Proceduralno generiranje terena

Mužar, Lovro

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1677

PROCEDURALNO GENERIRANJE TERENA

Lovro Mužar

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1677

PROCEDURALNO GENERIRANJE TERENA

Lovro Mužar

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1677

Pristupnik:	**Lovro Mužar (0036543476)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Marko Čupić

Zadatak: **Proceduralno generiranje terena**

Opis zadatka:

Postupci za proceduralno generiranje terena (primjerice dvodimenzijske mape površine grada, mape zemljopisnih krajolika koji uključuju planine, nizine te vodene površine i slično) su interesantni jer su preduvjet za izradu igara koje igračima omogućavaju neograničeno igranje. Pri tome postoje različiti pristupi koji rade u dvije odnosno u tri dimenzije. U okviru ovog rada potrebno je odabrati jedan pristup za proceduralno generiranje terena, opisati ga, napraviti prototipnu računalnu implementaciju istoga te prikazati dobivene rezultate. Radu priložiti izvorni i izvršni kod, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 14. lipnja 2024.

Zahvaljujem se mentoru dr. sc. Marku Čupiću na pomoći i savjetima tokom izrade rada te kolegi Luki Grubišinu za odabir nasumičnog broja.

Sadržaj

1. Uvod

Bilo u igricama ili u filmovima, proceduralno generirani tereni sve češće su prisutni kao pozadinski, skriveni dio naše svakodnevne zabave. Cilj ovog rada je razmotriti i pokazati širok niz mogućnosti proceduralnog generiranja terena te metoda kojima se ostvaruje te pokazati mogućnost kompleksnog sustava za proceduralno generiranje terena na temelju jednostavnog primjera.

Na početku ćemo dati pregled proceduralnog generiranja općenito te njegove češće primjene.

Nakon toga ćemo opisati proceduralno generiranje terena. Prvo ćemo proći kroz njegove najčešće primjene i koristi, a zatim kroz vrste terena koje se mogu generirati. Zatim ćemo dati opis nekih od metoda kojima se postiže proceduralno generiranje raznih vrsta terena.

Konačno dat ćemo opširni opis demonstracijskog primjer temeljen na Perlinovom šumu, kao dokaz koncepta za kompleksniji sustav. Pokazat ćemo kako se tim, relativno jednostavnim algoritmom, može postići proceduralno generiranje terena s velikim brojem mogućnosti za upravljanje željenim rezultatom.

2. Proceduralno generiranje

Proceduralno generiranje u računalnoj znanosti označava postupke algoritamskog stvaranja podataka na temelju određenih pravila, ulaznih parametara i nasumično generiranih vrijednosti, često nazvanih šumom.

Uglavnom se primjenjuje u području računalne grafike za generiranje i dorađivanje tekstura, modela, animacija, krajolika. Također se može koristiti i za generiranje zvukova. Jedna od najčešćih primjena proceduralnog generiranja je pri izradi videoigara. Osim za već spomenute svrhe može se koristiti i za generiranje terena, o čemu ćemo više reći kasnije. Još jedna od mogućih primjena je generiranje podataka koji čine pozadinu igrice. Stvari poput vremenskih pojava ili čak čitave povijesti svijeta kao što je to na primjer slučaj u igrici Ultima Ratio Regum. U njoj je proceduralno generiranje u toliko širokoj uporabi da je bilo opisano kao „ništa manje no proceduralno generiranje kulture“ ("nothing short of the procedural generation of culture") (2015.[1])

Velika prednost pristupa proceduralnom generiranju leži u mogućnosti stvaranja gotovo beskonačnog broja jedinstvenih stvari uz minimalnu potrebu za ljudskim radom. Tako možemo, na temelju ljudski modeliranog objekta, stvoriti koliko nam je potrebno sličnih, ali ipak malo različitih objekata. Tako možemo, na primjer, na temelju par drveća generirala čitava šuma.

Proceduralno generiranje također ima široku i čestu primjenu u filmskoj industriji. Alati poput MASSIVE-a koji su korišteni da se stvore epske i fantastične bitke u filmovima poput Gospodar prstenova.

Proceduralno generiranje omogućilo je postavljanje i animiranje tisuća vojnika odjednom umjesto ručnog postavljanja jednog po jednog (2.1.). Također u filmovima poput Tron gdje su razvijene nove tehnike za proceduralno generiranje tekstura pomoću šuma.

Pri kreiranju animiranih filmova, poput Pixarovih, također se koristi proceduralno generiranje za najraznolikije svrhe, kao na primjer stvaranje volumenski prikaza likova u filmu Elemental (2.2.).

Slika 2.1. Primjer generiranja scene iz Gospodara prstenova (2015.[2])

Slika 2.2. Prikaz proceduralno generiranih likova od vatre u filmu Elemental (Rice, 2023.[3])

Osim za igrice i filmove proceduralno generiranje je iznimno korisno i kada se rade simulacije stvarnih situacija. Isto kao što može generirati teren za igrice može ga generirati i za alate poput simulatora leta. No može i puno više od toga. Proceduralnim generiranjem možemo simulirati i razne uvjete leta poput vremenskih nepogoda i kvarove zrakoplova, kako bi se piloti mogli pripremni na djelovanje i u situacijama za koje se ne bi mogli pripremati u stvarnim letovima.

Jedna od modernijih prednosti u području proceduralnog generiranja je uporaba umjetne

inteligencije. Umjesto da se algoritmi proceduralnog generiranja određuju ljudskim radom, možemo istrenirati umjetnu inteligenciju da, na temelju određenih podataka, odredi kako proceduralno generirati neku pojavu. Tako možemo postići i proceduralno generiranje stvari koje su iznimno kompleksne i za koje je osmišljanje algoritma generiranja dugotrajan i nimalo jednostavan proces.

3. Generiranje terena

3.1. Primjena generiranja terena

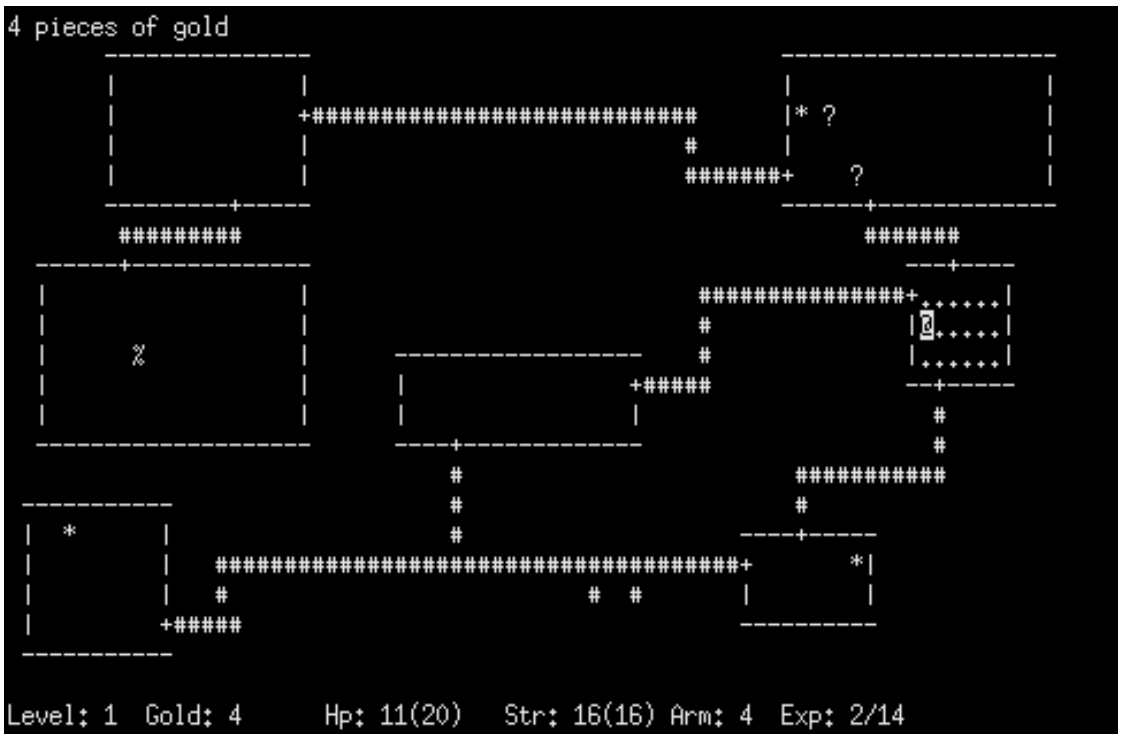
Kao što smo prije spomenuli, jedna od koristi proceduralnog generiranja je upravo proceduralno generiranje terena. Problem na koji mnogi naiđu kada pokušaju napraviti igru ili napisati priču je pitanje gdje se radnja odvija. To nije trivijalan problem. Izmišljanje svijeta je težak i dugotrajan posao koji zahtjeva mnoge vještine i znanja, no još više, zahtjeva i mnogo vremena. Naravno ako uz to još i proizvodite na primjer igru, time zadatak postaje još kompleksniji.

