Proceduralno modeliranje beskonačnih terena

Ergotić, Adam

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 393

PROCEDURALNO MODELIRANJE BESKONAČNIH TERENA

Adam Ergotić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 393

PROCEDURALNO MODELIRANJE BESKONAČNIH TERENA

Adam Ergotić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 393

Pristupnik:	**Adam Ergotić (0036524760)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Proceduralno modeliranje beskonačnih terena**

Opis zadatka:

Proučiti osnovne izrade terena. Posebice obratiti pažnju na moderne tehnologije proceduralnog načina generiranja koje omogućuje izradu vizualno atraktivnih terena visoke kvalitete uz utjecaj vremenskih uvjeta i okolišnih parametara. Ostvariti prikaz terena razrađenog modela. Načiniti testiranje na nizu primjera i usporedbu s mogućnostima koje nudi programski pogon Unreal Engine. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 28. lipnja 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 393

Proceduralno modeliranje beskonačnih terena

Adam Ergotić

Zagreb, kolovoz 2024.

Sadržaj

Uvod

Proceduralno modeliranje terena je metoda računalne grafike kojom, uz pomoć algoritma, možemo generirati podatke potrebne za prikaz terena umjesto ručnog unošenja podataka. Navedena metoda vrlo je korisna prilikom stvaranja većih površina budući da je za takve terene potrebna velika količina podataka. Primjenom nekoliko pravila generiranja terena moguće je učinkovito generirati raznovrsne i kvalitetne terene.

Metoda proceduralnog generiranja beskonačnih terena može se koristiti u različitim aplikacijama poput simulacija, video igrica i aplikacija za virtualnu stvarnost. Ona omogućuje stvaranje jedinstvenih terena što daje svakom dijelu svijeta specifičan identitet.

Neke od prednosti proceduralnog generiranja beskonačnog terena su skalabilnost, raznolikost i interaktivnost. Proceduralni generiranje omogućuje stvaranje velikog svijeta i pri tome ne zauzima previše memorije (ovisno o implementaciji i načinu alokacije memorije). Algoritmi poput Perlinovog šuma omogućuje generiranje terena koji izgleda prirodno i bez ponavljajućih uzoraka, što znatno doprinosi raznolikosti svijeta. Također, budući da je svijet „beskonačan“, korisnik ima neograničenu slobodu istraživanja svijeta.

Kako bi proceduralno generiranje ispravno radilo, potrebno je pripaziti na nekoliko bitnih detalja. Budući da je potrebno generirati veliku količinu podataka, potrebno je pripaziti na način alokacije (i dealociranje) memorije kako bi program radio učinkovito. Također, potrebno je uspostaviti sustav skladištenja (engl. *cache*) izračunatog terena kako ne bi došlo do redundantnog izračuna terena i zamrzavanja ekrana (engl. *freeze*) prilikom izračuna novog terena.

Za implementiranje proceduralnog generiranja terena u ovom radu koristi se programski jezik C++ i OpenGL. Osim navedenih tehnologija, koristi se i biblioteka FastNoiseLite za generiranje Perlinovog šuma koji služi za određivanje visine terena. Dodatna poboljšanja generiranja terena su napravljene pomoću pogona Unreal Engine. Također, opisane su alternativni alati koji se mogu koristiti za proceduralno generiranje terena.

1. Povijest proceduralnog generiranja terena

Koncept proceduralnog generiranja prisutan je u računalnoj grafici od same pojave računalne grafike. Jedan od početnih izazova proceduralnog generiranja je bio razvoj algoritma za proceduralno modeliranje krajolika prirodnog izgleda. Glavna motivacija takvih algoritama bila je mogućnost izrade velikih i složenih okruženja bez potrebe ručne izrade koja bi zahtijevala puno vremena i planiranja.

Neke od početnih metoda koje su se koristile za proceduralno generiranje terena bili su fraktali[1]. Korištenje fraktala omogućilo je kreiranje vrlo kompleksnih struktura. Jedan od primjena fraktalnih algoritama je bilo generiranje planinskih terena za koje su se koristili algoritam premještaja središnje točke (engl. *midpoint displacement algorithm*) ili algoritam dijamant-kocka (engl. *diamond-square algorithm*).

Početkom 1980-ih godina, Ken Perlin razvio je funkciju gradijentnog šuma, Perlinov šum, koja je tada postala neophodna metoda za proceduralno generiranje. Perlinov šum omogućuje generiranje glatkog i kontinuiranog šuma koji se može koristiti za stvaranje realističnih tekstura i terena.

Tijekom 1990-ih godina, zahvaljujući razvoju računalne tehnologije, dolazi do potreba za okruženjima koja su realističnija i prirodnija. Samim time, dolazi do poboljšanja i razvoja novih tehnika. Jedna od njih je Perlinov šum s više oktava koji omogućuje kombinaciju više funkcija (s različitim frekvencijama i amplitudama) kako bi generirao još detaljnije i raznovrsnije terene. Osim toga, došlo je do razvoja metoda generiranja terena temeljenog na vokselima (engl. *voxel*) što je imalo veliki doprinos u aplikacijama za geološko modeliranje i simulaciju.

Od 2000-tih godina i nakon, proceduralno generiranje terena je sve više napredovalo zajedno s složenijim algoritmima i pojačanjem računalne snage. Kao poboljšanje Perlinovog šum, nastao je Simplex šum koji ima poboljšane performanse i kvalitetu terena.

Danas proceduralno generiranje terena koristi se u različitim aplikacijama. Neki od primjera su video igrica „Minecraft“ koja koristi generiranje terena temeljeno na vokselima te video igrica „No Man's Sky“ koja koristi proceduralno generiranje za stvaranje novih planeta s

[1] U računalnoj grafici, kompleksni geometrijski uzorak nastao pomoću matematičkih jednadžbi i algoritama

jedinstvenim karakteristikama. Osim video igrica, primjena proceduralnog generiranja terena nalazi se i u prikazu podataka stvarnog svijeta, poput kreiranje terena iz satelitskih snimki i geografskih informacijskih sustava.

Također, istražuju se mogućnosti primjene umjetne inteligencije i strojnog učenja kako bi dodatno poboljšalo generiranje kompleksnih i realističnih terena. Budući da već postoji jako puno detaljnih snimaka Zemlje, mogle bi se iskoristiti navedene snimke kako bi se trenirao model koji bi generirao krajolike koji su slični postojećim krajolicima.

2. Teorijska podloga za implementaciju

2.1. Ključni koncepti proceduralnog modeliranja

Prilikom planiranja implementacije algoritma za proceduralno generiranje terena potrebno je pripaziti na nekoliko ključnih koncepata koji znatno utječu na rad i uspješnost algoritma.

Za početak, potrebno je odrediti pravila po kojima algoritam generira teren što omogućuje konzistentnost algoritma. Navedena pravila određuju kako će se različiti elementi terena (npr. modeli) rasporediti prilikom generiranja.

Također, potrebno je omogućiti kontrolu nad algoritmom uz pomoć parametara. Samo promjenom nekoliko parametara, korisnici mogu dobiti veliki broj različitih generiranih terena. To omogućuje lakše korištenje i visoku prilagodljivost algoritma.

Jedan od ključnih koncepata je slučajnost i šum. Kako bi se stvorilo realistično okruženje, neophodno je uvesti šum prilikom generiranja terena. Šum omogućuje generiranja terena bez ponavljajućih uzoraka što omogućuje stvaranje krajolika prirodnog izgleda.

2.2. Funkcije šuma

Funkcije šuma su matematički algoritmi koji generiraju pseudo-slučajne vrijednosti. Funkcije šuma imaju raznovrsne primjene poput stvaranje tekstura i modela prirodnog izgleda. One imaju sljedeće karakteristike:

- Pseudo-slučajnost
- Glatkost
- Dimenzionalnost

Pseudo-slučajnost se odnosi na vrijednost koje funkcija vraća kao izlaz. Iako brojevi izgledaju nasumično izabrani, funkcija je potpuno deterministična te će za iste ulazne vrijednosti dati istu izlaznu vrijednost.

Osim toga, funkcije šuma imaju svojstvo glatkosti. Za razliku od pravih funkcija za slučajan izbor brojeva (engl. *random number generator; RNG*), funkcije šuma daju izlazne vrijednosti koje se postepeno povećavaju (ili smanjuju). To omogućuje glatke prijelaze između vrijednosti što rezultira glatkim prijelazima bez neočekivanih diskontinuiteta terena.

Razlika između prave funkcije za slučajan izbor brojeva i funkcije šuma (u ovom slučaju Perlinov) prikazana je na slici Slika 2.1.

Slika 2.1 Razlika izlaznih vrijednost *RNG* funkcije i Perlinovog šuma

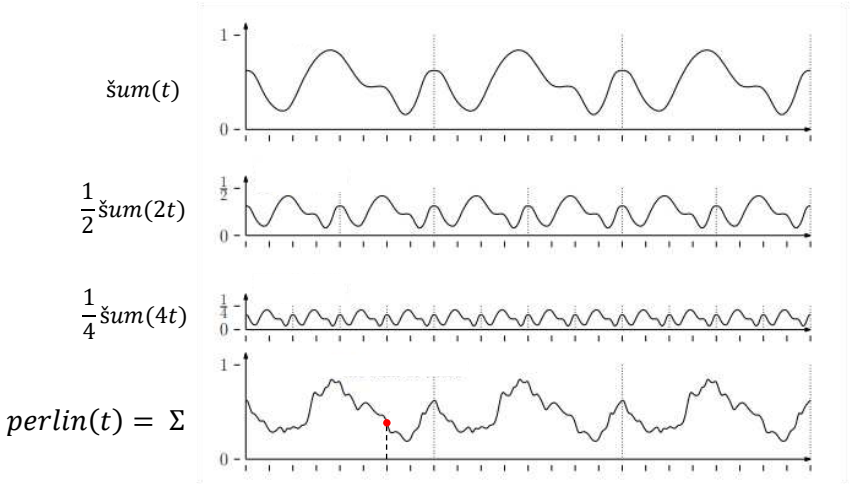
Perlinov šum u jednoj dimenziji zbroj je niza funkcija šuma različitih amplituda i frekvencija. Navedene funkcije šuma mogu se definirati na sljedeći način:

$$šum(t) = f(t \bmod n)$$

Funkcija šuma ima određeni uzorak do vrijednosti n nakon koje se ponavlja. Koristeći formulu funkcije šuma, funkcija Perlinovog šuma može se zapisati ovako:

$$perlin(t) = \sum_{i=0}^{k} p^i * šum(2^i + t)$$

U navedenoj formuli, *k* predstavlja oktavu najveće frekvencije, a *p* predstavlja vrijednost upornosti (engl. *persistence value*) i njegova vrijednost mora biti između 0 i 1. Vrijednost upornosti omogućuje kontrolu nad prisutnosti funkcija viših frekvencija. Što je vrijednost bliža 1 to su funkcije s višom frekvencijom prisutnije. Sumu svih funkcija moguće je vidjeti na sljedećoj slici (Slika 2.2).



Slika 2.2 Suma funkcija za Perlinov šum (p = 0.5)

Algoritam Perlinovog šuma (1D) prima vrijednost na x-osi za koju vraća vrijednost zbroja svojih funkcija u toj točki.

