teksture detalja, potrebno je definirati koje vrste trave će se koristiti te svakoj vrsti definirati parametre. Slika 24 desno prikazuje pomoćno sučelje za definiranje trave. Parametri *Min Size* i *Max Size* definiraju dozvoljenu varijaciju u veličini pojedinog busena. Na sličan način, parametri *Healthy Color* i *Dry Color* definiraju varijaciju u boji. Parametri *Max Camera Distance* i *Transition* koriste se za postepeni prijelaz trave iz nevidljive u potpuno vidljivu radi umanjivanja efekta iskakanja (poglavlje 3.1.2.3). *Max Camera Distance* je udaljenost na kojoj se trava više ne vidi, a *Transition* udaljenost tijekom koje trava postepeno nestaje. Konačno, parametar *Spread* određuje udio vrste trave na terenu i koristi se za algoritme razmještanja. Nije važno definirati parametar u intervalu [0,1] jer se to obavlja automatski prilikom računanja koja vrsta trave će biti postavljena na pojedinu točku.

4.1.1.4 Viši pokrov

Pomoćno sučelje za definiranje parametara višeg pokrova prikazano je na slici 24 (sredina). Parametri grupirani imenom *Ground Parameters* imaju isto značenje kao i za teksture. *Min Size* i *Max Size* kao i kod trave definiraju dozvoljenu varijaciju u veličini objekta. Parametri *Spread, Radius,* i *Max Age* koriste se u algoritmima razmještanja. *Spread* definira koliki je udio pojedine vrste na terenu, *Radius* označava radijus ekološkog susjedstva promatranog objekta unutar kojeg ne smije postojati nijedan drugi objekt, a *Max Age* je maksimalni broj iteracija algoritma preživljavanja i borbe za resurse koje objekt može preživjeti. Nakon što dosegne starost veću od *Max Age*, objekt je potrebno izbrisati.

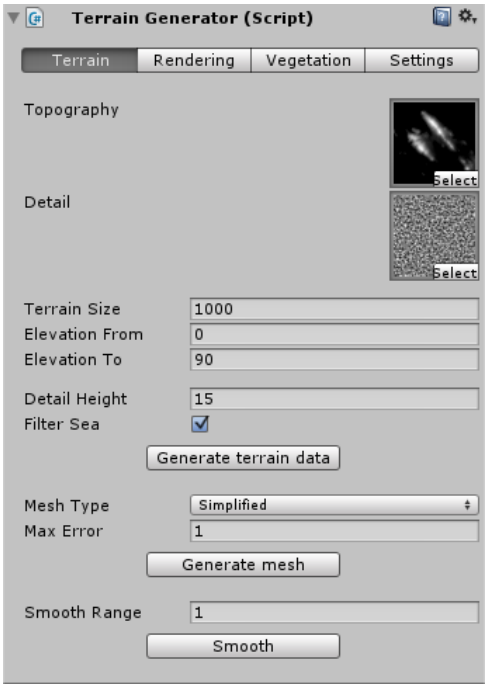
4.1.2. Generiranje krajolika

Nakon što je obavljena pripremna faza i definirani svi potrebni parametri moguće je započeti s postupkom generiranja krajolika. Postupak generiranja krajolika provodi se kroz tri glavna koraka: generiranje terena, preslikavanje tekstura te postavljanje pokrova. Sukladno tome, glavno sučelje sadrži četiri komponente, po jednu za svaki navedeni korak te jednu dodatnu za općenite postavke.

4.1.2.1 Generiranje terena

Nakon što imamo visinsku mapu, moguće je generirati mrežu trokuta za teren. Slika 25 prikazuje prvu komponentu glavnog korisničkog sučelja koja je zadužena za generiranje geometrije terena. Visinska mapa *Topography* predstavlja osnovni oblik terena čija se površina i visina zadaju parametrima *Terrain Size, ElevationFrom* i *ElevationTo*. U slučaju

da je osnovna visinska mapa previše monotona, moguće je dodati i teksturu detalja (tekstura *Detail*) kako bi se ostvario zanimljiviji izgled površine. Visina teksture detalja određena je parametrom *Detail Height*. Kad se radi s visinskim mapama koje sadrže i kopno i more, potrebno je filtrirati područje na koje dodajemo teksturu detalja, odnosno osigurati da se detalji dodaju samo na kopno. Ako je uključena opcija *Filter Sea*, tekstura detalja dodaje se samo na površinu koja u visinskoj mapi *Topography* nije potpuno crna. Konačno, pritiskom na gumb *Generate terrain data* iz navedenih parametara generiraju se svi potrebni podatci o terenu.