U daljnjem ćemo tekstu govoriti o tome kako se taj problem rješava proceduralnim generiranjem. Kako je velik broj primjera upravo vezan uz videoigre bitno je naglasiti da se za nas pojam terena ne odnosi samo na reljef otvorenog prostora, već i na zatvorene prostore, a ponekad ćemo se osvrnuti i na sadržaj tog prostora. Također, kako se u igricama često koristi izraz mapa i mi ćemo se ponekad poslužiti tim nazivom umjesto termina teren.

Razumljivo, mogućnost stvaranja i korištenja beskonačnih terena je vrlo primamljiva za ljude koji rade igre. Pri tome proceduralno generiranje im omogućava da generiraju mape za svoje igre u puno većem broju i brže nego što bi to ikad mogli ručno. Postupci proceduralnog generiranja terena prvi su se pojavili u igrama igranja uloga (eng. role-playing game, RPG). Jedan od najpoznatijih primjere je svakako igra The Elder Scrolls: Daggerfall (1996.). Pri stvaranju Daggerfalla, Bethesda je koristila proceduralno generiranje kako bi stvorila čitavu mapu, koja je bila veličine usporedive s Velikom Britanijom (Bethesda, 2004.[4]) i godinama je nosila titulu "najveće" videoigre. Uz to proceduralno generirani su i velika većina gradova, šuma, i špilja koje je igrač mogao istraživati, kao i neigrivi likovi (eng. non-player character, NPC). Međutim kao posljedica pokazuje se

i jedna od mana proceduralnog generiranja. Naime zbog svoje skale i relativno uskih mogućnosti algoritma, većina proceduralno generiranih gradova, iako u pojedinostima različiti, bili su iznimno slični i repetitivni.

Još jedan žanr igrica u kojem je iznimno prisutno proceduralno generiranje terena su takozvane roguelike igrice. Naziv im dolazi od toga što dijele sličnosti sa ili su inspirirane igrom Rogue. U Rogueu se pri svakom pokretanju nove igre kreira potpuno novi svijet koji uvijek sadrži nove špilje, nova blaga i nove protivnike koje igrač može otkriti. To je postignuto tako da je gotovo svaki aspekt igre proceduralno generiran pri pokretanju igre s novim likom. Rezultat je da je svako igranje igre novo i jedinstveno iskustvo te ne postoji nikakvo ograničenje u smislu koliko puta se igra može iznova pokrenuti, a da igrač svaki puta ima novo iskustvo.



Slika 3.1. Proceduralno generirana mapa igre Rogue prikazana u konzoli [5]

Proceduralno generiranje također je dio najprodavanije svjetske igre (za vrijeme pisanja ovog rada 2024.), Minecraft. Minecraft je igra iz 2011. godine, u kojoj se prije pokretanja igre mora stvoriti svijet. Nakon toga igrač može ući u taj svijet te u njemu može istraživati, sakupljati resurse, graditi, itd. Sam taj svijet je proceduralno generiran te sadrži iznimno velik broj raznolikih terena. U njemu nalazimo pustinje planine, špilje i podzemlja, mora, tundre, močvare i mnoge druge biome, sve generirane istim algorit-

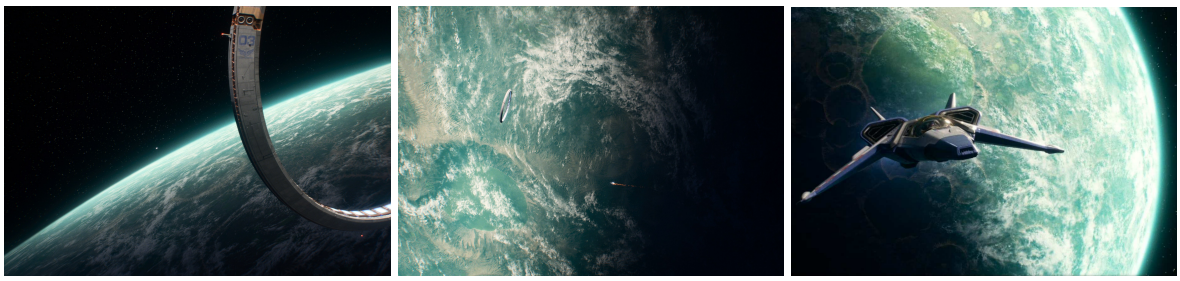
mom proceduralnog generiranja. Cijeli algoritam prima kao ulaz samo jedan broj (seed). Dakle, to znači da postoji 2^{64} mogućih svjetova u kojima se može igrati ova igra.

No Minecraft nije jedina igra koja pri svojem pokretanju generira čitav svijet. Dwarf Fortress je igra, iz 2006. godine u kojoj, isto kao i u Minecraftu, igrač prije pokretanja mora generirati svijet u kojem će igrati. No ovdje se, osim za generiranje samoga svijeta i njegove geografije, proceduralno generiranje koristi i za generiranje stotina godina povijesnih događaja, ljudi te njihovih osobnosti, vrlina i mana, za svakog od likova koje igrač može susresti u igri.

Jedan od modernijih primjera korištenja proceduralnog generiranja za kreiranje terena je igrica No Man's Sky od proizvođača Hello Games iz 2016 godine. U ovoj su igri, korištenjem proceduralnog generiranja, generirani planeti. Za svaki planet je korišten drugi seed za algoritam. Time je to najveća igra u povijesti, gdje igrač može, leteći ili pješice, istraživati 18 trilijuna planeta. Svaki od njih ima svoju jedinstveno generiranju geografiju, vrijeme, klimu, floru i faunu.

Osim u izradi igrica proceduralno generiranje terena ima primjene i u drugim područjima. Već prije spomenuta mogućnost korištenja u simulacijama za treniranje upravljanja avionom čini ga jako korisnim alatom u aeronautici, ali, iz istih razloga, i u vojnoj industriji.

Proceduralno generiranje terena također se pojavljuje i u filmskoj industriji. Može se koristiti za generiranje ili nadograđivanje krajolika, kao na primjer za oblikovanje planeta Pandora u filmu Avatar Jamesa Camerona. Također ima i široku primjenu u izgradnji animiranih filmova. Pixar je u svojem filmu Lightyear (2022.) proceduralno generirao reljef i oblake čitavog planeta kako bi ga prikazao iz svemira (3.2.).