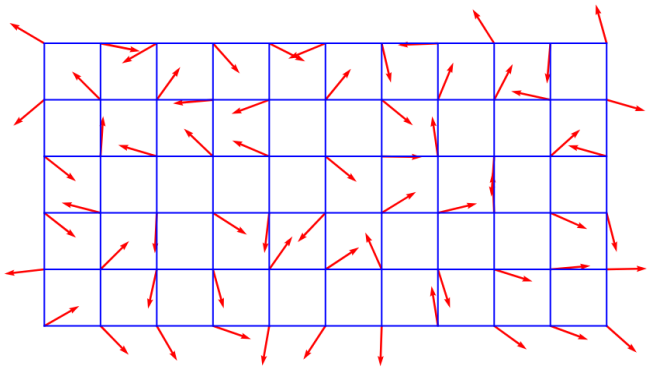
Zadnje svojstvo funkcija šuma je dimenzionalnost. Funkcije šuma mogu se definirati u jednoj, dvije ili više dimenzija što omogućuje korištenje funkcija šuma za različite primjene (npr. jedna dimenzija za audio frekvencije, dvije dimenzije za generiranje mape visina).

Neki od poznatih funkcija šumova su: Perlinov šum, Simplex šum, vrijednosni šum, Worleyev šum i fraktalni šum. U ovom radu, koristit će se Perlinov šum i Simplex šum.

2.2.1. Perlinov šum (engl. *Perlin noise)*

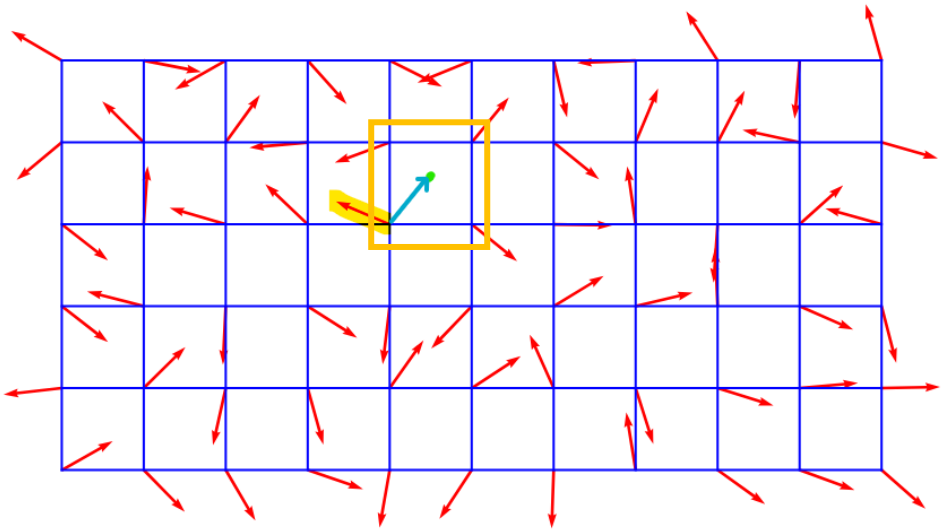
Budući da se Perlinov šum koristi za generiranje terena u ovom radu, u nastavku je detaljnije opisan princip rada algoritma Perlinovog šuma (u dvije dimenzije).

Prvo je potrebno generirati rešetku (engl. *grid*) gdje u svakom vrhu imamo nasumično generirani gradijentni vektor. Svako polje rešetke ima 2^d gradijentnih vektora, gdje je d broj dimenzija. Rezultat generiranja rešetke prikazan je na slici Slika 2.3.



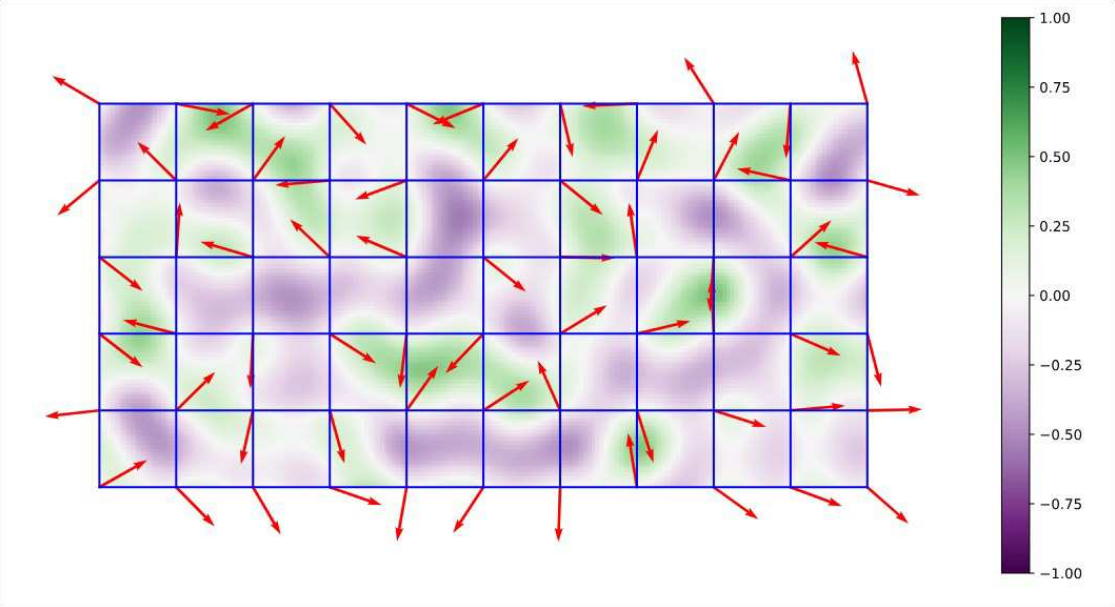
Slika 2.3 Generirana rešetka s gradijentnim vektorima

Drugi korak algoritma je određivanje skalarnog produkta gradijentnih vektora i *offset* vektora. Prvo je potrebno odrediti u kojem se polju nalazi tražena točka. *Offset* vektor je vektor od izvorne točke gradijentnog vektora do tražene točke. Primjer izračuna *offset* vektora prikazan je na slici Slika 2.4. Za svaki gradijentni vektor unutar polja rešetke, potrebno je izračuna skalarni produkt između gradijentnog vektora i *offset* vektora.



Slika 2.4 Primjer *offset* vektora (svijetlo plava boja) za točku (zelene boje)

Kao zadnji korak preostalo je interpolirati vrijednosti između svih skalarnih produkata. Rezultat interpolacije prikazan je na slici Slika 2.5. Bojom je označena interpolirana vrijednost (svih skalarnih produkata polja) za svako polje u rešetki.

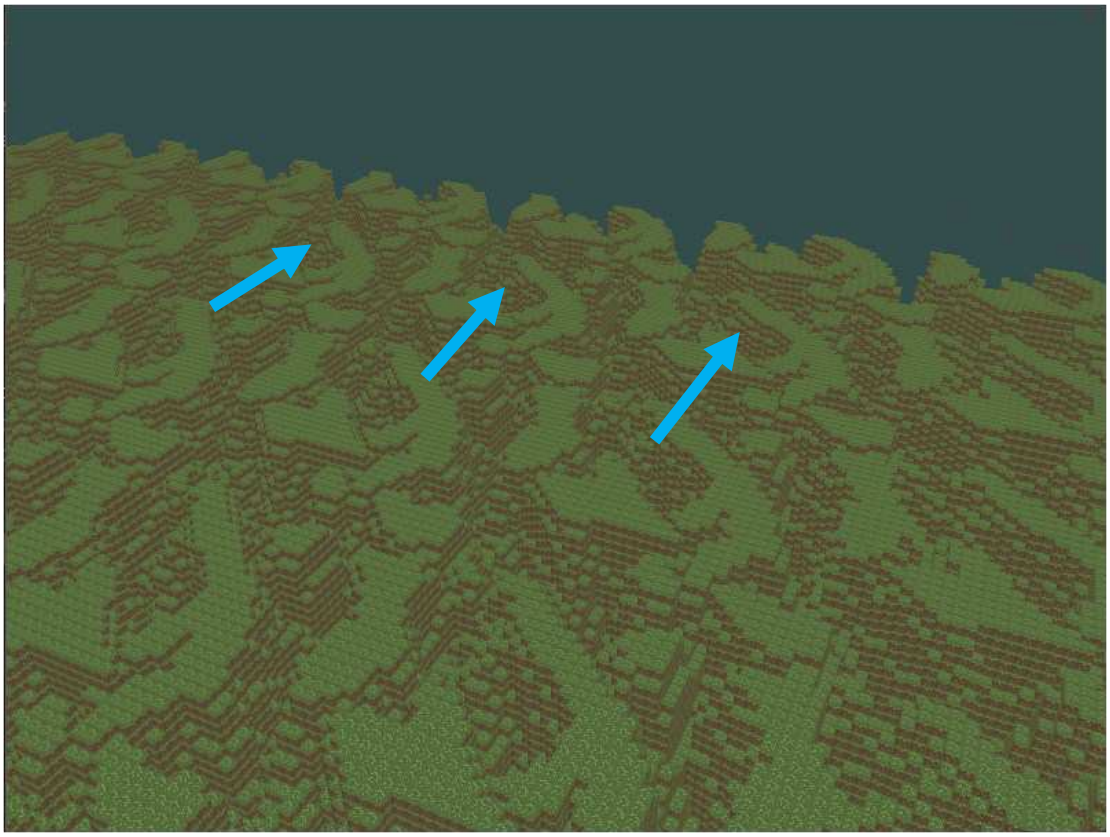


Slika 2.5 Rezultat krajnjeg koraka algoritma (interpolacija)

Iako je moguće nasumično generirati gradijentne vektore za svaki vrh rešetke, rezultat algoritma neće dati rezultat koji je zadovoljiv (raznovrstan krajolik, bez ponavljanja i bez naglih neprirodnih promjena terena). Osim toga, ako koristimo pravu RNG funkciju za generiranje gradijentnih vektora, onda će za iste ulazne vrijednosti funkcija šuma dati različite vrijednosti (kontradikcija sa svojstvom pseudo-slučajnosti).

Za zadovoljavajući rezultat potrebno je osigurati uniformnu distribuiranost i normaliziranost generiranih vektora[1]. Također, umjesto prave RNG funkcije mora se koristiti funkciju generiranja brojeva sa svojstvom pseudo-slučajnosti.

Umjesto generiranja gradijentnog vektora za svaki vrh, u početnoj implementaciji algoritma Perlinovog šuma koristilo se predefinirano polje s gradijentnim vektorima te bi se svim točkama (uniformno) pridijelio jedan od tih gradijentnih vektora. To je napravljeno radi ubrzanja algoritma te smanjenje potrebne memorije. Naime, budući da se gradijentni vektori ponavljaju u istom redoslijedu, došlo bi do ponavljanja uzoraka terena (Slika 2.6).