Slika 25: Glavno sučelje: komponenta za generiranje terena

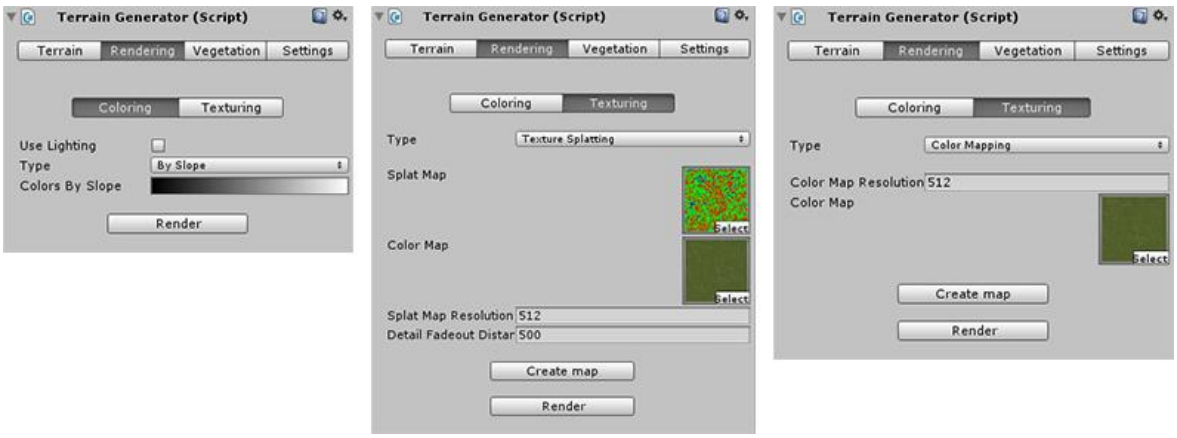
Iz tako dobivenih podataka potrebno je stvoriti mrežu trokuta. Parametar *Mesh Type* određuje metodu kojom će se generirati mreža trokuta. Odabir opcije *Brute Force* generira mrežu trokuta naivnom metodom opisanom u poglavlju 2.2, a opcija *Simplified* metodom Lindstrom-Koller opisanom u poglavlju 2.2.2. Ako je odabrana opcija *Simplified*, dodatno je potrebno odrediti vrijednost parametra dozvoljene greške *Max Error* za simplifikaciju terena. Pritiskom na gumb *Generate mesh* generira se mreža trokuta.

Moguće je da tako generirana površina terena ne bude lijepo zaglađena te je zato dodatno implementirana funkcionalnost zaglađivanja terena. Parametar *Smooth Range* definira raspon zaglađivanja. Pritiskom na gumb *Smooth* obavlja se zaglađivanje površine tako da svaka točka terena poprimi prosječnu vrijednost sebe i svih susjednih točaka unutar definiranog raspona.

4.1.2.2 Preslikavanje tekstura

Nakon što je ostvaren geometrijski prikaz terena, potrebno je definirani skup tekstura postaviti na teren. Slika 26 prikazuje komponentu glavnog sučelja zaduženu za bojanje terena. Postoje dva glavna načina za iscrtavanje terena: *Coloring* i *Texturing*. Opcija *Coloring* uključuje mogućnosti bojanja terena s obzirom na normale, nagib ili nadmorsku visinu (parametar *Type*) i prvenstveno služi za vizualizaciju opisanih svojstava terena. Pritom se mogu definirati boje u kojima će se odabrana svojstva prikazivati uz mogućnost da se uključi ili isključi osvjetljenje (slika 26 lijevo).

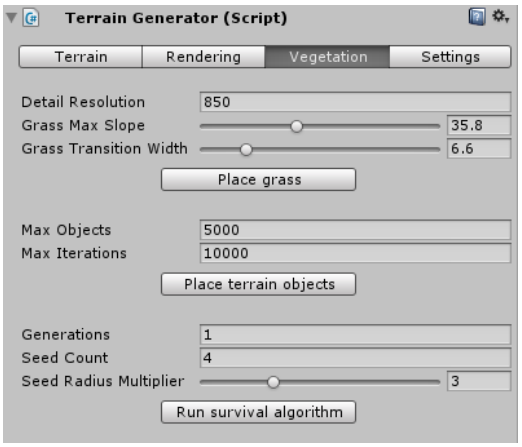
Za prikaz terena teksturama koristi se opcija *Texturing*. Ako je kao vrijednost parametra *Type* odabrana opcija *Texture Splatting* (slika 26 sredina), teren se iscrtava metodom miješanja tekstura opisanom u poglavlju 2.3.2. Mapa težina postavlja se kao vrijednost parametra *Splat Map*. Ako mapa težina još nije generirana, moguće ju je generirati pritiskom na gumb *Create map* uz prethodno definiranje rezolucije parametrom *Splat Map Resolution*. Slično značenje imaju i parametri definirani za opciju *Color Mapping* (slika 26 desno) kojom se na teren preslikava jedna tekstura boje (poglavlje 2.3.1). Dodatno, kod opcije *Texture Splatting* moguće je zadati i teksturu boje *Color Map* kako bi se u daljini omogućio glatki prijelaz između dvije tehnike teksturiranja. Pritom je parametrom *Detail Fadeout Distance* potrebno specificirati udaljenost na kojoj se u potpunosti prestaje koristiti tehnika miješanja tekstura detalja.



Slika 26: Glavno sučelje: komponenta za bojanje terena

4.1.2.3 Postavljanje pokrova

Treća komponenta glavnog sučelja (slika 27) zadužena je za prostorno razmještanje odabranih vrsta trave i višeg pokrova.



Slika 27: Glavno sučelje: komponenta za razmještanje pokrova

Prvi dio komponente zadužen je za razmještanje trave po terenu. Trava se razmješta algoritmom nasumičnog pomaka s rešetke opisanim u poglavlju 3.3.3. Parametar *Detail Resolution* određuje broj ćelija u jednom retku ili stupcu pravilne rešetke. Budući da se različite vrste trave ne ponašaju značajno različito s obzirom na nagib i nadmorsku visinu ako promatramo manje geografsko područje, moguće je jednostavno definirati maksimalni nagib terena dozvoljen za sve vrste trave. Za to je zadužen parametar *Grass Max Slope*. Na taj način se definira rubno područje nakon kojeg trava više ne može postojati. Može dogoditi da je trava vrlo visoka uz samu granicu te potom naglo nestaje zbog povećanja nagiba. Da se ne bi dogodio prenagli prijelaz, uvodi se parametar *Grass Transition Width* koji postepeno smanjuje veličinu trave kroz zadano tranzicijsko područje. Tranzicijsko područje je u ovom slučaju izraženo u stupnjevima nagiba.