Slika 3.2. Proceduralno generiran planet gledan iz svemira(Murry, 2022.[6])

Danas postoje i mnogi programi, kao što je primjerice Terragen koji pružaju korisni-

cima sučelje visoke razine za generiranje terena u opće svrhe. Takvi programi mogu se služiti i kombinacijom više različitih metoda proceduralnog generiranja terena kako bi korisnik mogao postići željene rezultate. Sada ćemo spomenuti neke takve metode, kao i neke druge metode koje se mogu koristiti za proceduralno generiranje terena.

Slika 3.3. Slika generirana Terragenom (Megen, 2016.[7])

3.2. Metode generiranja terena

Iako se za postupak proceduralnog generiranja terena oduvijek najčešće koriste računala, jedna od najstarijih metoda je u potpunosti analogna. Radi se o knjizi Dungeon Masters Guide (Gygaxa, 1979.[8]). To je knjiga pravila za stolu igru uloga Advanced Dungeons and Dragons. U privitku te knjige nalazi se niz tablica. Voditelj igre, na temelju rezultata bacanja kocke, određuje redak te tablice u kojem su napisane upute za daljnje generiranje prostorije. Na primjer, rezultat kocke 5, na tablici 1, može označavati uputu "dodaj novu pravokutnu prostoriju, te na tablici 3 odredi broj vrata u njoj". Taj postupak može ponavljati dok nije generiran povezan niz soba raznih oblika te lokacije raznih objekata u tim sobama. Rezultat je potpuna mapa koja se može dalje iskoristit za igru.

Velik broj igrica, pogotovo onih iz prije spomenutog roguelike žanra, pri proceduralnom generiranju svojih mapa, pokušavaju generirati mape slične onima kakve se pojavljuju u Dungeons and Dragons. Oni ne generiraju velike, otvorene reljefe, već unutarnju strukturu špije, dvorca, tvrđave itd. Jedan od najjednostavnijih i osnovnih algoritama za postići ovakvo proceduralno generiranje je sljedeći. Započnemo s praznom mapom. Izaberemo jedan nasumični pravokutnik te ako se ne preklapa ili graniči s postojećim

pravokutnikom, postavimo ga u mapu kao prostoriju. To ponavljamo sve dok nemamo željeni broj prostorija. Nakon toga povežemo sve prostorije nasumičnim hodnicima. Rezultat će biti nešto slično slici 3.1.

Drugi algoritam kojim se mogu postići manje pravilni rezultati je algoritam pijanog hoda. Odaberemo neku točku na našoj mapi te od nje započnemo hod. Svaki korak radimo nasumično te bilježimo kuda smo prošli i to područje označavamo kao dio prostorije. Nakon nekog broja korak prekidamo hod i stvaramo nekoliko novih "pijanaca" unutar označenog prostora koji započinju novi hod. Ovaj postupak ponavljamo dok nismo otkrili dovoljno površine, nakon čega prekidamo algoritam. Time smo dobili znatno manje pravilan teren koji izgleda kao da je nastao erozijom. Jedna od prednosti ovog algoritma je ta da osigurava mogućnost dosezanja svakog dijela mape.

Sljedeći algoritam koji ćemo opisati je algoritam temeljen na staničnom automatu (eng. Cellular automaton). Ideja je početi s poljem koje ima nasumično označene ćelije kao prolazne ili neprolazne. Na takvom polju pokrenemo stanični automat s nekim setom pravila, na primjer onima iz Conwayevoj Igri života. Nakon nekog broja koraka kad prekinemo algoritam vidjet ćemo puno prirodniju strukturu od one nasumične s kojim smo počeli. Dobivamo prostor koji ima strukturu sustava špija. Ovdje je također moguće iskoristiti jedan od čestih pomoćnih algoritama kojeg algoritmi proceduralnog generiranja terena koriste, Dijkstrin. Pomoću njega možemo otkriti koji su dijelovi stvorenih špilja povezani, a koji nisu te ih možemo ili odbaciti ili prokopati put do njih.

Dok su dosad spomenuti algoritmi više pogodni za generiranje zatvorenih prostora, sljedećim algoritmom možemo generirati veliki otvoreni teren s nizom brda. Algoritam dijamant-kvadrat je algoritam u kojem započinjemo s četverokutom koji u svakom vrhu ima pridijeljenu neku visinu. Taj četverokut onda dijelimo u 4 nova četverokuta. To radimo u dva koraka. Prvi je korak da za svaki četverokut odredimo središnju točku, odredimo joj visinu, kao prosjek visina u vrhovima, te je postavimo na polje. Drugi je korak da pomoću novonastalih sredina i rubova izračunamo visinu središta svakog brida te nju variramo nekim šumom i dodamo na polje. Ovako smo dobili vrhove 4 manja kvadrata te rekurzivno ponavljamo postupak pri čemu postepeno smanjujemo amplitudu šuma. Rezultat će biti fraktalna visinska mapa koja, kad bi ju prikazali, daje brdovit i planinski teren koji izgleda izrazito realistično. Dodatna prednost algoritma je što omo-

gućava da određena područja imaju veće ili manje razine detalja čime možemo uštedjeti na vremenu pri generiranju terena.

Još jedan algoritam kojim možemo postići slične rezultate je Perlinov šum. Perlinov šum je također metoda koju smo izabrali za demonstracijski primjer te će biti detaljno objašnjen u nastavku rada. Iako Perlinov šum nije sam po sebi jako dobar algoritam za generiranje realističnog terena, zbrajanjem više slojeva njega dobivamo fraktalni šum koji je iznimno koristan i često korišten. Perlinov šum je također temelj mnogih drugih vrsta šuma koje iznimno dobro generiraju teren. Jedan od takvih je hrbatni šum kojim se generiraju iznimno strmi planinsku lanci. Zatim imamo postupak savijanja domene kojim se postižu oblici kao oni nastali riječnim procesima. Postoje i metode kojima se za Perlinov šum mogu dobiti analitičke derivacije, te se njihovim dodavanjem u račun mogu dobiti efekt erozije na terenu. Sam Perlinov šum (tj. više različitih slojeva njega) je temelj proceduralnog generiranja terena igrice Minecraft. Također se može koristiti i kao izvor šuma za algoritme koji pri generiranju terena traže neki šum. Tvorci igrice No Mans Sky su kombiniranjem svih navedenih i još drugih vrsta šumova, stvorili unificiranu metodu generiranja šumom, takozvani Uber noise (Murray, 2017.[9]). Uber nosie ima niz parametara kojima se kontroliraju razni aspektu željenog šuma, na primjer koliko strma brda želimo ili koliko je jak utjecaj erozije. Ideja je da ti parametri tijekom generiranja ne moraju ostati konstantni, već se mijenjaju na temelju nekog meta šuma. Teren koji nastaje kao rezultat dakle može imati puno različitih značajki koje ne bismo uspjeli dobiti korištenjem samo jedne vrste šuma.

Još jedan koristan alat za generiranje terena su Voronoi dijagrami. To je metoda kojom na temelju nekoliko nasumičnih početnih točaka možemo podijeliti prostor u regije, koristeći neku funkciju udaljenosti. Rezultat je da sada imamo nejednoliku podjelu našeg prostora. Sada za svaku regiju nasumično možemo odrediti je li to kopno ili more. Pridijelimo im i neke početne visine. To mogu biti konstantne vrijednosti ili se mogu generirati metodama poput Perlinova šuma ili dijamant-kvadrat. Nakon toga svakoj regiji još i odredimo nasumični vektor kretanja. Time smo dobili tektonske ploče. Svakoj ploči na rubovima odredimo odmiče li se od susjeda ili mu se približava. Na temelju toga prateći set pravila ponašanja tektonskih ploča povisujemo ili spuštamo njihove visine te kao rezultat dobivamo realistične geografske oblike reljefa. Na primejr ako se dvije kop-

nene ploče međusobno približavaju jedan drugoj, obje strane ruba će se podići i nastat će veliki planinski lanac.