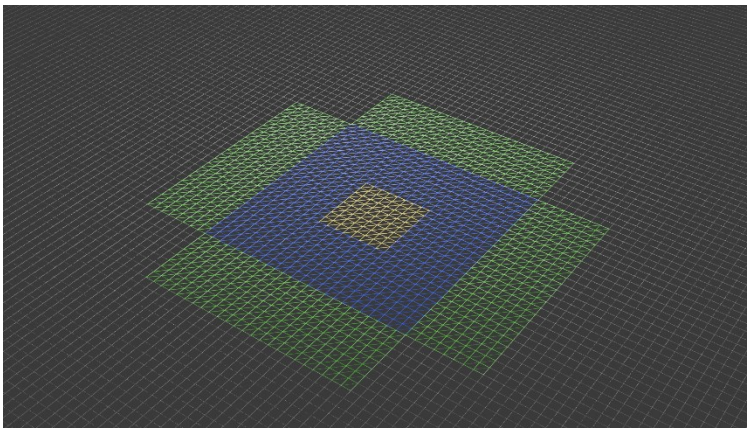
Slika 2.6 Ponavljanje uzoraka terena

Osim za generiranje terena, šum se može koristiti i za druga svojstva terena[5]. Na primjer, može se generirati jedan šum koji prikazuje temperaturu na području i drugi šum koji prikazuje vlažnost na području. Pomoću ta dva šuma mogu se odrediti mnoga svojstva terena. Ako je na određenom području jako niska vlažnost i jako visoka temperatura, to područje možemo označiti da je pustinja.

3. Pregled algoritma i struktura podataka

U ovom poglavlju nalazi se opis algoritma i svih potrebnih struktura podataka koji se koriste za implementaciju proceduralnog generiranja terena u ovom radu.

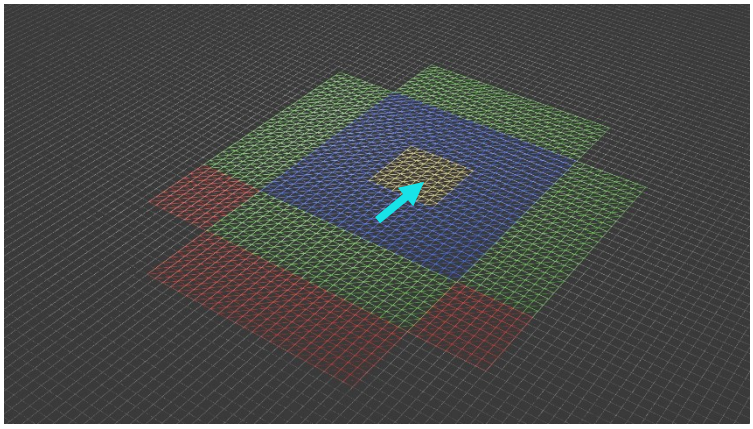
Za početak, potrebno je generirati početnu rešetku. Na sljedećoj slici (Slika 3.1) prikazana je inicijalna rešetka.



Slika 3.1 Prikaz inicijalne rešetke

Na slici je prikazana rešetka s poljima obojenima različitim bojama. Polje s žutom bojom označava aktivno polje u kojem se korisnik nalazi unutar aplikacije. Polja s plavom bojom označavaju polja koja okružuju aktivno polje korisnika. Aktivno polje i okruženje (plava boja) se iscrtavaju tijekom izvođenja aplikacije. Teren označen zelenim poljima je generiran, ali se ne prikazuje. Ova polja su generirana unaprijed kako bi tijekom prijelaza na novo polje bilo moguće odmah prikazati novi dio terena.

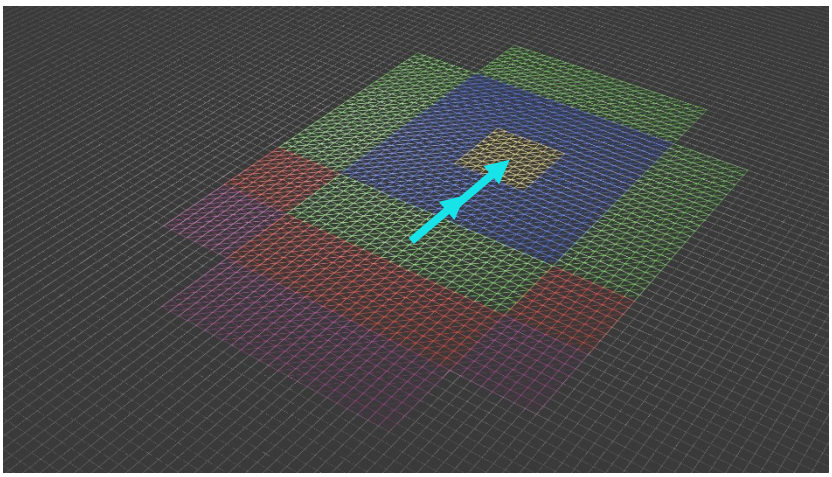
Na sljedećoj slici (Slika 3.2), prikazano je kretanje korisnika na novo polje:



Slika 3.2 Prijelaz korisnika na novo polje

Kako je korisnik prešao za jedno polje gore, tako su prethodno zeleno obojena polja postala dio okruženja aktivnog polja koji se iscrtavaju zajedno s aktivnim poljem. Osim prethodno definiranih polja, na prethodnoj slici se također pojavljuju crvena polja. Crvena polja označavaju dio terena koji se spreman za brisanje. Ako se korisnik udalji još jedno polje od tih polja, polja označena crvenom bojom će se trajno izbrisati. Glavni razlog postojanja rezervnih (zelenih) i crvenih polja je slučaj u kojem korisnik naglo prelazi između dva polja. Kada ne bi postojala rezervna i crvena polja onda bi bilo potrebno ponovno generirati dio terena što je zahtjevno te može izazvati zamrzavanje aplikacije.

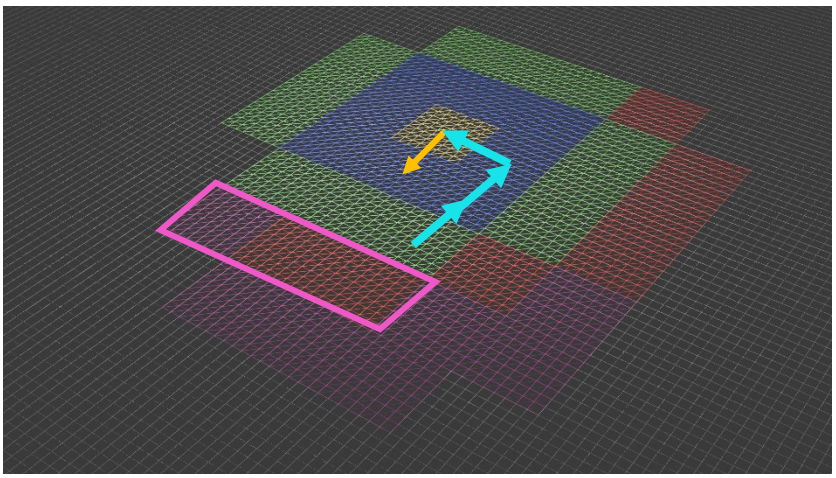
Na sljedećoj slici (Slika 3.3), korisnik je napravio još jedan prelazak prema gore:



Slika 3.3 Drugi prijelaz korisnika na novo polje

Na prethodnoj slici mogu se vidjeti polja ljubičaste boje. Ova polja označavaju polja koja su obrisana. Budući da se korisnik udaljio za još jedno polje prema gore, polja koja su prethodno bila označena za brisanje (polja crvene boje) su se obrisala.

Na sljedećoj slici (Slika 3.4), prikazan je zadnji prijelaz korisnika:



Slika 3.4 Posljednji prijelaz korisnika

U prikazanom slučaju, ako se korisnik vrati za jedno polje prema dolje (strelica narančaste boje), dio će polja biti već izračunato. U tom slučaju, već izračunata polja odmah postaju rezervna (zelena) polja, a polja koja nisu (polja crvene boje) prvo se generiraju te potom pridružuju ostalim rezervnim poljima.

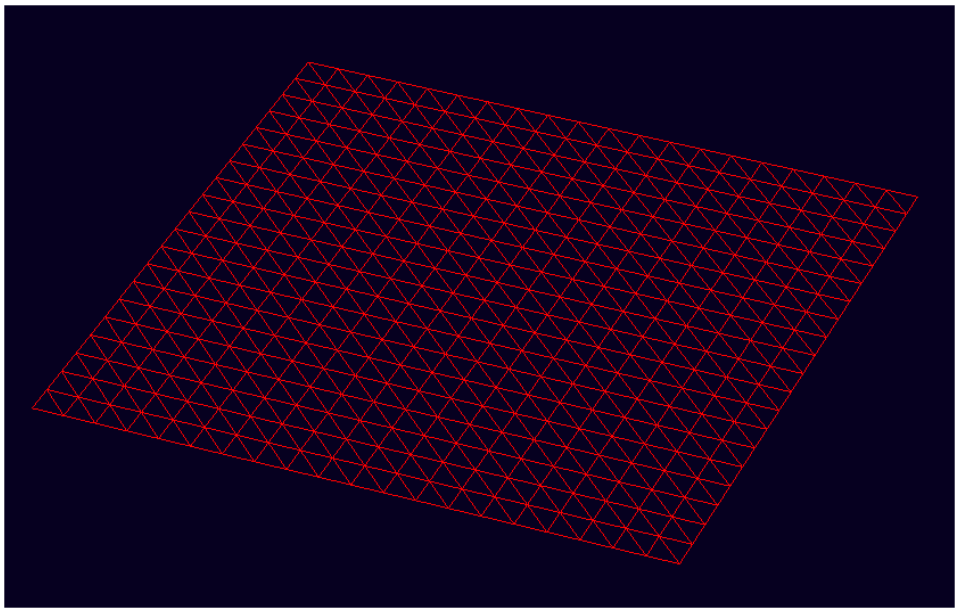
4. Implementacija

Program je implementiran korištenjem OpenGL-a i programskog jezika C++. Inicijalna implementacija, izrađena pomoću OpenGL-a, prikazana je u prvom potpoglavlju, zajedno s rezultatima. Ova implementacija uključuje algoritam opisan u poglavlju "Pregled algoritma i struktura podataka." Nakon inicijalne implementacije, kod je prilagođen za Unreal Engine. Nadalje, sva poboljšanja su primijenjena na kod namijenjen za izvođenje unutar Unreal Engine okruženja. U posljednja dva potpoglavlja opisane su mogućnosti proceduralnog generiranja unutar Unity pogona te ostali alati za generiranje terena.