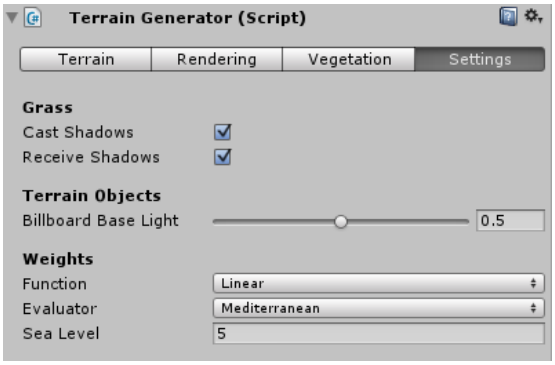
Drugi dio komponente zadužen je za postavljanje višeg pokrova algoritmom nasumičnog razmještanja s provjerom kolizije opisanim u poglavlju 3.3.2. Na teren se postavlja maksimalno *Max Objects* objekata. Ako nismo uspjeli postaviti zadani broj objekata unutar *Max Iterations* iteracija, područje se proglašava punim i algoritam se zaustavlja.

Treći dio komponente obavlja istu zadaću, ali algoritmom preživljavanja i borbe za resurse (poglavlje 3.3.5). Algoritam se provodi kroz *Generations* iteracija. U svakoj iteraciji svaki objekt stvori *Seed Count* sjemenki u radijusu ekološkog susjedstva pomnoženom s parametrom *Seed Radius Multiplier*. Radijus ekološkog susjedstva definira se u pomoćnom sučelju za viši pokrov opisanom u poglavlju 4.1.1.4.

4.1.2.4 Općenite postavke

Posljednja komponenta glavnog sučelja uključuje općenite postavke (slika 28). Parametri *Cast Shadows* i *Receive Shadows* reguliraju sjene na travi, odnosno hoće li trava bacati

sjenu na druge objekte i primati sjenu s njih. Parametar *Billboard Base Light* koristi se za viši pokrov na terenu i regulira osvjetljenje panoa kojim je predstavljena njihova najniža razina detalja. Posljednji dio komponente vezan je za računanje težina za teksture (poglavlje 2.3.3) i različite vrste višeg pokrova. Parametar *Function* definira odabir između težinskih funkcija. Podržane su prve dvije težinske funkcije sa slike 10. Parametar *Evaluator* definira način na koji se izračunavaju i miješaju težine različitih parametara tla. Opcija *Mediterranean* dodatno uključuje i parametar *Sea Level*, razinu ispod koje nijedna vrsta pokrova ne može opstati.



Slika 28: Glavno sučelje: komponenta s općenitim postavkama

4.2. Implementacijski detalji

U ovom poglavlju ukratko su opisani detalji vezani za implementaciju pojedinih postupaka.

Svaki busen trave je jedan osni pano čija se geometrija generira dinamički u procesoru geometrije (poglavlje 3.1.1). Varijacija u boji i veličini busena određuje se s obzirom na teksturu šuma koja je također proslijeđena grafičkom protočnom sustavu. Busenje iste vrste grupirano je u zone radi smanjenja broja poziva iscrtavanja. Svaka zona je jedan oblak točaka u kojoj svaki vrh predstavlja poziciju jednog busena. Cijela zona se odjednom šalje grafičkom protočnom sustavu te se na poziciji svakog vrha dinamički generira busen.

Za postepeno pojavljivanje trave koristi se algoritam izrezivanja po prozirnosti (poglavlje 3.1.2.3) kroz tranzicijsko područje zadano parametrima definiranim u poglavlju 4.1.1.3. Sličan se postupak koristi i za postepeni prijelaz iz posljednje trodimenzionalne razine detalja stabla u njegovu reprezentaciju osnim panoom. Neko vrijeme se prikazuju oba modela koja sudjeluju u prijelazu (poglavlje 3.2.2) te se vidljivost osnog panoa povećava ili smanjuje korištenjem izrezivanja po prozirnosti. Umjesto udaljenosti, kao iznos tranzicije uzima se površina projekcije obujmice.

Za ostvarenje sjena, *Unity3D* ima integriranu metodu kaskadnih tekstura sjena (engl. *cascaded shadow maps*). Scena se iscrtava iz perspektive izvora svjetlosti u Z-spremnik, tj. iscrtavaju se samo dubine. U sljedećem prolazu, prilikom iscrtavanja scene iz perspektive kamere, moguće je koristiti izračunate dubine za provjeru je li neki fragment osvijetljen ili nije. Da bi mogli koristiti navedenu metodu potrebno je generirati dva dodatna prolaza za bacanje i primanje sjena. Budući da geometrija trave ne postoji prije prolaska kroz *geometry shader*, za računanje sjena je potrebno geometriju trave iznova generirati i u svakom prolazu za sjene. Trava se prikazuje djelomično prozirnim teksturama pa je za ispravno preslikavanje sjena, prilikom upisivanja dubine u teksturu sjena, potrebno je odbaciti fragmente koji su prozirni. Izvorni kôd programa za sjenčanje trave detaljno je objašnjen u privitku (poglavlje 10).

5. Rezultati

U nastavku ćemo prikazati rezultate i komentirati utjecaj različitih parametara na ukupni izgled krajolika.