Postoje i algoritmi za proceduralno generiranje terena koji su korišteni kao nadogradnje na postojeće algoritme. Primjer toga je kad imamo već gotovu visinsku mapu te na nju želimo nadodati učinke erozije. To bi onda napravili tako da odredimo utjecajne faktore erozije na svakom dijelu naše visinske mape (poput količine kiše, toka vode, itd.) te na temelju njih radimo fizikalnu simulaciju nekog modela erozije. Ovakvim postupkom fizikalne simulacije možemo dobiti i rijeke. No fizikalne simulacije fluida su skupe i kompleksne, pa se za određivanje toka vode uglavnom koriste drugi algoritmi. Jedan od načina je korištenje algoritama pronalaženja puta poput A* ili pohlepni algoritam.

Još jedan od algoritama kojom možemo proceduralno generirati terena je Wave function collapse. Temeljen na istoimenoj pojavi u kvantnoj fizici, ovaj algoritam podijeli prostor u polja te kao početnu vrijednost svih ćelija stavlja superpoziciju svih mogućih rezultata. Svaki rezultat iz te superpozicije ima određenu vjerojatnost da se ostvari. Te vjerojatnosti također su uvjetovane okolnim ćelijama. Na primjer, može postojati uvjet koje kaže da ako je s jedne strane ćelije rijeka, a s druge strane more, onda ta ćelija mora biti ušće . Algoritam kreće tako da za neku ćeliju odabere nasumično vrijednost na temelju distribucije superpozicije te ćelije. Nakon toga se provjerava jesu li se uvjetne vrijednosti neke od ostalih ćelija promijenile tako da one više nisu u superpoziciji, već imaju određenu vrijednost. Ako ima takvih, postavlja ih se na tu vrijednost te ponovno provjerava je li ta promjena uzrokovala još neku. Nakon što se ponovno vrati u stanje gdje su sve preostale ćelije u superpoziciji, ponovno nasumično biramo jednu te joj određujemo vrijednost, i taj postupak ponavljamo dok nismo odredili vrijednost svih ćelija. Ovaj postupak nam daje veću kontrolu nad rezultatom jer pomoću pravila možemo osigurati određene uvjete.

Kao što smo već spomenuli, glavna ideja i prednost ovih algoritama je što se mogu međusobno nadograđivati i kombinirati kako bi ispravljali vlastite slabosti i mane čime se postiže točnije i detaljnije proceduralno generiranje terena.

4. Proceduralno generiranje karte pomoću Perlinova šuma

4.1. Perlinov šum

Perlinov šum je vrsta i prvi primjer gradijentnog šuma. Razvio ga je Ken Perlin 1983. godine, a objavio 1985. u časopisu ACM SIGGRAPH Computer Graphics, pod naslovom "An Image Synthesizer". "This work arose out of some experiments into developing efficient naturalistic looking textures. Several years ago we developed a simple way of creating well behaved stochastic functions. We found that combinations of such functions yielded a remarkably rich set of visual textures." (Perlin, 1985.[10]). Razvijen je kao način za micanje prevelike pravilnosti koja je bila prisutna u računalno generiranim slikama (CGI) kako bi izgledale realističnije. "I was frustrated by the fact that everything looked machine-like"(Perlin, 2007.[11]). Glavna prednost ovakvog šuma je to da, iako je pseudo-nasumičan za svaku točku, vrijednost koju daje ima glatki prijelaz između više susjednih točka. Ovakav šum ima iznimno široku uporabu u računalnoj grafici i proceduralnom generiranju. Najčešće se koristi za generiranje tekstura. Velika mu je prednost što je vrijednost šuma jednostavna za izračunati i ne ovisi o susjednim vrijednostima, što omogućava korištenje šuma kao komponente matematičkih funkcija za dobivanje raznih efekata i također omogućava njihovo paralelno računanje. Također se često koristi kada je potrebno koristiti velike teksture kad imamo memorijsko ograničenje.

Još jedna česta uporaba, i ona na koju ćemo se mi najviše osvrnuti, je proceduralno generiranje realističnog terena. Najčešća primjena ovoga je u filmskoj industriji i u proizvodnji videoigrica.

Kako bismo objasnili postupak generiranja Perlinovog šuma moramo biti svjesni toga što je to za nas šum. Za naše potrebe definiramo šum kao funkciju koja prima n-dimenzionalnu

Slika 4.1. Teksturiranje vaze Perlinovim šumom (Perlin, 1985.[10])

točku s realnim koordinatama (n je najčešće 2, 3 ili 4) te vraća vrijednost najčešće iz intervala $[-1, 1]$ ili $[0, 1]$. Perlinov šum to radi tako da na početku svakoj cjelobrojnoj koordinati odredi neki pseudo-nasumični gradijent. Zatim za neku ulaznu točku odredi sve okolne cjelobrojne točke. Za 2D točku na primjer to su vrhovi okolnog jediničnog kvadrata. Za svaki od tako dobivenih vrhova odredi se vektor od tog vrha do te zadane točke. Za taj vektor i gradijent u toj točci izračunamo skalarni produkt. Taj postupak ponovimo za svaki vrh. Nakon toga radimo smoothstpe interpolaciju između dobivenih vrijednost po svim osima te konačnu vrijednost interpolacije vraćamo kao rezultat funkcije šuma..

Implementacija gore opisanog algoritma izgleda ovako:

```
// perlin.h
typedef struct {
        float x, y;
} vector2d;
```

```
class PerlinNoise {
    protected:
        float smoothsept(float w);

    private:
        unsigned _seed;
        std::vector<int> permutation;
        float interpolate(float a0, float a1, float w);
        float dotGridGradient(
        int ix, int iy, float x, float y);

    public:
        vector2d randomGradient(int ix, int iy);
        PerlinNoise() : PerlinNoise(std::time(NULL)){};
        PerlinNoise(unsigned seed);

        ~PerlinNoise(){};
        float perlin(float x, float y);
};

// perlin.cpp
PerlinNoise::PerlinNoise(unsigned seed) : _seed(seed) {
        std::srand(_seed);
        permutation = std::vector<int>();
        for (int i = 0; i < 256; i++) {
                permutation.push_back(i);
        }
        std::shuffle(permutation.begin(), permutation.end(),
                        std::default_random_engine(seed));
        for (int i = 0; i < 256; i++) {
                permutation.push_back(permutation[i]);
        }
```

```
}

float PerlinNoise::smoothsept(float w) {
        if (w <= 0) return 0;
        if (w >= 1) return 1;
        return (6 * w * w - 15 * w + 10) * w * w * w;
}

float PerlinNoise::interpolate(float a0, float a1, float w) {
        return a0 + (a1 - a0) * smoothsept(w);
}

vector2d PerlinNoise::randomGradient(int ix, int iy) {
        vector2d v;
        ix = (ix + 4 * 256) % 256;
        iy = (iy + 4 * 256) % 256;
        int h = permutation[(permutation[ix] + iy)] & 3;
        switch (h) {
        case 0:
            v.x =  1; v.y =  1; break;
        case 1:
            v.x = -1; v.y =  1; break;
        case 2:
            v.x = -1; v.y = -1; break;
        case 3:
            v.x =  1; v.y = -1; break;
        default:
            v.x =  0; v.y =  0; //nikad
        }
    return v;
}
```

```
float PerlinNoise::dotGridGradient(
    int ix, int iy, float x, float y) {
        vector2d gradient = randomGradient(ix, iy);
        float dx, dy, dz;

        dx = x - (float)ix;
        dy = y - (float)iy;

        return (dx * gradient.x + dy * gradient.y);
}

float PerlinNoise::perlin(float x, float y) {
        int x0 = (int)std::floor(x);
        int x1 = x0 + 1;
        int y0 = (int)std::floor(y);
        int y1 = y0 + 1;

        float sx, sy, sz;
        sx = x - (float)x0;
        sy = y - (float)y0;

        float n0, n1, ix0, ix1, value;
        n0 = dotGridGradient(x0, y0, x, y);
        n1 = dotGridGradient(x1, y0, x, y);
        ix0 = interpolate(n0, n1, sx);

        n0 = dotGridGradient(x0, y1, x, y);
        n1 = dotGridGradient(x1, y1, x, y);
        ix1 = interpolate(n0, n1, sx);

        value = interpolate(ix0, ix1, sy);
        // interval od 0 do 1
```