4.1. Inicijalna implementacija

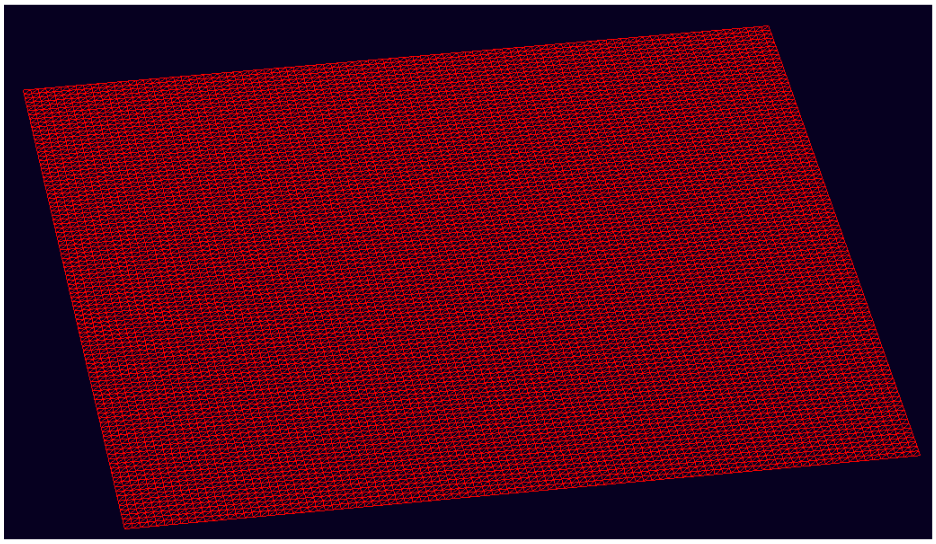
Što se tiče inicijalne implementacije, potrebno je istaknuti dvije glavne klase koje se koriste za iscrtavanje: *Tile* i *TileManager*.

Klasa *Tile* sadrži sve potrebne informacije za iscrtavanje jedne plohe koja se sastoji od m x n ćelija. Prikaz jedne instance klase *Tile* prikazan je na sljedećoj slici (Slika 4.1).



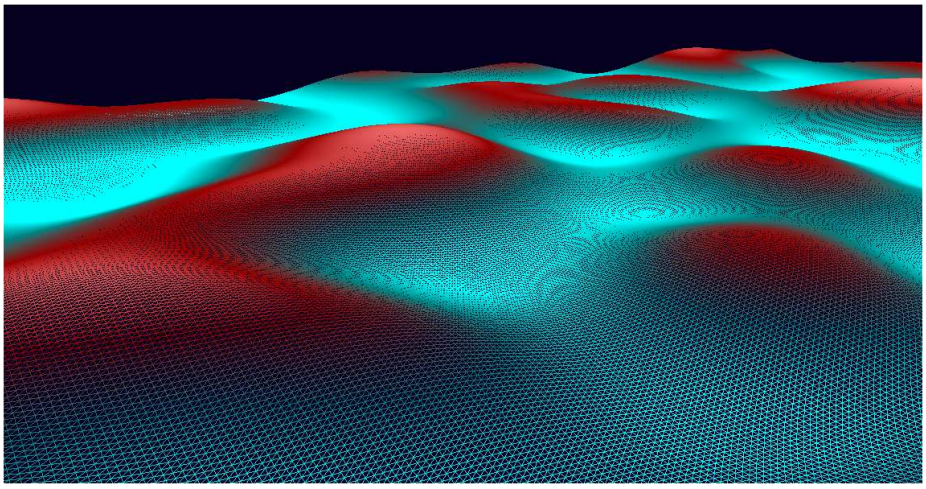
Slika 4.1 Prikaz instance Tile klase (20 x 20)

Klasa *TileManager* upravlja stvaranjem i brisanjem instanci klase *Tile* te sadrži ključni dio algoritma za generiranje sekcija terena. Instanca klase *TileManager* generira rešetku od 5x5 instanci klase *Tile*. Primjer takve rešetke prikazan je na sljedećoj slici (Slika 4.2).



Slika 4.2 Prikaz rešetke generirane pomoću TileManager klase

Završni korak uključuje primjenu funkcije šuma kako bi se dobio valovit krajolik. Za izračun vrijednosti šuma u točkama korišten je Perlinov šum pomoću biblioteke FastNoiseLite. Primjenom funkcije šuma na vrhove rešetke dobiven je rezultat prikazan na sljedećoj slici (Slika 4.3).



Slika 4.3 Primjena Perlinovog šuma na rešetku (video[2])

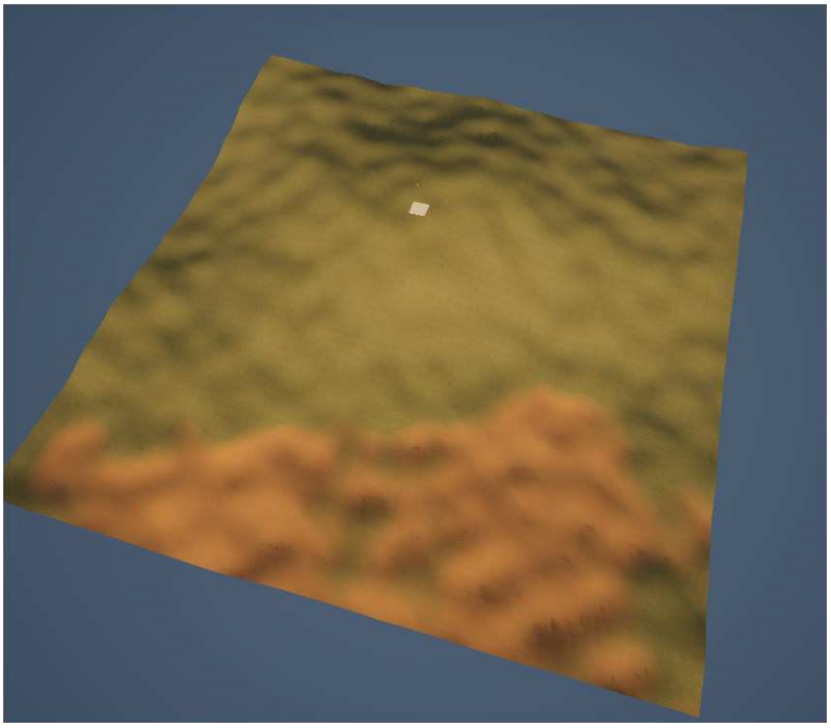
Iako algoritam pruža zadovoljavajuće rezultate u stvaranju ravnina krajolika, povećanjem broja vrhova unutar sekcije (instance klase *Tile*) dolazi do zamrzavanja zaslona. Glavni uzrok zamrzavanja je ažuriranje velikog broja vrhova koji se moraju prenijeti u memoriju grafičke kartice kako bi se sekcija terena mogla prikazati. Kako bi se spriječilo zamrzavanje ekrana, implementirano je asinkrono stvaranje terena (višedretvenost). Osim toga, trenutno

[2] https://youtu.be/IvA8QHv5014

se instanciraju potpuno nove sekcije terena prilikom izlaska iz početne sekcije terena. Umjesto stvaranja potpuno nove sekcije terena, što može biti zahtjevna operacija ako sekcija sadrži mnogo vrhova, ažuriraju se postojeće sekcije koje korisnik ne vidi. Navedene optimizacije implementirane su u sljedećem potpoglavlju prilikom prijenosa kôda u Unreal Engine.

4.2. Implementacija u pogonu Unreal Engine

Jedan od glavnih razloga za prebacivanje koda u Unreal Engine jest njegova gotova infrastruktura potrebna za izradu krajolika, koja uključuje dodjelu materijala poput tekstura zemlje, kamenja i pijeska, izradu materijala te učitavanje modela poput drveća i stijena. Iako je algoritam ostao potpuno isti kao u inicijalnoj verziji, ključna razlika je u korištenju *UProceduralMeshComponent*[2][7] unutar Unreal Enginea, kojoj se prosljeđuju podaci o vrhovima koje je potrebno iscrtati. Nakon konverzije koda u Unreal Engine, postignut je rezultat prikazan na sljedećoj slici (Slika 4.4).



Slika 4.4 Prikaz rezultata implementacije algoritma u pogonu Unreal Engine

Čak i nakon prelaska na Unreal Engine, problemi opisani u prethodnom potpoglavlju ostaju prisutni. Tijekom generiranja novih sekcija terena dolazi do zamrzavanja sustava, s vremenom odziva između 60ms i 90ms. U nastavku su opisani postupci implementiranja navedenih optimizacija: asinkrono stvaranje sekcija terena (višedretvenost) i ažuriranja sekcija koje se ne vide umjesto stvaranje potpuno novih sekcija. Nakon opisanih optimizacija, prikazan je postupak dodavanja modela koji su često prisutni u prirodi (poput drveća i stijena), implementacija vode te modifikacija terena ovisno o okolnim vremenskim uvjetima (poput kiše).

4.2.1. Optimizacija: Implementacija asinkronosti (višedretvenost)

Asinkrono izvođenje ostvareno je putem dodatne pomoćne klase *FAsyncWorldGenerator*, koja nasljeđuje klasu *FNonAbandonableTask*. Klasa *FNonAbandonableTask* omogućuje asinkronu izvedbu zadataka uz jamstvo da će se zadatak sigurno dovršiti. U okviru ove klase, potrebno je definirati funkciju *DoWork()* u kojoj se obavlja asinkroni posao. Sljedeći isječak koda prikazuje implementaciju unutar funkcije *DoWork()*:

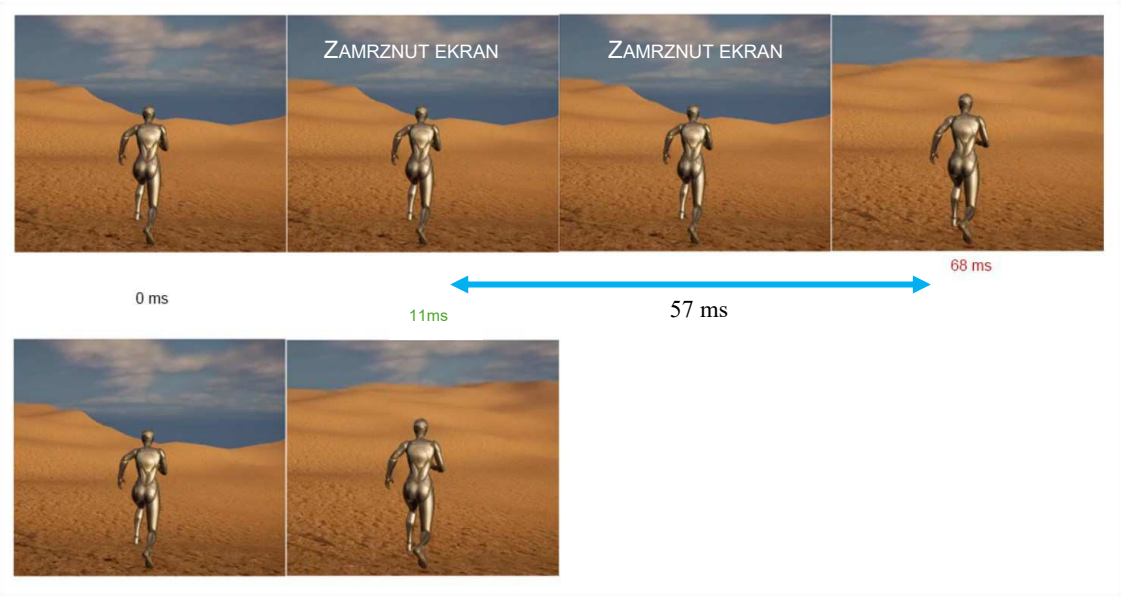
```
void FAsyncWorldGenerator::DoWork(){

      WorldGenerator->GenerateTerrain(
      WorldGenerator->SectionIndexX, WorldGenerator->SectionIndexY);

}
```

GenerateTerrain() funkcija generira sve potrebne informacije za iscrtavanje nove sekcije terena. Varijable *SectionIndexX* i *SectionIndexY* koriste se za izračun pozicija vrhova terena. Po završetku zadatka, dretva obavještava klasu *WorldGenerator* da su podaci spremni za iscrtavanje, nakon čega se ti podaci iscrtavaju.