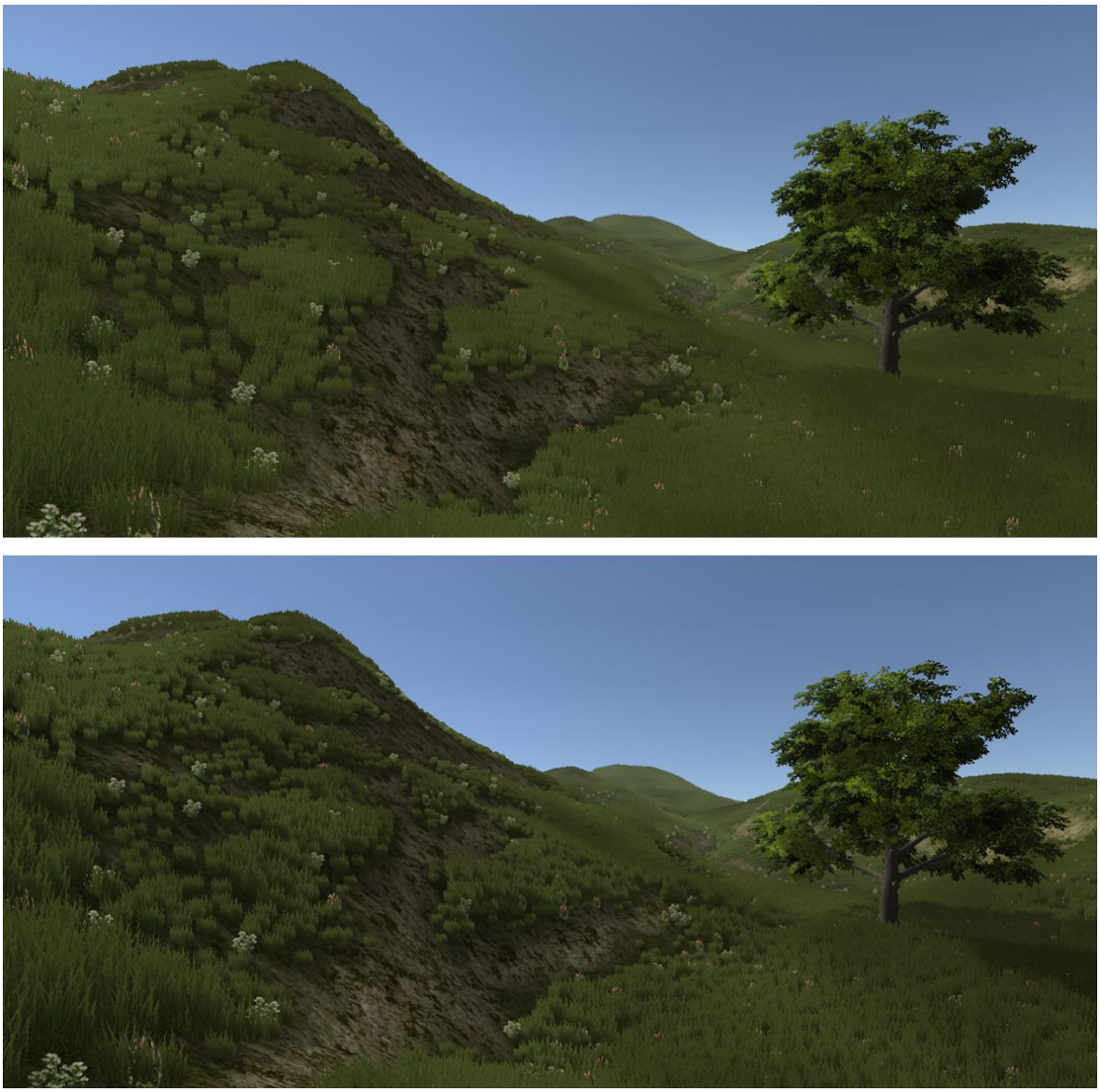
Utjecaj sjena na izgled krajolika prikazan je na slici 29. Sjene bitno doprinose dubini i trodimenzionalnosti trave, ali njihov izračun može biti vrlo skup. Sjene koje baca trava uglavnom su skrivene ispod travnatog pokrova i iako produbljuju doživljaj trave, nisu dominante u prikazu krajolika. U slučaju da je potrebno smanjiti složenost prikaza, moguće je ograničiti da trava ne baca sjene već da ih samo prima.

Doživljaj klime ostvaruje se prvenstveno kroz modele stabala i teksture trave. Teksture terena imaju bitan utjecaj prvenstveno na područjima koja nisu prekrivena travom poput vrlo strmih padina. Korištenjem odgovarajućih tekstura trave i modela vrlo brzo se stvara osjećaj vegetacijske zone. Slika 30 prikazuje konačne rezultate generatora krajolika implementiranog u ovom radu. Prikazane su različite vegetacijske zone: zimzelena šuma (gore), listopadna šuma (sredina) i mediteranska makija (dolje). Svaka prikazana scena u potpunosti je proceduralno generirana i izvodi se u prosjeku brzinom od 60 slika u sekundi unutar *Unity3D* editora na računalu s 8GB radne memorije, procesorom Dual core Intel i5-2500K 3.30GHz/3.60GHz i grafičkom karticom AMD Radeon HD 6800 u prozoru veličine 1920x1030 piksela uz mnoge dodatne mogućnosti optimizacije.