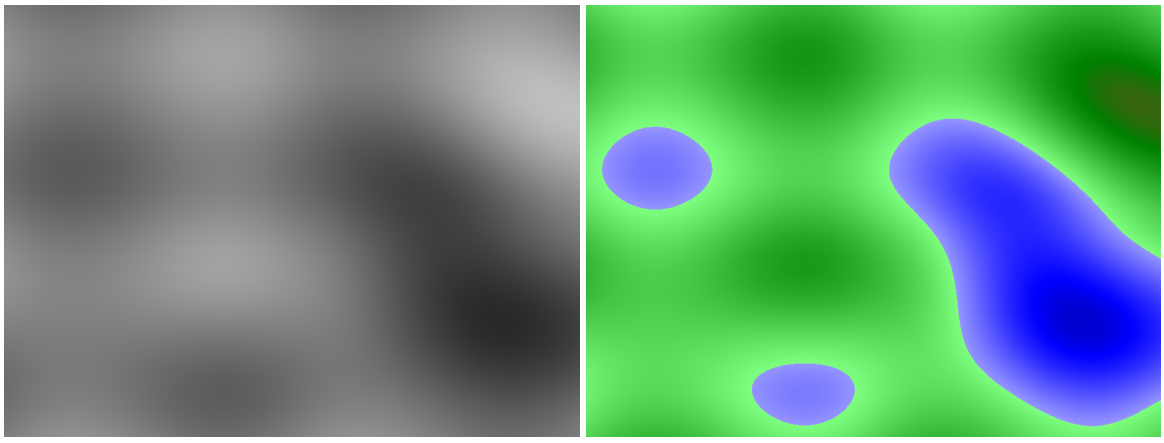
```
        value = value * 0.5 + 0.5;
        return value;
}
```

Svakako treba obratiti pažnju na to da je funkciju šuma potrebno inicijalizirati seed vrijednošću kako bi inicijalizirali permutacijsku tablicu koju koristimo za određivanje gradijenata.

4.2. Korištenje šuma za generiranje visinske mape

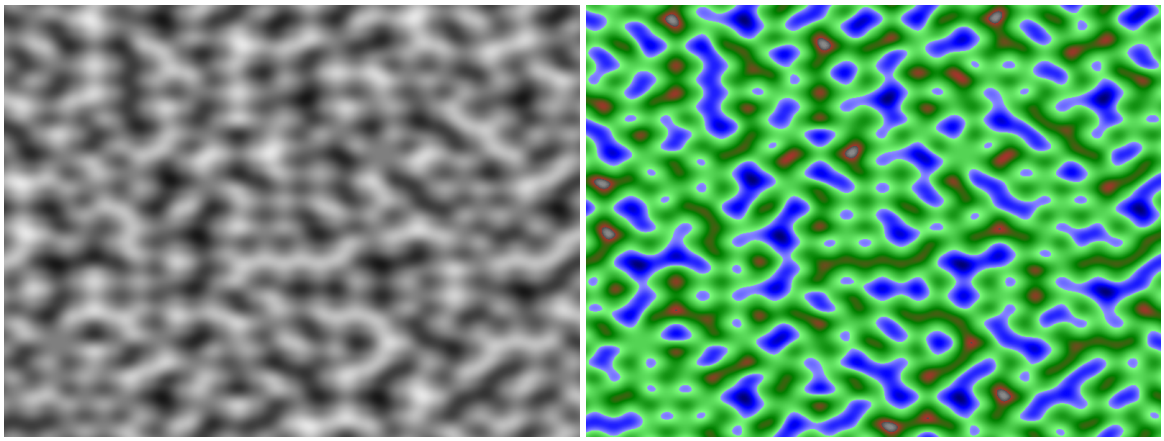
Ideja nam je koristiti Perlinov šum kako bi generirali visinsku mapu koju onda na temelju dobivanih visina možemo obojiti tako da dobijmo kartu proceduralno generiranog terena. Time želimo demonstrirati kako se pomoću jednostavnog šuma može generirati teren na kojem bi se na primjer mogla igrati igra ili prikazati simulacija. Cilj nam je pokazati da usprkos stohastičke prirode šuma možemo dobiti veliku razinu kontrole nad našim rezultatom pomoću parametara koje koristimo u našem postupku. Sama ideja i velik dio idućeg postupka temeljeni su na blog objavi "Making maps with noise functions" od Red Blob Games [14].

Inicijalno postavimo rubove naše karte kao koordinatne vrijednosti iz intervala $[-1, 1]$ te odredimo rezoluciju u kojoj želimo prikazati tu kartu. Za svaki piksel na tom rasteru potrebno je izračunati njegovu xy vrijednost u prostoru karte te ćemo za njega izračunati vrijednost Perlinova šuma. Mi ćemo također mapirati te vrijednosti na interval $[0, 1]$ te za tako dobivene visine mapirati na funkciju koja će pridijeliti boje određenim visinama.



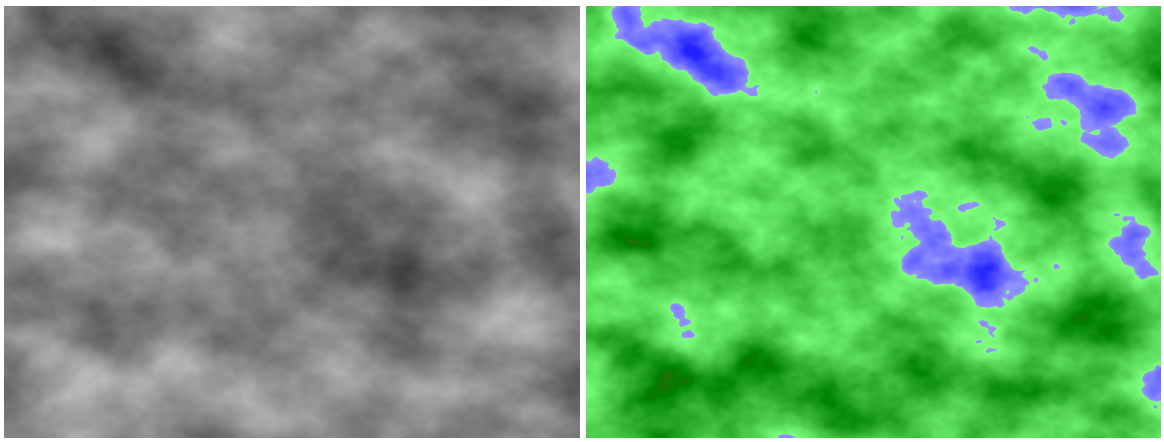
Slika 4.2. Običan šum

Vidimo da je dobiveni rezultat (4.2.) loš. Kako je Pelinov šum temeljen na gradijentima u koordinatama s cjelobrojnim vrijednostima, a naša slika ih sadrži samo pet, nemamo veliku raznolikost u našoj slici. Kako bismo to popravili možemo pokušati s većim frekvencijom Perlinova šuma tako da ulazne vrijednosti funkcije pomnožimo s, na primjer, 8 prije no što ih pošaljemo kroz funkciju šuma.