Korištenje asinkronih zadataka za generiranje vrhova terena znatno poboljšava performanse programa te u potpunosti rješava problem zamrzavanja ekrana (Slika 4.5). Vrijeme potrebno za iscrtavanje nove sekcije terena smanjilo se s ~68 ms na ~11 ms (u prosjeku).



Slika 4.5 Razlika brzine iscrtavanja terena prilikom korištenja asinkronosti (video[3])

4.2.2. Optimizacija: Stvaranja novih sekcija

Još jedna značajna optimizacija koja može znatno poboljšati performanse jest ponovno iskorištavanje i ažuriranje postojećih sekcija terena umjesto stvaranja potpuno novih. Ova metoda temelji se na udaljenosti korisnika od sekcija terena. Ako je korisnik dovoljno udaljen od određenih sekcija (obično ih više ne vidi), te se sekcije mogu ažurirati i premjestiti na novo mjesto. Primjenom ove metode, prosječno vrijeme potrebno za iscrtavanje novih sekcija terena smanjeno je za otprilike 4 ms.

4.2.3. Dodavanje modela iz prirode

Osim stvaranja terena, potrebno je i dodati razne modele za vegetaciju (engl. *foliage*) poput drveća i trave.

Za određivanje lokacija na kojima će se generirati modeli drveća korišteni su već generirani podaci za iscrtavanje terena, tj. pozicije vrhova. Pomoću pseudoslučajnog generatora brojeva

[3] https://youtu.be/j1OTgghV0R8

odlučuje se hoće li se na određenom vrhu terena postaviti model drveta ili ne. Rezultati dodavanja modela prikazani su na sljedećoj slici (Slika 4.6).

Slika 4.6 Prikaz terena s dodanim modelima drveća

Iako teren sada izgleda prirodnije, povećani broj modela koji se trebaju iscrtati uzrokuje znatno usporenje programa (Slika 4.7). Glavni problem leži u tome što modeli drveća nemaju precizno definirane razine detalja[4] (LOD, engl. *level of detail*). Osim razina detalja, potrebno je definirati i udaljenost odstranjivanja[5] (engl. *culling distance*) što će dodatno poboljšati performanse. U nastavku je opisan postupak definiranja novih LOD razina.

Slika 4.7 Performanse prije i nakon dodavanja modela drveća

[4] tehnika smanjenja kompleksnosti modela kako se korisnik udaljava od modela

[5] udaljenost s koje se objekti ili modeli prestaju iscrtavati na ekranu

4.2.3.1 Optimizacija: Definiranje LOD razina i udaljenosti odstranjivanja modela

Unutar pogona Unreal Engine moguće je prilagoditi način određivanja LOD razine (engl. Level of Detail) prema veličini modela na ekranu[6] (engl. screen size). Budući da modeli dodani u aplikaciju sadrže veliki broj detalja (na nultoj LOD razini imaju oko 40.000 poligona), potrebno je postaviti odgovarajuće LOD razine. Kao početna razina odabrana je LOD2, koja pruža dobar omjer između detalja i broja vrhova. Ostale LOD razine učitavaju se u skladu s udaljenošću korisnika od modela.

Dodatna optimizacija uključuje definiranje udaljenosti odstranjivanja. Ova tehnika je posebno korisna pri stvaranju velikog broja instanci jer značajno smanjuje broj potrebnih instanci uklanjanjem onih koje nisu vidljive korisniku.

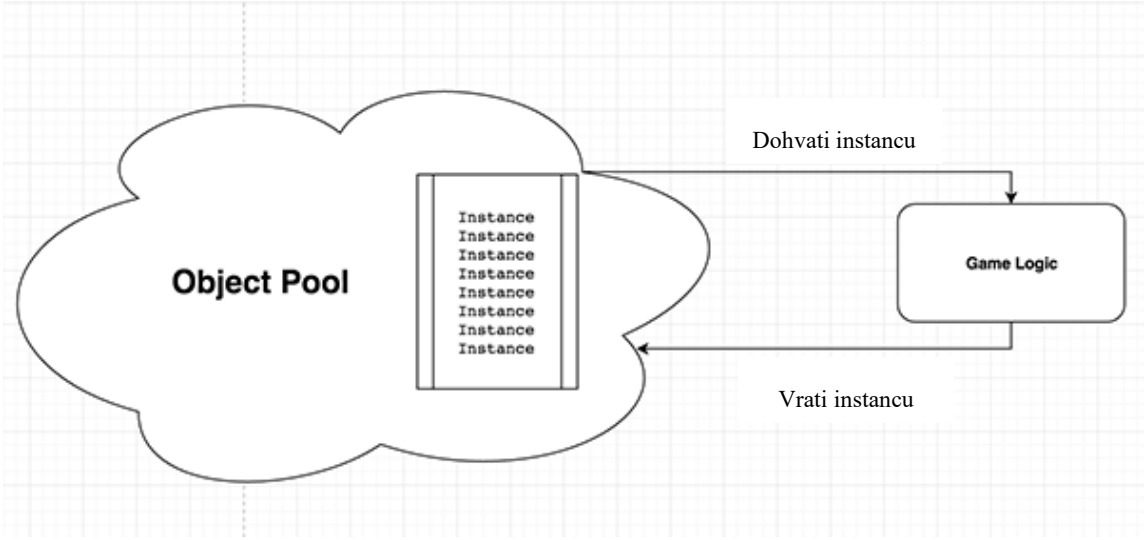
Rezultati provedenih optimizacija prikazani su na sljedećoj slici (Slika 4.8). Kao što se može uočiti, samo podešavanjem LOD razina broj okvira po sekundi (engl. frames per second) se povećao za otprilike 65 %. Uz definiranje udaljenosti odstranjivanja, broj okvira po sekundi povećao se za dodatnih približno 14 %.

Slika 4.8 Prikaz razlike performansi

4.2.3.2 Optimizacija: Korištenje oblikovnog obrasca *object pooling*

Instanciranje velikog broja modela, posebno onih koji se sastoje od velikog broja poligona, zahtjeva dosta računalnih resursa. Kako bi se izbjegao taj slučaj, za instanciranje modela drveća koristi se oblikovni obrazac *object pooling*[4] (Slika 4.9). On omogućava ponovno korištenje objekata koji više nisu potrebni. U slučaju kada se korisnik dovoljno udalji od određenih instanci drveća, te se instance premještaju na novo mjesto umjesto da se stvaraju

potpuno nove instance. Iako će na početku programa i dalje biti potrebno instancirati potrebne modele drveća, kasnije će se koristiti oni modeli koji više nisu potrebni unutar svijeta. Pomoću ove metode više nema zamrzavanja ekrana prilikom prelaska u nove sekcije terena.



Slika 4.9 Princip rada oblikovnog obrasca *object pooling*

4.2.4. Dodavanje modela trave

Za razliku od drveća kojih može biti manji broj, modela trave treba biti puno kako bi teren izgledao realističan. Naime, instanciranje velikog broja modela trave preko cijelog terena je vrlo skupo te može izazvati zamrzavanje ekrana. Stoga u sklopu ovog rada, napravljena je još jedna dodatna komponenta koja omogućava stvaranje objekata samo u bližoj okolini korisnika. Tako će se instancirati samo modeli koje korisnik može vidjeti. Za dodatnu optimizaciju, implementirano je brisanje dalekih instanci modela trave. Tako kada se korisnik dovoljno udalji od određenih instanci, one će nestati.

Apstraktni kod korišten za stvaranje trave nalazi se u nastavku (Kod 4.1):

```
void SpawnObject(LineTrace trace)
{
	if (!trace.HitGround)
		return;

	Transform InstanceTransform;
	for (int FolIndex = 0; FolIndex < FoliageTypes.Num(); FolIndex++)
	{
		FoliageType FolType = FoliageTypes[folIndex];
		// Provjera visine
		if (FolType.HeightMin > trace.height > FolType.HeightMax)
			continue;

		if (FolType.Density < RandomFloat(0.0, 10.0))
			continue;

		// Provjera nagnutosti terena
		float Dot = DotProduct(Hit.ImpactNormal, FVector::UpVector);
		float Slope = FMath::Acos(Dot);

		if (FolType.SlopeMin > Slope > FolType.SlopeMax)
			continue;

		// Pozicija, rotacija, skaliranje na nasumične vrijednosti
		InstanceTransform.SetTransform(…);
		FoliageComponents[FolIndex]->AddInstance(InstanceTransform, true);
	}
}
```

Kod 4.1 Stvaranje objekata u okruženju korisnika

Kao parametar se daje varijabla *trace* koja sadrži informacije o tome koji je tip površine na koji se stvara objekt, na kojoj je visina površina, koja je nagnutost površine i slično. Polje *FoliageTypes* sadrži sve tipove trave koje korisnik želi instancirati. Prije stvaranja objekta potrebno je provjeriti za taj tip objekta jesu li svi uvjeti zadovoljeni (navedeni prilikom opisa varijable *trace*). Tek nakon što je ustanovljeno da su svi uvjeti zadovoljeni, objekt se može stvoriti na toj lokaciji.

Apstraktni kod za spremanje objekata za ponovno korištenje nalazi se u nastavku (Kod 4.2):

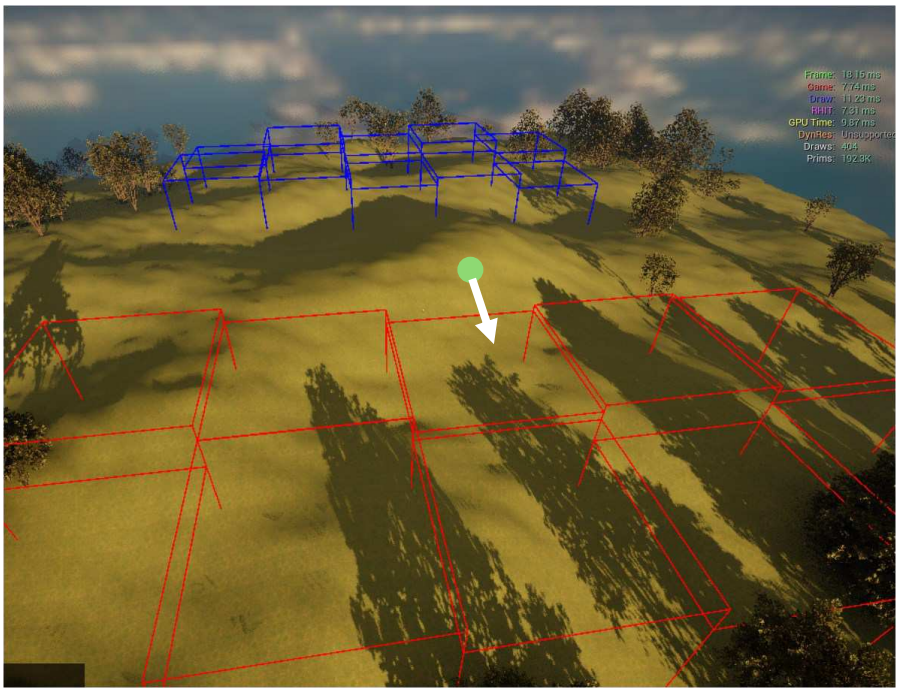
```
void RemoveTileObjects(FVector TileCenter)
{
	Vector Min = Vector(TileCenter + (-0.5f) * FVector::One() * CellSize);
	Vector Max = Vector(TileCenter + 0.5f * FVector::One() * CellSize);
	FBox Box(Min, Max);
	Min.Z = TileCenter.Z - TraceDistance;
	Max.Z = TileCenter.Z + TraceDistance;

	for (int FolIndex = 0; FolIndex < FoliageTypes.Num(); FolIndex ++)
	{
		auto FoliageType = FoliageTypes [FolIndex];
		TArray<int> Instances = FoliageComp.GetInstancesOverlappingBox(Box,
                                                                       true);
		if (Instances.Num() > 0)
		{
			FoliageType->RemoveInstances(Instances);
		}
	}
}
```

Kod 4.2 Brisanje objekata koji korisnika više nisu vidljivi

Instance modela trave dobivaju se pomoću funkcije *GetInstancesOverlappingBox()* koja vraća sve instance unutar definiranog kvadra. Potom se sve instance brišu pomoću funkcije *RemoveInstances()*.