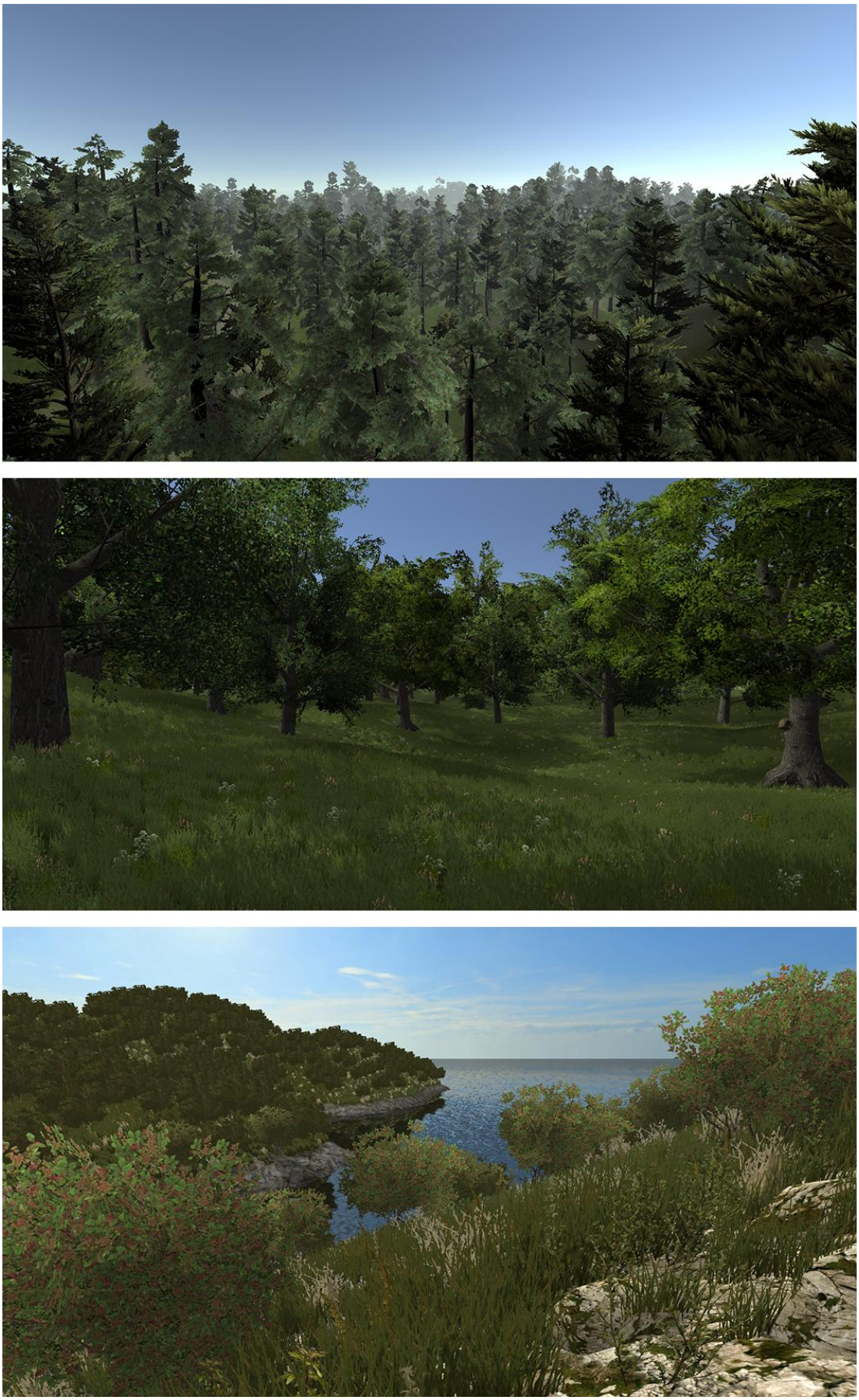
Sveukupni izgled dodatno je moguće dotjerati osvjetljenjem te definiranjem željene boje i visine trave (parametri sa slike 24). Viša trava zelene boje uz srednje osvjetljenje daje

naslutiti da je riječ o proljeću. Blago plavkasto osvjetljenje stvara osjećaj zime dok suhe nijanse trave poput sive i narančaste uz blago žućkasto osvjetljenje daju naslutiti kraj ljeta ili jesen. Za potpuni doživljaj godišnjeg doba bilo bi još potrebno stvoriti različite modele listopadnih stabala u različitim fazama opadanja lišća.

Konačno, slika 31 prikazuje usporedbu između stvarnog krajolika i njegove proceduralne inačice. Krajolik je generiran potpuno proceduralno osim dva grma u prvom planu koja su postavljena ručno kako bi se slike mogle lakše uspoređivati.



Slika 29: Prikaz trave bez sjena (gore) i sa sjenama (dolje)



Slika 30: Prikaz različitih vegetacijskih zona: zimzelena šuma (gore), listopadna šuma (sredina) i mediteranska makija (dolje)

Slika 31: Usporedba stvarnog i proceduralno generiranog mediteranskog krajolika

6. Zaključak

Zbog vizualne složenosti i velike količine podataka koji moraju biti istovremeno prikazani, područje generiranja krajolika vrlo je opsežno te da bi konačni rezultat bilo moguće prikazati u stvarnom vremenu potrebno je koristiti razvijene metode za učinkovit prikaz modela te postupke optimizacije za ubrzavanje iscrtavanja.

Simplifikacija terena metodom Lindstrom-Koller dobar je način za generiranje statičkih razina detalja te može poslužiti i kao osnova za brojne druge algoritme razina detalja. Ipak, statičke razine detalja su rigidne i ovisne o pojedinoj aplikaciji pa je moguće iskoristiti višak računalne moći i umjesto statičkih razina, radije koristiti neke od spomenutih metoda kontinuiranih razina detalja.

Podjela terena na zone nužan je korak optimizacije iscrtavanja. Ako se žele prikazati imalo veći krajolici, nužno je omogućiti brzo odbacivanje dijelova terena po projekcijskom volumenu. Vegetacijski pokrov je također potrebno grupirati po definiranim zonama radi efikasnog odbacivanja poligona, ali i zbog mogućnosti iscrtavanja većeg broja istovrsnih objekata samo jednim pozivom iscrtavanja.