Slika 4.3. Običan šum s višom frekvencijom

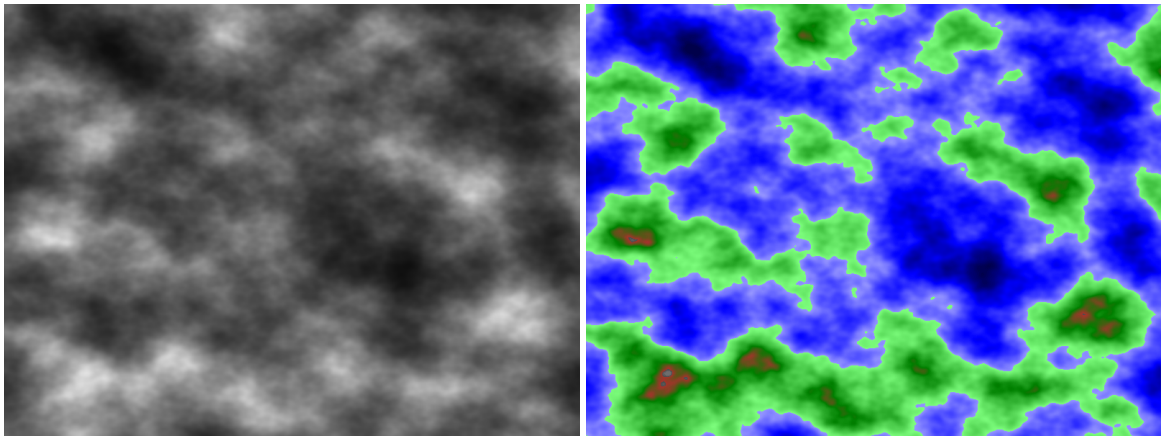
Ni ovako dobiveni rezultat (4.3.) još uvijek nije dobar. Naime, dobili smo oble obale i preglatke površine. Rješenje tog problema je fraktalno Brownovo gibanje. To je postupak kojim pokušavamo dobiti šum nalik Brownovom gibanju tako da u nekoliko oktava zbrajamo različite frekvencije Perlinova šuma jednu s drugo te kao rezultat dobivamo novu vrstu šuma s fraktalnim uzorcima. Takvim fraktalnim uzorcima dobit ćemo realističniju obalu jer je u stvarnosti obala također iznimno fraktalna.



Slika 4.4. Fraktalni šum

$$FBMNoise(point) = \sum_{i=1}^{oct} \frac{Noise(point * 2^i)}{2^i} \tag{4.1}$$

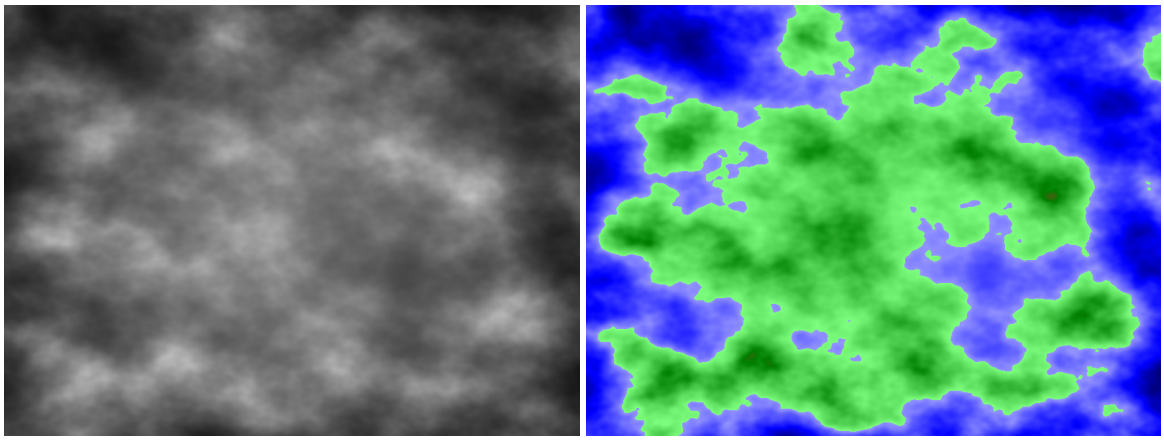
Ovdje već vidimo bolje rezultate (4.4.) i nešto što izgleda kao karta. No primjećujemo manjak visokih područja i velikih dubina. Posljedica zbrajanja više nasumičnih vrijednosti šuma je ta da se varijanca te vrijednosti smanjuje te se vrijednosti počinju težiti prema središtu intervala. Ovaj problem možemo riješiti tako da naš dobiveni rezultat skaliramo nekom vrijednosti. Uz to možemo primijetiti da naš dobiveni rezultat imam manjak strmosti pa na to možemo utjecati tako da potenciramo dobivenu vrijednost (4.5.).



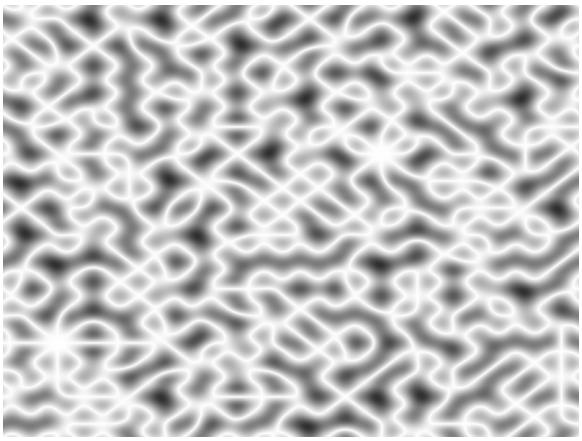
Slika 4.5. Farktalni šum + skaliranje + potenciranje

Za daljnje potrebe ovoga rada odlučili smo da želimo generirati jedan središnji otok. Time ćemo nastojati pokazati kako se može na razne načine utjecati na inicijalno nasumične rezultate kako bismo dobili nasumično generirani teren prema nekim našim specifikacijama. To ćemo napraviti tako da odredimo neku funkciju udaljenosti (u našem primjeru je to Square Bump) od ruba karte te na temelju nje izrađujemo visinsku mapu. Nakon tog linearno interpoliramo između te mape i mape našeg šuma po nekom faktoru. Na taj način ćemo potopiti rubove i uzdići sredinu naše karte. Sada dakle pomoću tog faktora možemo odrediti združenost zemljane mase naše karte.

Ovaj rezultat (4.6.) dodatno možemo poboljšati dodavanjem planinskih lanaca. Naime trenutnim postupkom nemamo način za dobivanje lanaca prirodno strmih planina. No to možemo postići tako da umjesto običnog šuma koristimo hrbatni šum (eng. ridge noise) (4.7.). Ideja je da uzmemo našu vrijednost šuma te joj preslikamo vrijednosti na način da ona ne ide linearno između nula i jedan, već da raste do jedan na prvoj polovici te pada nazad do 0 na drugoj (4.8.).