Dodatan opis rada algoritma opisan je u nastavku te je prikazan na sljedećoj slici (Slika 4.10).



Slika 4.10 Prikaz stvaranja i brisanja modela

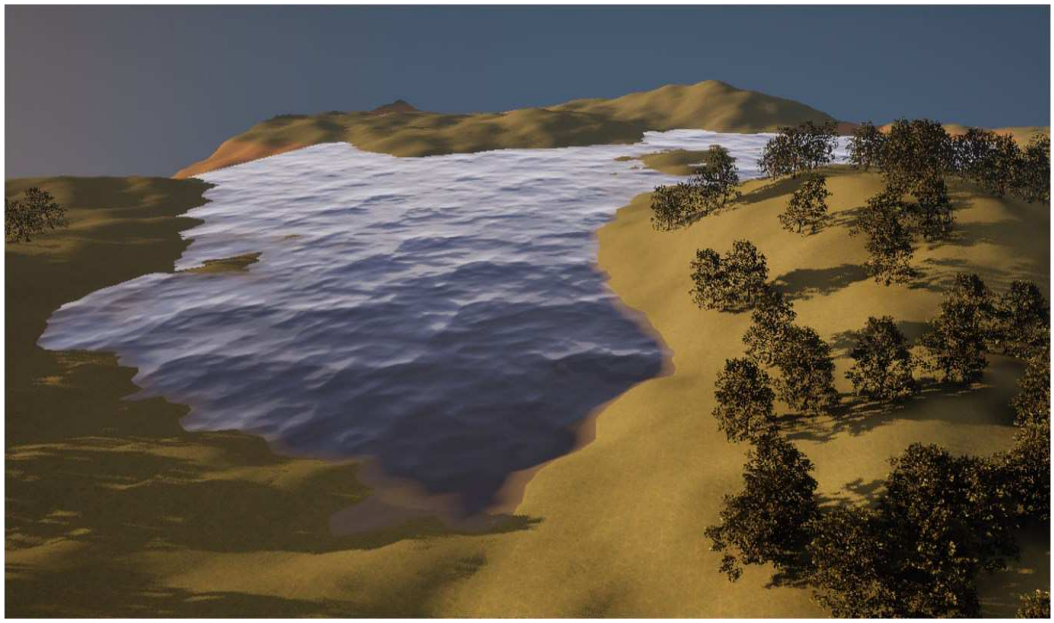
Na slici, korisnik se kreće prema dolje i kako se kreće prema dolje, ispred njega se stvaraju nove instance trave (prikazano crvenim kvadrima). U isto vrijeme, iza njega se brišu instance koje više nisu vidljive njemu (prikazano plavim kvadrima).

Rezultati nakon dodavanja modela trave prikazani su na sljedećoj slici (Slika 4.11).

Slika 4.11 Rezultati terena nakon dodavanja trave

4.2.5. Dodavanje vode prilikom generiranja terena

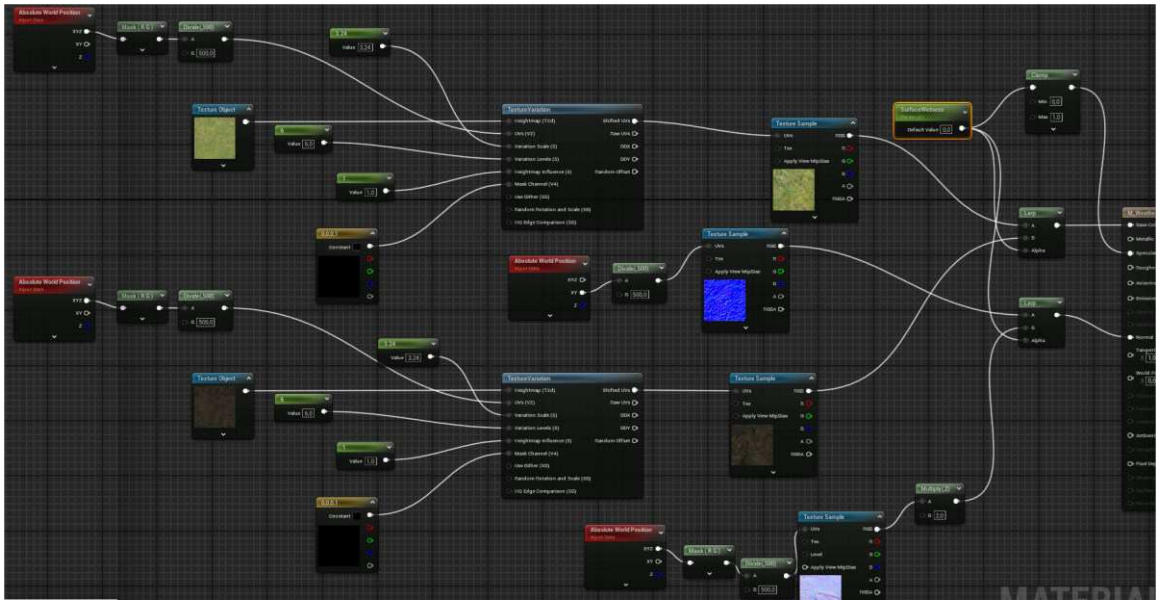
Budući da je program namijenjen za generiranje terena većih dimenzija, za vodu se koristi ravna površina koja ima dodijeljen materijal za vodu[8]. Materijal za vodu sadrži teksturu koja se kroz vrijeme pomiče i tako simulira izgled vode. Na ovaj način moguće je dodati vodu prilikom generiranja terena bez prevelikog korištenja računalnih resursa. Alternativno, mogla bi se primijeniti metoda simulacije vode, no ona bi zahtijevala znatno više računalnih resursa. Izgled terena s vodom prikazan je na sljedećoj slici (Slika 4.12).



Slika 4.12 Prikaz terena s vodom

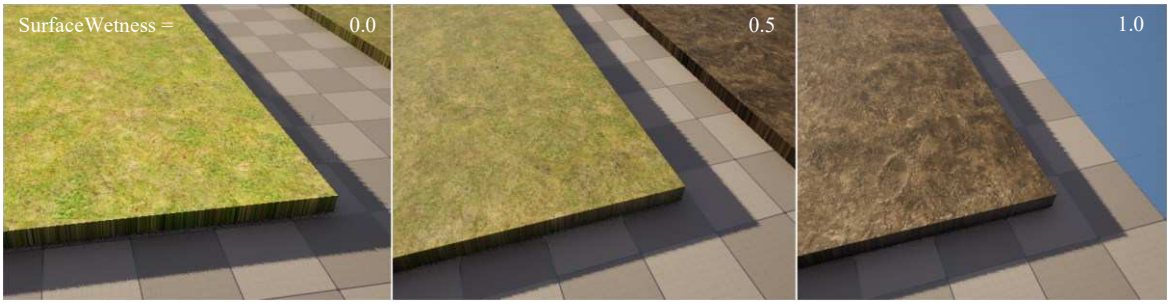
4.2.6. Implementacija utjecaja vremenskih uvjeta na površinu terena

Za implementaciju utjecaja vremenskih uvjeta (u ovom slučaju kiše), koristi se modificiranje materijala za vrijeme izvođenja. Naime, Unreal Engine omogućuje modificiranje instance materijala objekta za vrijeme izvođenja programa bez utjecaja na druge objekte koji sadrže isti materijal. Ova karakteristika omogućuje stvaranje terena s različitim razinama vlažnosti. Prikaz materijala u uređivaču čvorova za materijale (engl. *node editor*) nalazi se na sljedećoj slici (Slika 4.13).



Slika 4.13 Kalkulacije teksture unutar materijala

Promjenom parametra *SurfaceWetness* moguće je konfigurirati koliko će teren biti mokar. Na sljedećoj slici (Slika 4.14) prikazan je teren s različitim vrijednostima navedenog parametra.



Slika 4.14 Teren s različitim vrijednostima parametra *SurfaceWetness*

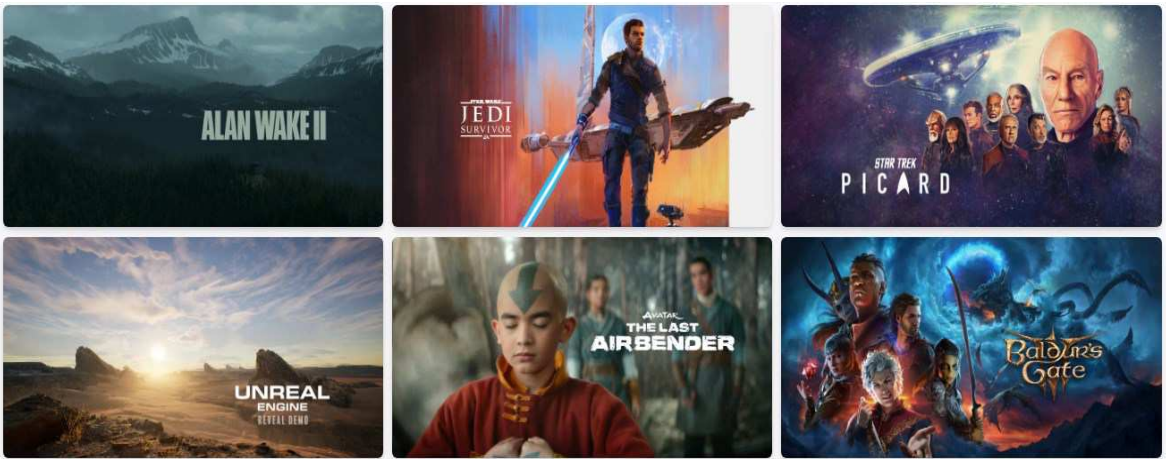
Kao što je moguće vidjeti na slici, za male vrijednosti materijal nema odsjaj te ima teksturu trave. Za veće vrijednosti, teren dobiva odsjaj te se pretvara u teksturu blata.