Pokazalo se da proceduralno teksturiranje terena s obzirom na nagib i nadmorsku visinu može značajno olakšati proces postavljanja tekstura. Slična metoda može biti efikasno iskorištena i za globalno postavljanje detalja i višeg pokrova. Opisani algoritmi razmještanja vrlo dobro popunjavaju prostor te iako bi se mogle koristiti neke naprednije inačice temeljene na sustavu opruga, rezultati ne bi nužno bili kvalitetniji. Algoritam preživljavanja i borbe za resurse ima svojstvo grupiranja pokrova iste vrste što doprinosi vizualnom dojmu krajolika, ali ujedno zahtijeva mnogo više vremena za izvođenje te nije pogodan za dinamičko razmještanje pokrova. Za većinu primjena, dovoljni su algoritmi provjere kolizije i pomaka s rešetke.

Trava se vrlo vjerno i efikasno može predstaviti osnim panoima uz varijacije u boji i veličini za razbijanje monotonosti. Sjene koje trava baca na sebe i na teren doprinose realističnosti krajolika, ali nisu dominantne i mogu biti isključene ako je potrebno postići veće brzine iscrtavanja. Značajan utjecaj imaju i modeli stabala koji se koriste. Scene listopadnih šuma koriste modele generirane profesionalnim alatom, dok su modeli mediteranske makije vrlo primitivni te bi se korištenjem kvalitetnijih modela mogao značajno unaprijediti izgled mediteranskog krajolika. Unatoč tome, ostvareni rezultati izgledaju poprilično uvjerljivo.

7. Literatura

Archer, T.: Procedurally Generating Terrain. Midwest Instruction and Computing Symposium, Duluth, 2011.

Beneš B.: A Stable Modeling of Large Plant Ecosystems. International Conference on Computer Vision and Graphics, 2002.

Carucci, F.: GPU Gems 2: Chapter 3. Inside Geometry Instancing, 2005., http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter03.html, 18.6.2015.

de Boer, W. H.: Fast Terrain Rendering Using Geometrical MipMapping, 2000.

Discoe, B.: Virtual Terrain Project, www.vterrain.org, 16.6.2015.

Dachsbacher, C. Interactive Terrain Rendering: Towards Realism with Procedural Models and Graphics Hardware. Disertacija. Universität Erlangen–Nürnberg zur Erlangung des Grades., 2006.

Duchaineau, M., Wolinsky, M., Sigeti, D. E., Miller, M. C., Aldrich, C., Mineev-Weinstein, M. B.: ROAMing Terrain: Real-time Optimally Adapting Meshes. Proceedings of the Conference on Visualization, IEEE, 1997., 81-88.

Ebert, D.S., Musgrave, F.K., Peachey, D., Perlin, K., Worley, S.: Texturing and Modeling: A Procedural Approach. Third edition. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2003.

Hoppe, H.: Progressive Meshes. In Computer Graphics. Proceedings of SIGGRAPH, 1996., vol. 30, 99-108.

Kemen, B.: Procedural Grass Rendering, 27.5.2012., *Procedural Grass Rendering*, http://outerra.blogspot.com/2012/05/procedural-grass-rendering.html, 10.6.2015.,

Lane B., Prusinkiewicz P.: Generating Spatial Distribution for Multilevel Models of Plant Communities. In Proceedings of Graphics Interface, Calgary, 2002., 69-80.

Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L., Faust, N., Turner, G.: Real-Time Continuous Level of Detail Rendering of Height Fields. Proceedings of SIGGRAPH, 1996., 109-118

Lindstrom, P., Pascucci, V.: Visualization of Large Terrains Made Easy. Proceedings of the Conference on Visualization, IEEE, San Diego, 2001., 363-371.

Lindstrom, P., Pascucci, V.: Terrain Simplification Simplified: A General Framework for View-Dependent Out-of-Core Visualization. IEEE Transactions on Visualization and Computer Graphics, 2002., 239-254.

Miller, G.S.P.: The Definition and Rendering of Terrain Maps. SIGGRAPH, Dallas, 1986., 39-48.

Pandžić I.S., Pejša T., Matković K., Benko H., Čereković A., Matijašević M.: Virtualna okruženja: interaktivna 3D grafika i njene primjene. 1. izdanje, Zagreb: Element, 2011.

Strugar, F.: Continuous Distance-Dependent Level of Detail for Rendering Heightmaps (CDLOD). Journal of Graphics, GPU, and game Tools, Volume 14, Issue 4, 2009., 57-74.

Ulrich, T.: Rendering Massive Terrains Using Chunked Level of Detail Control. SIGGRAPH Super-Size It! Scaling Up to Massive Virtual Worlds Course Notes, New York, 2002.

8. Sažetak

Naslov: Postupci generiranja modela krajolika

U ovom radu opisan je cjelokupni proces proceduralnog generiranja krajolika kroz faze generiranja i teksturiranja terena te postavljanja pokrova. Predstavljeni su algoritmi razina detalja za teren te je detaljno objašnjena Lindstrom-Koller metoda simplifikacije. Ukratko su opisani postupci teksturiranja terena te je detaljnije objašnjena metoda miješanja tekstura detalja s obzirom na parametre tla. U nastavku su predstavljene različite metode prikaza trave i stabala te su predložene optimizacijske metode za njihovo efikasno iscrtavanje. Opisani su različiti algoritmi nasumičnog razmještanja koji su potom korišteni za vizualno uvjerljivo postavljanje pokrova. Konačno, predstavljena je implementacija generatora krajolika postupcima prezentiranima u ovom radu te je napravljena usporedba između stvarnih krajolika i njihovih proceduralnih inačica.

Ključne riječi: računalna grafika, proceduralno generiranje, krajolik, teren, pokrov, razine detalja, šum, algoritmi razmještanja

9. Abstract

Title: Generation of Environments Model

This thesis describes an overall process for procedural generation of environments through terrain generation, texturing and placement of terrain cover. Algorithms for terrain level of details were presented with detailed description of Lindstrom-Koller simplification method. Different approaches to terrain texturing were given with emphasis on a texture splatting technique based on ground parameters. Furthermore, various methods for modeling grass and trees were presented and algorithms were proposed for their efficient rendering. Scattering algorithms were described and used for achieving visually attractive yet random placement of terrain cover. Finally, an implementation of given techniques is presented and comparison given between real landscapes and their procedurally generated representations.