Slika 4.6. Otok



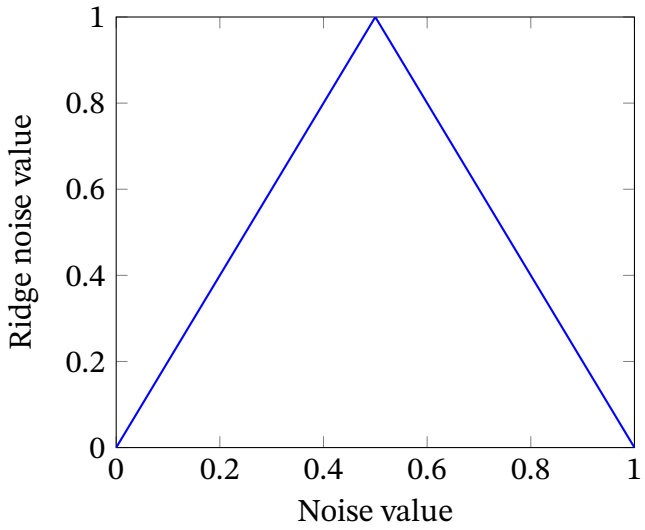
Slika 4.7. Hrbatni šum

$$Ridge(point) = 1 - |2 * Noise(point) - 1| \tag{4.2}$$

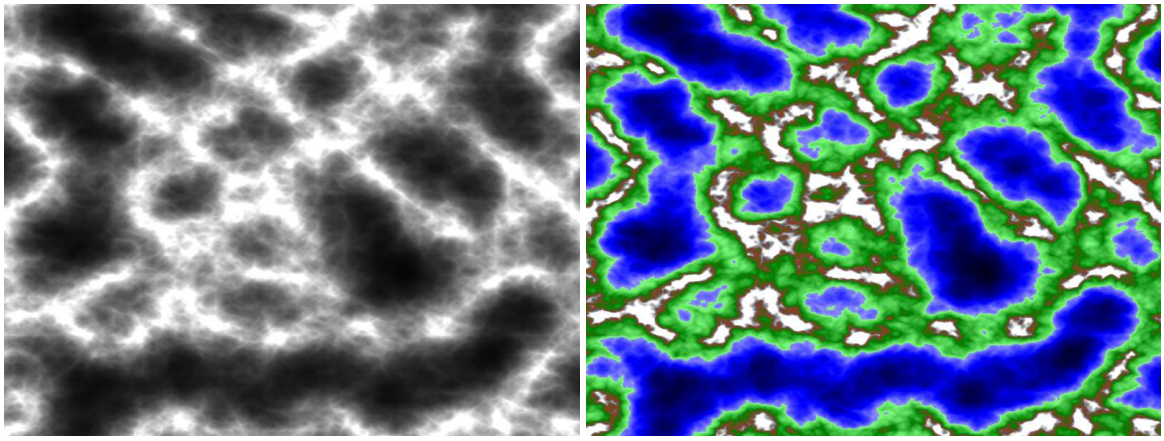
Za takav hrbatni šum ponavljamo postupak fraktalnog zbrajanja skaliranja i potenciranja. Na temelju rezultat ponovno radimo kartu.

Vidimo da smo dobili planinske lance, no previše ih je te se presijecaju i opet ne izgledaju realistično (4.9.). Zato je naša ideja ne koristit samo ovu mapu, već linearno interpolirati između nje i one prijašnje kako bismo dobili novu kartu s planinskim lancima (na koju onda ponovo primjenjujemo udaljenost od rubova).

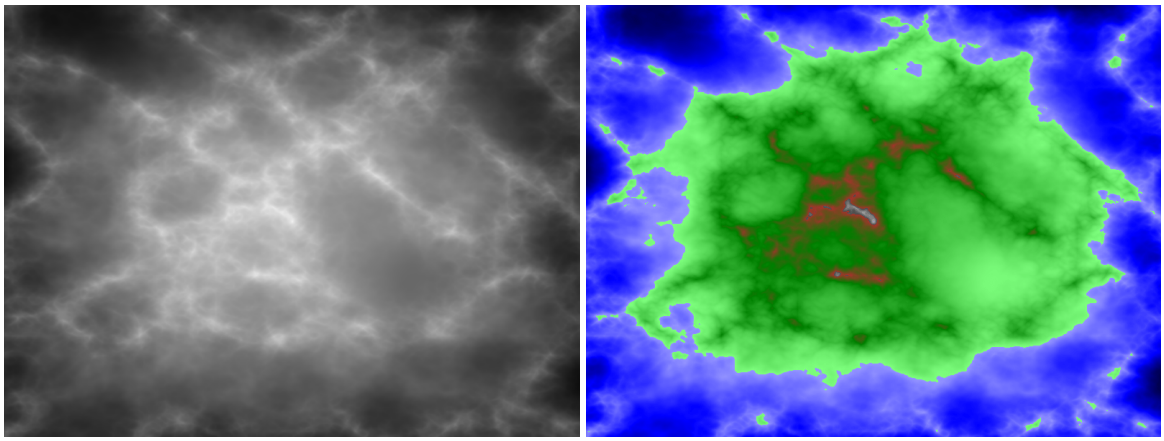
S ovakvim rezultatom (4.10.) smo zadovoljni no ipak ga želimo dodatno izmijeniti. Umjesto konstantnog faktora interpolacije između fraktalne i hrbatno fraktalne mape možemo uvrstiti strategiju koja interpolira između njih na temelju udaljenosti od ruba karte (4.11.).



Slika 4.8. Preslikavanje šuma u hrbatni šum

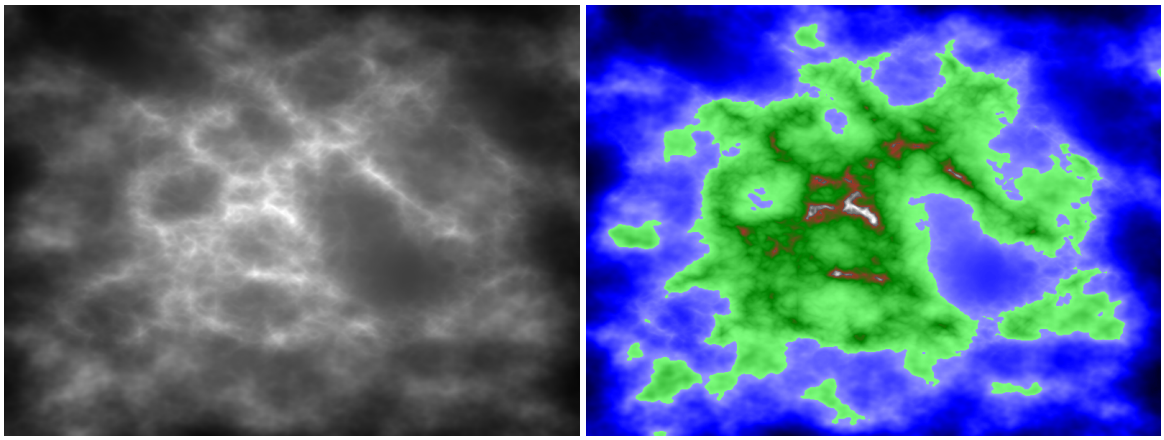


Slika 4.9. Fraktalni hrbatni šum



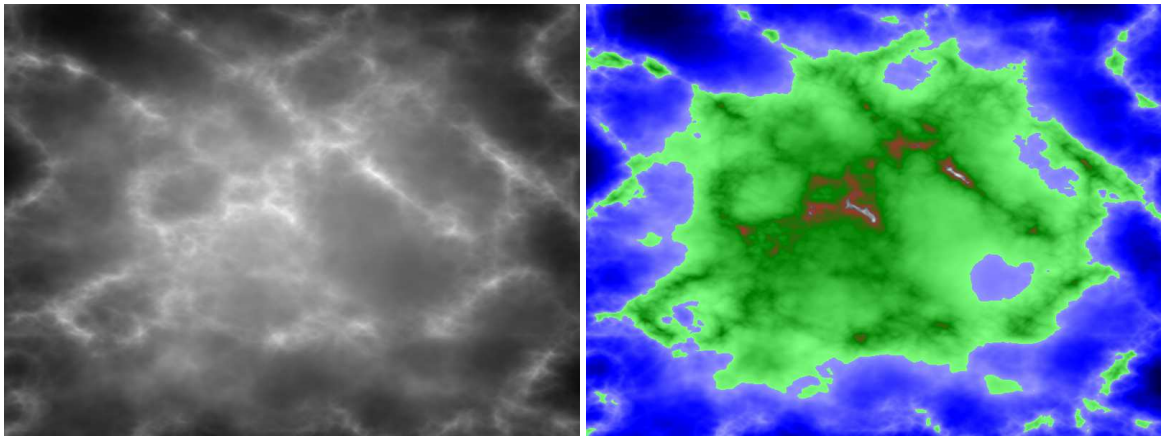
Slika 4.10. Konstantno interpoliranje

Osim ove strategije još jedna koju možemo primijeniti je korištenje sloja meta šuma. Kao vrijednost interpolacije između naših visinskih mapa možemo iskoristiti vrijednost neke funkcije šuma u toj točci (4.12.). Ovo je isto još jedna korisna primjena šuma gdje



Slika 4.11. Interpoliranje na temelju udaljenosti

ga koristimo kao ulazne parametre nekih drugih funkcija za generiranje nečega.