4.3. Ostali alati za proceduralno generiranje terena

Postoje mnogi različitih alata koji služe za proceduralno kreiranje terena. U nastavku su opisani alati te se spominju različite primjene navedenih alata.

4.3.1. QuadSpinner Gaea

Što se tiče proceduralnog generiranja terena, neizostavno je spomenuti Gaea. Gaea nudi širok spektar mogućnosti, poput generiranja tekstura, izvedbu promjene nad već generiranim terenom, promjene materijala u skladu s okolnim uvjetima, i mnoge druge opcije. Zbog svoje iznimne prilagodljivosti, Gaea se koristi u različitim medijima poput filmova i video igara. Program omogućuje brzo stvaranje vrlo prirodnih terena, no ima poteškoće s kreiranjem iznimno velikih terena (rezolucija iznad 20,000x20,000). Primjeri filmova i video igara koji koriste Gaeu prikazani su na sljedećoj slici (Slika 4.15).



Slika 4.15 Mediji koji koriste Gaeau za proceduralno generiranje terena

4.3.2. Houdini

Houdini je program za izradu 3D animacija i vizualnih efekata. Primarno se koristi za izradu vizualnih efekata za filmsku i televizijsku produkciju, no zahvaljujući različitim mogućnostima proceduralnog generiranja (i terena) u novijim verzijama, krenuo se koristiti i u video igrama. Neki od poznatih studija koji koriste Houdini su: Disney, Pixar i Dreamworks Animation. Filmovi u kojima se koristi Houdini prikazani su na sljedećoj slici (Slika 4.16).

Slika 4.16 Primjena programa Houdini u filmovima

4.3.3. World Machine

Iako je manje popularan od programa Gaea i Houdini, World Machine je još jedan vrlo fleksibilan program za izradu proceduralno generiranog terena koji također ima mnogo mogućnosti poput određivanja razmještaja zemlje i vode na terenu, sustav za simulaciju vode, mogućnost podjele terena u odjeljke (korisno prilikom učitavanja velikom terena). Za razliku od Gaea, daje više kontrole nad terenom. Za izradu kvalitetnog terena, potrebno je dobro poznavanje principa izrada terena. Isto tako, pomoću World Machinea moguće je napraviti i veće terene bez poteškoća.

4.3.4. Pogoni za izradu video igara

Pogoni većinom nemaju izravnu podršku za izrađivanje proceduralnog terena. Iako Unreal Engine sadrži klase potrebne za proceduralno generiranje terena (poput *UProceduralMeshComponent*, *UDynamicMeshComponent*, *RealtimeMeshComponent*)*[3]*, ne postoji mogućnost generiranja terena bez učitavanja dodataka (engl. *plugin*).

Uz popularne pogone za izradu igara, postoje mnogo manjih pogona među kojima je i Bevy Engine. Bevy Engine je pogon pisan u programskom jeziku Rust koji je još uvijek u ranoj fazi razvoja, ali sadrži dosta funkcionalnosti potrebno za proceduralno generiranje terena.

4.3.5. Ostali

Uz sve navedene programe, potrebno je i istaknuti Blender. Zahvaljujući svojoj velikoj zajednici, postoji jako veliki broj dodataka, ali i podrške za proceduralno generiranje terena, materijala, vizualni efekata i mnogih drugih efekata. Pomoću geometrijskih čvorova (engl. *geometry nodes*) moguće je napraviti modularne i vrlo fleksibilne terene koji se mogu definirati promjenom vrijednosti parametara. Primjeri terena generiranih pomoću geometrijskih čvorova u Blenderu prikazani su na sljedećoj slici (Slika 4.17).

Slika 4.17 Teren generiran pomoću geometrijskih čvorova unutar Blendera

5. Pregled aplikacije

U ovom poglavlju predstavljene su sve mogućnosti projekta te način pokretanja projekta. Budući da je projekt napravljen pomoću pogona Unreal Enginea, za pokretanje projekta potrebno je imati instaliran Unreal Engine. Također se može pokrenuti aplikacija od projekta koja ima predefinirane vrijednosti za generiranje terena koje nije moguće mijenjati za vrijeme izvođenja programa.

Za početak, sve potrebne datoteke nalaze se na repozitoriju projekta. Nakon preuzimanja projekta, potrebno je pokrenuti projekt otvaranjem datoteke *WorldGenerator.uproject*. Nakon otvaranja datoteke, pokrenut će se Unreal Engine i u njemu učitan projekt. Nakon učitavanja, moguće je vidjeti sve postavljene objekte u svijetu unutar grafa scene (u Unreal Engine se zove *World Outliner*). Ključni objekti koji su korišteni za ovaj projekt su: *BP_World* i *BP_GrassSpawner*. *BP_World* služi za proceduralno generiranje terena, a *BP_GrassSpawner* služi za generiranje trave u okruženju korisnika.

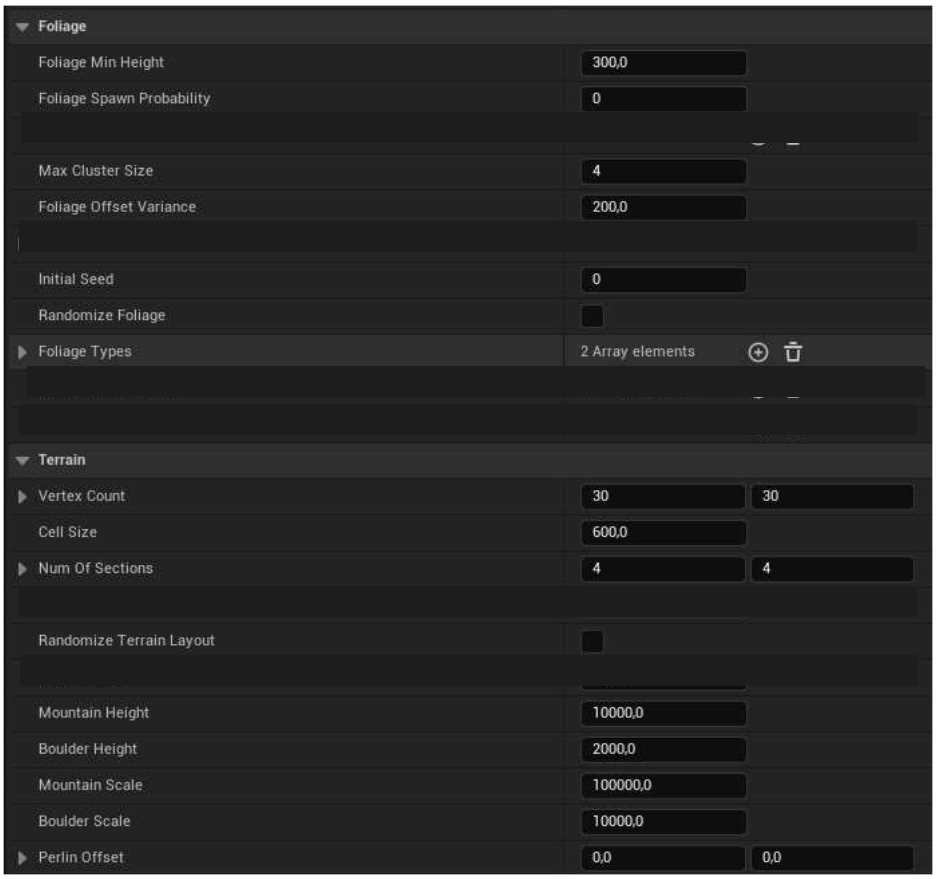
5.1. BP_World: generator terena

Na sljedećoj slici (Slika 5.1) prikazani su svi parametri koji se mogu modificirati za proceduralno generiranje terena.

U nastavku su opisane uloge svakog parametra:

- *Foliage Min Height* – minimalna visina na kojoj se stvaraju modeli (drveće, trava)
- *Foliage Spawn Probability* – vjerojatnost da će se na pojedinačnoj lokaciji (vrhu) stvoriti objekt
- *Max Cluster Size* – maksimalan broj objekata na određenoj lokaciji
- *Foliage Offset Variance* – varijanca za udaljenost između objekata
- *Initial Seed* – služi za podešavanje pseudoslučajnog generatora
- *Randomize Foliage* – zastavica; ako je uključena, prilikom svakog pokretanja stvara se drugačiji raspored objekata
- *Foliage Types* – tipovi objekata koji se stvaraju (mogu se staviti bilo koji objekti)
- *Vertex Count* – broj vrhova za svaku sekciju (po X i Y osi)
- *Cell Size* – veličina jedne ćelije sekcije terena

- *Num Of Sections* – broj sekcija terena (po X i Y osi)
- *Randomize Terrain Layout* – zastavica; ako je uključena, prilikom svakog pokretanja se stvara novi teren
- *Mountain/Boulder Height* – koristi se za definiranje visine terena (prilikom izračuna šuma)
- *Mountain/Boulder Scale* - koristi se za definiranje visine terena (prilikom izračuna šuma)
- *Perlin Offset* – pomak šuma (po X i Y osi)

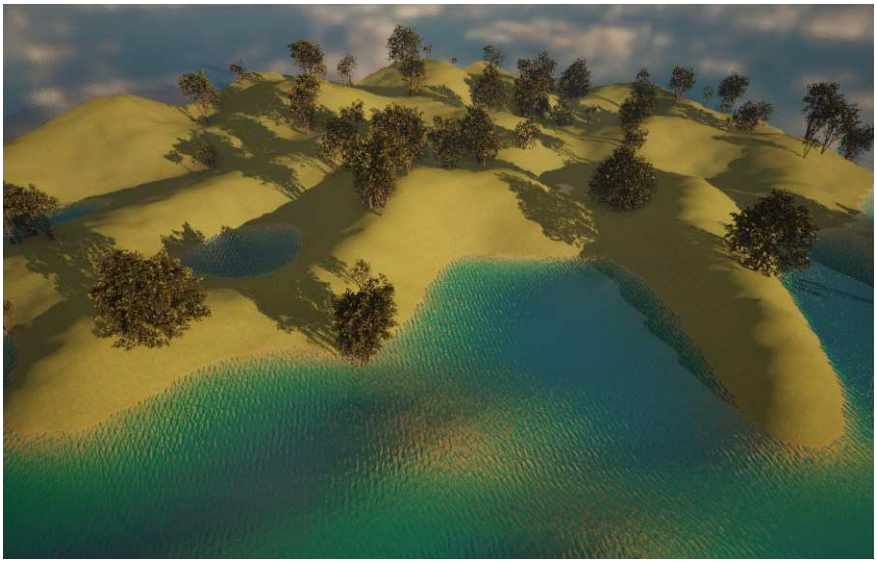


Slika 5.1 Parametri za proceduralno generiranje terena

Modificiranjem navedenih parametara mogu se dobiti različite kombinacije terena. Nekoliko primjera prikazano je na sljedećim slikama (Slika 5.2, Slika 5.3, Slika 5.4).