Keywords: computer graphics, procedural generation, environment, terrain, terrain cover, level of detail, noise, scattering algorithms

10. Privitak

U nastavku se nalaze najvažniji dijelovi izvornog kôda programa za sjenčanje trave. Program započinje definiranjem varijabli koje se mogu postavljati u sučelju. Prvi implementirani prolaz je prolaz koji iz svakog vrha stvara osni pano te ga iscrtava metodom izrezivanja po prozirnosti s obzirom na udaljenost od promatrača. Ako se ne koriste sjene, taj prolaz je dovoljan sam za sebe. Sljedeći implementirani prolaz je *ShadowCaster* kojeg *Unity3D* koristi za zapisivanje dubina u teksturu sjena. Unutar tog prolaza je za ispravno preslikavanje sjena potrebno ponovo generirati geometriju busena trave budući da ona ne postoji prije prolaska kroz grafički protočni sustav. Kôd koristi sučelje, tj. makroe koje *Unity3D* nudi za rad sa osvjetljenjem, maglom i sjenama. Detalji implementacije pobliže su opisani u komentarima.

```
Shader "Custom/GrassBillboard"
{
    Properties
    {
        /* Varijable koje se postavljaju u sučelju definirane su u poglavlju 4.1.1.3.
           Dodatno se postavlja i tekstura šuma za unošenje varijacije u boji i
           veličini. _Cutoff označava vrijednost ispod koje se fragmenti odbacuju.*/

        _MainTex ("Grass Texture", 2D) = "white" {}
        _NoiseTexture ("Noise Texture", 2D) = "white" {}
        _HealthyColor ("Healthy Color", Color) = (0,1,0,1)
        _DryColor ("Dry Color", Color) = (1,1,0,1)
        _MinSize ("Min Size", float) = 1
        _MaxSize ("Max Size", float) = 2
        _MaxCameraDistance ("Max Camera Distance", float) = 250
        _Transition ("Transition", float) = 30
        _Cutoff ("Alpha Cutoff", Range(0,1)) = 0.1
    }

    SubShader
    {
        Tags { "Queue" = "Geometry" "RenderType"="Opaque" }
        Pass
        {
            /* prolaz koji u geometry shaderu stvara osni pano i iscrtava ga metodom
               izrezivanja po prozirnosti s obzirom na udaljenost od promatrača */
            Tags { "LightMode" = "ForwardBase"}

            CGPROGRAM

            // deklaracije programa za sjenčanje
            #pragma vertex vertexShader
            #pragma fragment fragmentShader
            #pragma geometry geometryShader

            // naputci za ispravno prevođenje programa za sjenčanje
            #pragma multi_compile_fwdbase
            #pragma multi_compile_fog

            // potrebne biblioteke
            #include "UnityCG.cginc"
            #include "AutoLight.cginc"

            // struktura koju vertex shader prima iz aplikacijske razine
            struct VS_INPUT
            {
                float4 position : POSITION;   // lokalna pozicija vrha
                float4 uv_Noise : TEXCOORD0;  // teksturne koordinate teksture šuma
                fixed sizeFactor : TEXCOORD1; // faktor smanjenja trave zbog nagiba
            };
```

```
// struktura koja se iz vertex shadera prosljeđuje geometry shaderu
struct GS_INPUT
{
    float4 worldPosition : TEXCOORD0; // globalna pozicija vrha
    fixed2 parameters : TEXCOORD1;    // .x = noiseValue .y = sizeFactor
};

// struktura koja se iz geometry shadera prosljeđuje fragment shaderu
struct FS_INPUT
{
    float4 pos : SV_POSITION;          // projicirana pozicija vrha
    float2 uv_MainTexture : TEXCOORD0; // teksturne koordinate teksture busena
    float4 tint : COLOR0;              // varijacija u boji
    LIGHTING_COORDS(1,2)               /* makro za deklaraciju potrebnih
                                          podataka za osvjetljenje */
    UNITY_FOG_COORDS(3)                /* makro za deklaraciju potrebnih
                                          podataka za maglu */
    /* varijabla pos se mora tako zvati jer ju Unity pod tim nazivom koristi
       unutar makroa */
};

// deklaracija varijabli iz korisničkog sučelja
uniform sampler2D _MainTex, _NoiseTexture;
uniform fixed _Cutoff;
uniform float _MinSize, _MaxSize;
uniform fixed4 _HealthyColor, _DryColor;
uniform float _MaxCameraDistance;
uniform float _Transition;

// boja svjetla u sceni
uniform float4 _LightColor0;

// vertex shader
GS_INPUT vertexShader(VS_INPUT vIn)
{
    GS_INPUT vOut;

    /* transformacija vrha u globalni koordinatni sustav i prosljeđivanje
       vrijednosti šuma i faktora smanjenja trave zbog nagiba terena */
    vOut.worldPosition =  mul(_Object2World, vIn.position);
    vOut.parameters.x = tex2Dlod(_NoiseTexture, float4(vIn.uv_Noise.xyz,0)).r;
    vOut.parameters.y = vIn.sizeFactor;

    return vOut;
}


// geometry shader - prima vrh i na njegovoj poziciji stvara osni pano
[maxvertexcount(4)]
void geometryShader(point GS_INPUT p[1], inout TriangleStream<FS_INPUT>
                                                    triStream)
{
    // odbaci panoe koji su dalje od maksimalne dozvoljene udaljenosti
    float cameraDistance = length(_WorldSpaceCameraPos - p[0].worldPosition);
    if (cameraDistance > _MaxCameraDistance)
        return;

    // izrezivanje po prozirnosti kroz tranzicijsko područje
    float t = (cameraDistance - (_MaxCameraDistance - _Transition)) /