Slika 4.12. Interpoliranje na temelju šuma

Konačno naš postupak generiranja možemo opisati općom formulu za generiranje visinske mape (formula 4.3)

$$\text{Mix}\left(\text{Mix}\left((a_1 \cdot \text{FBMNoise(point)})^{p_1}, (a_2 \cdot \text{FBMRidge(point)})^{p_2}, f(\text{point})\right), d(\text{point}), l\right) \tag{4.3}$$

gdje Mix predstavlja funkciju linearne interpolacije, a point točku za koju računamo vrijednost visinske mape. Vidimo da u ovom pojednostavljenom primjeru svejedno imamo velik broj faktora kojima možemo utjecati na rezultat. a_1, a_2, p_1, p_2, l su brojčani koeficijenti kojima možemo utjecati na razne faktore te uz njih imamo i funkciju $d(point)$ koja može biti neka funkcija udaljenosti te strategija interpoliranja između visinskih mapa $f(point)$ koja može, između ostalog, biti još jedan sloj šuma.

Nadogradnjom ovakvog sustava korištenjem još više vrsta šuma (poput Billow, Wor-

ley, savijanje domene, itd.) i da radi u više dimenzija, možemo dobiti sustav za generiranje raznih vrsta terena s izrazito puno varirajućih parametara. Na taj način dobivamo nešto slično ranije spomenutom Uber noiseu ili Terragenu.

5. Zaključak

Ovim radom dali smo kraći pregled područja proceduralnog generiranja. Pokazali smo kako se primjenom raznih algoritama može proceduralno generirati teren prema našim specifikacijama.

Na kraju smo dali primjer na kojem smo demonstrirali jednostavnu primjenu jednog od tih algoritama kako bi se postiglo proceduralno generiranje terena. Kao rezultat dobili smo sustav kojim možemo generirati prikaz terena u obliku karte koja, ovisno o našim specifikacijama, može biti pretežito kopnena ili vodena, može sadržavati visoke planinske lance ili dolinska i brdovita područja. Mogli smo kontrolirati združenost kopna na našoj karti uzdizanjem sredine i potapanjem rubova.

Neke od mana naše metode su to što sadrži neka artefakte usmjerenja koji su posljedica Perlinovog šuma. Naša metoda također ne sadrži način za stvaranje rijeka te ne može simulirati učinke erozije. Naši planinski lanci također bi mogli biti preciznije pozicionirani, oponašajući nastanak na temelju tektonskih ploča.

Neke od ovih ograničenja i mana mogu se ispraviti drugim vrstama šuma ili korištenjem našeg šuma kao osnove za neki drugi od spomenutih algoritama.

Jedna od mogućih nadogradnji je generiranje podataka o temperaturi i vlazi kako bismo na temelju tih podataka i visine mogli generirati i podijeliti kartu na biome. Mogli bismo i dodati rijeke nekim od prije spomenutih algoritama. Također bi mogli na temelju takvih podataka odrediti i vegetaciju određenih područja. Sve to omogućilo bi nam određivanje područja pogodnih za obitavanje ljudi te bismo na kartu mogli dodati i prikaz naselja.

Ovakvim nadogradnjama dakle mogli bismo doći do iznimno kompleksnog sustava

za generiranje terena. Cilj ovog rada bio je prikazati temelj i pozadinu ovakvog konceptualnog sustava što smo ovime postigli.

Literatura

[1] B. Dean, “Norwich gaming festival 2015”, http://ready-up.net/2015/04/24/norwich-gaming-festival-2015/, 2015., pristupljeno: 2024-06-1.

[2] “Massive 8.0”, https://www.massivesoftware.com/massive8.0-press-release.html, 2015., slika, Pristupljeno: 2024-05-31.

[3] M. Rice, J. Jenny, i W. Harrower, “Engine-eering a procedural pipeline with usd”, u *SIGGRAPH ’23 Talks*, Los Angeles, CA, USA, 2023., str. 1–2.

[4] Bethesda, “Daggerfall – behind the scenes”, https://web.archive.org/web/20040407020037/http://www.elderscrolls.com/tenth_anniv/tenth_anniv-daggerfall.htm, 2004., arhivirana kopija orginala http://www.elderscrolls.com/tenth_anniv/tenth_anniv-daggerfall.htm napravljena 7.4.2004., Pristupljeno: 2024-05-31.

[5] “Roguelike”, https://en.wikipedia.org/wiki/Roguelike, 2021., slika, Pristupljeno: 2024-06-2.

[6] L. K. Murphy, J. Jenny, M. O’Brien, i C. Thompson, “Creating a planet and clouds lightyears away”, u *SIGGRAPH ’22 Talks*, Vancouver, BC, Canada, 2022., str. 1–2.

[7] R. V. Megen, “Utah mesas”, https://renevanmegen.artstation.com/projects/1RZz2, 2016., slika, Pristupljeno: 2024-06-4.

[8] G. Gygax, *Dungeon Master’s Guide*. TSR, 1979.

[9] S. Murray, “Building worlds using procedural generation”, https://www.gdcvault.com/play/1024514/Building-Worlds-Using, 2017., video prezentacija, Pristupljeno: 2024-05-31.

[10] K. Perlin, “An image synthesizer”, *ACM SIGGRAPH Computer Graphics*, sv. 19, br. 3, str. 287–296, 1985.

[11] ——, “Making noise”, http://https://web.archive.org/web/20071008162042/http://www.noisemachine.com/talk1/, 2007., arhivirana kopija orginala http://www.noisemachine.com/talk1/ napravljena 8.10.2007., Pristupljeno: 2024-05-31.

[12] A. Biagioli, “Understanding perlin noise”, https://adrianb.io/2014/08/09/perlinnoise.html, 2014., pristupljeno: 2024-05-31.

[13] R. Touti, “Perlin noise algorithm”, https://rtouti.github.io/graphics/perlin-noise-algorithm, 2023., pristupljeno: 2024-05-31.

[14] RedBlobGames, “Terrain from noise”, https://www.redblobgames.com/maps/terrain-from-noise/, 2015., pristupljeno: 2024-05-31.

Sažetak

Proceduralno generiranje terena

Lovro Mužar

Ovaj rad razmatra proceduralno generiranje terena. Daje osnovni pregled proceduralnog generiranje terena te opis metoda kojima se može ono postići. Konačno, dajemo opis temelja sustava za proceduralnog generiranje terena, prikazan kroz primjere proceduralno generiranih karata terena.

Ključne riječi: Perlinov šum, Proceduralno generiranje, Proceduralno generiranje terena, Generiranje terena, Generiranje karte

Abstract

Procedural terrain generation

Lovro Mužar

This paper considers procedural terrain generation. It provides a basic overview of procedural terrain generation and a description of the methods by which it can be achieved. Finally, we give an idea for a basis of a system for procedural terrain generation, shown by examples of procedurally generated maps of terrain.

Keywords: Perlin Noise, Procedural generation, Procedural terrain generation, Terrain generation, Map generation