Slika 5.2 Primjer generiranog terena 1

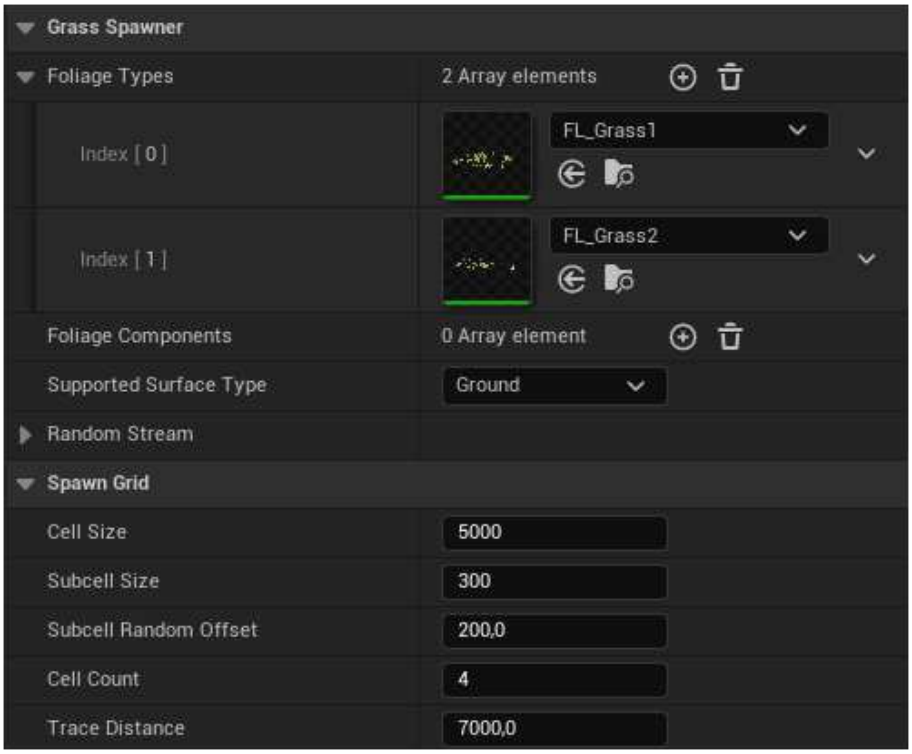
Slika 5.3 Primjer generiranog terena 2



Slika 5.4 Primjer generiranog terena 3

5.2. BP_GrassSpawner: generator trave

BP_GrassSpawner služi za dinamično stvaranje trave u blizini korisnika. Postoji nekoliko parametara koji omogućavaju modificiranje načina stvaranje trave. Na sljedećoj slici (Slika 5.5) prikazani su svi parametri za *BP_GrassSpawner*.



Slika 5.5 Parametri za generiranje trave

U nastavku su opisani parametri za generiranje trave:

- *Foliage Types* – definirani tipovi objekta (mogu biti bilo koji objekti)
- *Cell Size* – veličina čelije unutar koje se stvaraju modeli trave (Slika 4.10)
- *Subcell Size* – veličina podćelija unutar kojih se stvaraju modeli trave
- *Subcell Random Offset* - nasumičan pomak za svaku instancu modela trave
- *Cell Count* – broj ćelija (jednak po X i Y osi)
- *Trace Distance* – služi za izračun potrebnih podataka za instanciranje trave na površini terena

Unutar svake podćelije se stvaraju instance trave. Ukoliko je potrebno imati više instanca trave moguće je povećati veličinu ćelije ili smanjiti veličinu podćelije (ukoliko je potrebno da trava bude gušće raspoređena).

Na sljedećim slikama (Slika 5.6, Slika 5.7) moguće je vidjeti različite konfiguracije ovisno o odabranim parametrima.

Slika 5.6 Primjer generirane trave (rijetko raspoređena)

Slika 5.7 Primjer generirane trave (gusto raspoređena)

Potrebno je pripaziti prilikom određivanja parametara. Ako se definira premali broj za parametar *Subcell Size*, dolazi do privremenog zamrzavanja programa zbog velikog broja instanci na malom prostoru. Potrebno je pronaći optimalan broj ovisno o potrebama programa.

6. Dodatna poboljšanja

U ovom poglavlju navedena se moguća poboljšanja koja bi dodatno unaprijedila trenutnu iteraciju programa.

6.1. Unaprijeđeni materijali za površinu terena i modele

Iako teren trenutno reagira na kišu, dodatna nadogradnja bi bila nadopuna materijala da može reagirati i na druge vremenske uvjete (snijeg, vodu iz okoline, …). Materijali za modele bi se također mogli unaprijediti da reagiraju na kišu, snijeg i vjetar. Trenutno samo drveće reagira na vjetar.

6.2. Dodatno poboljšanje LOD sustava

U trenutnoj iteraciji, ako korisnik gleda drvo koje je jako daleko, i dalje se prikazuje relativno detaljan prikaz drveta. Na određenoj udaljenosti, bilo bi pogodno zamijeniti model drveta s teksturom koja imitira drvo. Korisnik s velike udaljenosti ne može vidjeti detalje drveta, stoga radi boljih performansi bi trebalo koristiti teksture na većim udaljenostima. Također, prelazak na Nanite[6] sustav bi moglo dodatno poboljšati performanse.

6.3. Definiranje parametara prilikom pokretanja aplikacije u komandnoj liniji

Trenutno, za modificiranje parametara generatora terena i trave potrebno je imati instaliran cijeli pogon Unreal Engine (oko 40 GB). Nakon što se kompilira aplikacija, veličina aplikacije je oko 2 GB. Bilo bi pogodno kada bi korisnik mogao mijenjati navedene parametre putem komandne linije. Time bi korisnik mogao instalirati samo aplikaciju (samo 2 GB) umjesto cijelog pogona za pokretanje projekta te bi i dalje imao mogućnost promjene konfiguracije generatora terena i trave.

[6] poseban način iscrtavanja[9] u pogonu Unreal Engine; često korišten za iscrtavanje vizualnih efekata

Zaključak

Istraživanje proceduralne generiranja beskonačnih terena u Unreal Engineu pokazalo je mogućnosti i izazove ovog pristupa u suvremenom razvoju igara i simulacijskih okruženja. Uz pomoć naprednog pogona Unreal Engine i mogućnosti proceduralnih algoritama, moguće je kreirati velika, dinamična okruženja koja ne samo da poboljšavaju vizualnu kvalitetu, već i značajno unapređuju učinkovitost i skalabilnost generiranja terena.

Jedna od glavnih prednosti proceduralnog generiranja terena je sposobnost generiranja gotovo neograničen broj raznolikih terena s minimalnim računalnim resursima, zahvaljujući upotrebi funkcija šuma, prostornih struktura podataka i višedretvenosti. To omogućuje besprijekorno proširenje svjetova igara i simulacija bez unaprijed definiranih granica, što je posebno korisno u igrama otvorenog svijeta i velikim simulacijama gdje su istraživanje svijeta i raznolikost ključni elementi.

Međutim, ovaj rad također ističe nekoliko izazova koji su svojstveni proceduralnoj generaciji beskonačnih terena. Upravljanje performansama, osiguravanje prihvatljivog broja okvira po sekundi i brzo učitavanje i dalje predstavljaju veliki izazov, pogotovo prilikom generiranja vrlo detaljnih okruženja. Također, postizanje ravnoteže između slučajnosti i realizma ključno je kako bi se spriječilo generiranje terena koji izgledaju umjetno ili ponavljajuće.

Zaključno, proceduralna generiranje beskonačnih terena ima velike prednosti za generiranje dinamičkih okruženja u Unreal Engineu. Ono omogućuje stvaranja složenih i prilagodljivih krajolika uz minimalne računalne resurse. Iako postoje tehnički izazovi, poput održavanja performansi i balansiranja između slučajnosti i realizma, potencijal ove tehnologije za daljnje inovacije i poboljšanja je velik.

Literatura

[1] Sainio N., *Terrain Generation Algorithms*, 2023.

[2] Epic Games, C++ API Reference, https://dev.epicgames.com/documentation/en-us/unreal-engine/API

[3] Epic Games, Unreal Engine Source Code, https://github.com/EpicGames/UnrealEngine

[4] Nystrom B., *Object Pool*, https://gameprogrammingpatterns.com/object-pool.html

[5] Kniberg H., Reinventing Minecraft world generation, https://www.youtube.com/watch?v=ob3VwY4JyzE

[6] Unreal Gems, *Levels of Detail (LOD) In-Depth*, https://dev.epicgames.com/community/learning/tutorials/EdB4/unreal-engine-levels-of-detail-lod-in-depth

[7] CodeLikeMe, Unreal Engine 5 - Procedural Terrain Generation, https://www.youtube.com/watch?v=3IGHI50NH1U

[8] Epic Games, Materials, https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-materials

[9] Epic Games, Nanite Virtualized Geometry, https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-virtualized-geometry-in-unreal-engine

Sažetak

Proceduralno generiranje beskonačnih terena

Ovaj rad istražuje upotrebu proceduralnog generiranja za stvaranje beskonačnih terena, tehnike koja omogućuje stvaranje prostranih i dinamičnih okruženja. Za implementaciju se u početku koristi OpenGL, a naknadno se kod nadograđuje u pogonu Unreal Engine. Ispituju se prednosti korištenja funkcija šuma i prostornih struktura podataka za generiranje raznolikih krajolika. Prilikom implementacije primjenjuju se različite tehnike optimizacije poput razina detalja (LOD) i udaljenosti odstranjivanja kako bi se dodatno poboljšalo izvođenje programa. Kroz implementaciju i evaluaciju ovih tehnika, rad pokazuje mogućnosti za proceduralno generiranje beskonačnih terena te navodi moguće primjene u različitim medijima poput video igara i filmova.

Ključne riječi: proceduralno generiranje, beskonačni tereni, funkcije šuma, dinamična okruženja, optimizacija

Summary

Procedural generation of infinite terrains

This thesis explores the use of procedural generation for creating infinite terrains, a technique that enables the development of expansive and dynamic environments. The initial implementation is created in OpenGL, which is subsequently upgraded in the Unreal Engine framework. The thesis examines the advantages of using noise functions and spatial data structures for generation of diverse landscapes. Various optimization techniques, such as level of detail (LOD) and distance culling, are applied to further improve program performance. Through the implementation and evaluation of these techniques, the thesis demonstrates the potential for procedural generation of infinite terrains and outlines possible applications in various media, such as video games and films.

Keywords: procedural generation, infinite terrains, noise functions, dynamic environments, optimization

Privitak

Instalacija programske podrške

Za instalaciju programske podrške potrebno je imati instalirani pogon Unreal Engine. Nakon instalacije pogona, moguće je pokrenuti projekt dvostrukim klikom na *.uproject* datoteku.

Sve datoteke projekta nalaze se na sljedećoj poveznici: https://gitlab.com/Wuffll/ue5-world-generation

Izvorni kod inicijalne implementacije (OpenGL) moguće je pronaći na sljedećoj poveznici: https://github.com/Wuffll/masters-thesis

Karakteristike računala

U nastavku su navedene specifikacije računala koje je korišteno za mjerenje performansi u ovom radu:

- CPU – AMD Ryzen 5 5600
- GPU – AMD Radeon RX 6700 XT
- RAM – 16 GB DDR4 3000 MHz