              _Transition;
    float alpha = clamp (1, 0, lerp (1.0, 0.0, t));

    /* računanje koordinatnog sustava panoa - pano gleda u smjeru normale na
       ravninu projekcije i ima fiksnu y-os */
    float3 viewDirection = UNITY_MATRIX_IT_MV[2].xyz;
    viewDirection.y = 0;
    viewDirection = normalize(viewDirection);
    float3 up = float3(0, 1, 0);
    float3 right = normalize(cross(up, viewDirection));

    // veličina i boja busena
    fixed noiseValue = p[0].parameters.x;
    fixed sizeFactor = p[0].parameters.y;
    float size = lerp(_MinSize, _MaxSize, noiseValue)  * sizeFactor;
    float halfSize = 0.5f * size;
    float4 tint = lerp(_HealthyColor, _DryColor, noiseValue);
    tint.a = alpha
```

```
            // stvori vrhove panoa
            float4 v[4];
            v[0] = float4(p[0].worldPosition + halfSize * right, 1.0f);
            v[1] = float4(p[0].worldPosition + halfSize * right + size * up, 1.0f);
            v[2] = float4(p[0].worldPosition - halfSize * right, 1.0f);
            v[3] = float4(p[0].worldPosition - halfSize * right + size * up, 1.0f);

            /* matrica za prebacivanje vrhova u iz globalnog u koordinatni sustav
               projekcije */
            float4x4 vpMatrix = mul(UNITY_MATRIX_MVP, _World2Object);

            // zapiši redom vrhove u izlaz
            FS_INPUT fIn;
            fIn.pos = mul(vpMatrix, v[0]);
            fIn.uv_MainTexture = float2(1.0f, 0.0f); // donji desni rub teksture
            fIn.tint = tint;

            // inicijalizacija svih potrebnih vrijednosti za izračun jačine svjetla
            TRANSFER_VERTEX_TO_FRAGMENT(fIn);

            // inicijalizacija vrijednosti za izračun magle
            UNITY_TRANSFER_FOG(fIn,fIn.pos);

            triStream.Append(fIn);

            // ... slično i za ostale vrhove ...
        }

        // fragment shader
        float4 fragmentShader(FS_INPUT fIn) : COLOR
        {
            fixed4 color = tex2D(_MainTex, fIn.uv_MainTexture) * fIn.tint;

            // odbaci nevidljive fragmente
            if (color.a < _Cutoff)
                discard;

            // izračun osvjetljenja
            float3 lightDirection = normalize(_WorldSpaceLightPos0.xyz);

            // izračun jačine svjetlosti s obzirom na teksture sjena i svjetlosne mape
            float atten = LIGHT_ATTENUATION(fIn);

            float3 ambient = UNITY_LIGHTMODEL_AMBIENT.xyz;
            float3 normal = float3(0,1,0);
            float3 lambert = float(max(0.0,dot(normal,lightDirection)));
            float3 lighting = (ambient + lambert * atten) * _LightColor0.rgb;

            color = fixed4 (color.rgb * lighting, 1.0f);

            // na konačnu boju dodaje se utjecaj magle
            UNITY_APPLY_FOG(fIn.fogCoord, color);

            return color;
        }

        ENDCG
    }

    Pass
    {
        // prolaz koji iz perspektive svjetla zapisuje dubine u teksture sjena
        Tags { "LightMode" = "ShadowCaster" }
        Fog { Mode Off }
        ZWrite On ZTest LEqual

        CGPROGRAM

        // ... deklaracije programa za sjenčanje i potrebne biblioteke ...
        // ... strukture VS_INPUT i GS_INPUT kao i u prethodnom prolazu ...

        struct SHADOW_VERTEX
        {
            /* struktura uvedena radi preglednosti budući da je varijabla
               naziva "vertex" potrebna unutar makroa */
            float4 vertex : POSITION;
        };
```

```
            struct FS_INPUT
            {
                float2 uv_MainTexture : TEXCOORD0;    // koordinate teksture busena trave
                V2F_SHADOW_CASTER;                    /* makro za deklaraciju podataka
                                                         potrebnih za bacanje sjena */
            };

            // ... deklaracije varijabli i vertex shader kao i u prethodnom prolazu...

            // geometry shader
            [maxvertexcount(4)]
            void geometryShader(point GS_INPUT p[1], inout TriangleStream<FS_INPUT>
                                                              triStream)
            {
                // ... stvaranje osnog panoa kao i u prethodnom prolazu ...
                // zapiši redom vrhove na izlaz
                FS_INPUT fIn;

                SHADOW_VERTEX v;
                v.vertex = mul (_World2Object, vertices[0]);
                fIn.uv_MainTexture = float2(1.0f, 0.0f);

                /* inicijalizacija svih potrebnih vrijednosti za izračun udaljenosti vrha
                   od izvora svjetla (koristi "v.vertex") */
                TRANSFER_SHADOW_CASTER(fIn)

                triStream.Append(fIn);

                // ... slično i za ostale vrhove ...
            }

            fixed4 fragmentShader (FS_INPUT fIn) : COLOR
            {
                // za ispravne sjene potrebno je odbaciti prozirne fragmente
                fixed alpha = tex2D(_MainTex, fIn.uv_MainTexture).a;
                if (alpha < _Cutoff)
                    discard;

                // vraća udaljenost fragmenta od izvora svjetlosti
                SHADOW_CASTER_FRAGMENT(fIn)
            }

            ENDCG
        }
    }
}
```