

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2247

PROCEDURALNO GENERIRANJE TERENA U GODOTU

Jakov Ramljak

Zagreb, lipanj 2026.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2247

PROCEDURALNO GENERIRANJE TERENA U GODOTU

Jakov Ramljak

Zagreb, lipanj 2026.

ZAVRŠNI ZADATAK br. 2247

Pristupnik: **Jakov Ramljak (0036559756)**
Studij: Elektrotehnika i informacijska tehnologija i Računarstvo
Modul: Računarstvo
Mentorica: prof. dr. sc. Željka Mihajlović

Zadatak: **Proceduralno generiranje terena u Godotu**

Opis zadatka:

U suvremenoj računalnoj grafici vrlo često se javlja potreba za izradom okoliša u kojima teren predstavlja osnovu. Kako bi ova izrada bila efikasna zlatni standard u današnje vrijeme je proceduralno generiranje jer omogućuje parametarsko podešavanje željenih rezultata i generiranje beskonačnih i raznovrsnih okoliša. Cilj rada je proučiti programski alat Godot, a posebice mogućnosti proceduralnog generiranja terena. Razraditi programsku implementaciju proučenih postupaka. Omogućiti interaktivno upravljanje ključnim parametrima za generiranje proceduralnog sadržaja. Analizirati utjecaj raznih parametara na kvalitetu ostvarenih rezultata, te dati kritički osvrt. Usporediti postignute rezultate sa sličnim rješenjima. Diskutirati upotrebljivost rješenja i dati smjernice za buduća poboljšanja. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 12. lipnja 2026.

Sažetak

Ovaj rad se bavi proceduralnim generiranjem Perlinovog šuma te njegovom primjenom u izradi 3D terena. Za izradu visinske mape korišten je Perlinov šum. U radu je objašnjen algoritam izrade Perlinovog šuma te je uveden pojam oktava kako bi se dobio prirodniji rezultat. Uz pomoć generirane visinske mape i sjenčara, rad se bavi oblikovanjem terena, te računanjem normala za isti. Prikazan je rad sjenčara pri bojanju terena u dva primjera: prvi samo na temelju visine terena, a drugi uzimajući u obzir i njegovu vlažnost. Također, u radu je uveden pojam mora te je pomoću primjera valova na moru objašnjen pojam mape normala. Napravljeno je korisničko sučelje radi lakše promjene parametara potrebnih za izradu Perlinovog šuma, a i pojašnjene su kontrole kamere. Na kraju rada je objašnjen proces optimizacije pomoću pozadinskih dretvi. Programsko rješenje napravljeno je u programskom alatu Godot, korištenjem programskih jezika GDScript i Godot Shading Language.

Ključne riječi: proceduralno generiranje, Perlinov šum, teren, visinska mapa, sjenčari, Godot, GDScript, Godot Shading Language

Summary

This thesis explains procedural noise generation and its application in constructing 3D terrain. Perlin noise was used to create a height map. The paper explains the algorithm for creating Perlin noise and introduces the concept of octaves to produce a more natural result. Using the generated height map and shaders, the paper explains terrain shaping and how to calculate normals. The shader's terrain coloring is demonstrated in two examples: one based solely on elevation and another that takes terrain moisture into account. The paper introduces the idea of the sea and explains normal maps using the example of sea waves. A user interface was created to make it easier to change the parameters needed to create noise, and camera controls were explained. At the end, simulation is optimized using background threads. Software solution was built in the Godot game engine using GDScript and Godot Shading Language.

Keywords: procedural generation, Perlin noise, terrain, heightmap, shaders, Godot, GDScript, Godot Shading Language

Sadržaj

Sažetak.....	1
Summary.....	2
Sadržaj	3
Uvod	4
1. Proceduralno generiranje terena	5
1.1. Proceduralno generiranje sadržaja.....	5
1.2. Gradijentni šum	5
1.3. Generiranje Perlinovog šuma	6
2. Implementacija generiranja terena u Godotu.....	11
2.1. Grafički sustav Godot.....	11
2.2. Izrada terena	11
2.3. Generiranje visinske mape.....	12
2.4. Sjenčari	13
2.5. Izrada 3D terena pomoću sjenčara.....	14
3. Vizualizacija terena	17
3.1. Bojanje terena prema nadmorskoj visini	17
3.2. Implementacija bioma	18
3.3. Implementacija razine mora	20
4. Korisničko sučelje i kamera	22
5. Optimizacija generiranja ćelija.....	24
Zaključak	26
Literatura	27
Izjava o korištenju umjetne inteligencije.....	29

Uvod

Područje razvoja računalnih igara predstavlja jednu od grana računalne grafike koja se posljednjih 20 godina ubrzano razvija. Razvojem tehnologije računalnih komponenti i grafičkih sustava rastu i mogućnosti što je sve moguće proizvesti. Ogromni i detaljni svjetovi s viskom stopom realističnosti su samo jedan od primjera što se očekuje od današnjih modernih igara. Međutim, rastom veličine i složenosti takvih svjetova povećava se i količina vremena potrebna za njihovu izradu. Iz tog razloga se danas koriste mnoge metode koje mogu ubrzati taj proces. Jedna od takvih metoda je proceduralno generiranje sadržaja. Proceduralno generiranje sadržaja je način za stvaranje sadržaja pomoću preddefiniranih pravila, kako bi se zaobišla ručna izrada svakog elementa. Na taj način se mogu generirati razni elementi igre poput 3D modela, terena, tekstura, animacija itd.

U ovom radu obrađeno je proceduralno generiranje 3D terena u programskom okruženju Godot. Cilj rada je bio proučiti grafički sustav Godot te način na koji se proceduralna generacija terena može izvesti u njemu. U poglavljima rada su predstavljeni način generiranja proceduralnog šuma, kako iz takvog šuma dobiti oblik koji nalikuje na teren te na kraju kako vizualno unaprijediti takav objekt u svrhu poboljšanja realističnosti. Na kraju rada je predstavljen rezultat te načini kako se može sustav unaprijediti ili eventualno koristiti u daljnjem razvoju video igre.

1. Proceduralno generiranje terena

1.1. Proceduralno generiranje sadržaja

U računarstvu, proceduralno generiranje sadržaja (engl. procedural generation) je metoda izrade podataka uz pomoću algoritama i unaprijed definiranih pravila [1]. Takav pristup se posebno koristi u području 3D grafike i računalnih animacija. Za razliku od ručnog modeliranja, gdje se svaki element izrađuje zasebno, proceduralni pristup ubrzava taj proces. Osnovna ideja je korištenje matematičkih funkcija kako bi proizveli rezultat koji izgledaju prirodno i nasumično. Također pomoću dodavanja početne vrijednosti pri generaciji omogućujemo ponovno generiranje istog rezultata. Proceduralna generacija je posebno korisna pri izradi video igara gdje su potrebne ogromne količine raznovrsnih tekstura i modela poput terena.

U području generiranja terena, proceduralno se generira sam oblik i izgled terena (planine, doline, ravnice...) te populacija istog objekta (vegetacija, rijeke, jezera, kuće...). Na taj način se znatno ubrzava proces izrade jer se izbjegava ručno modeliranje detalja te ponavljanje istog uzorka objekata. Također, se dobiva na realističnosti jer priroda je, sama po sebi, nasumična.

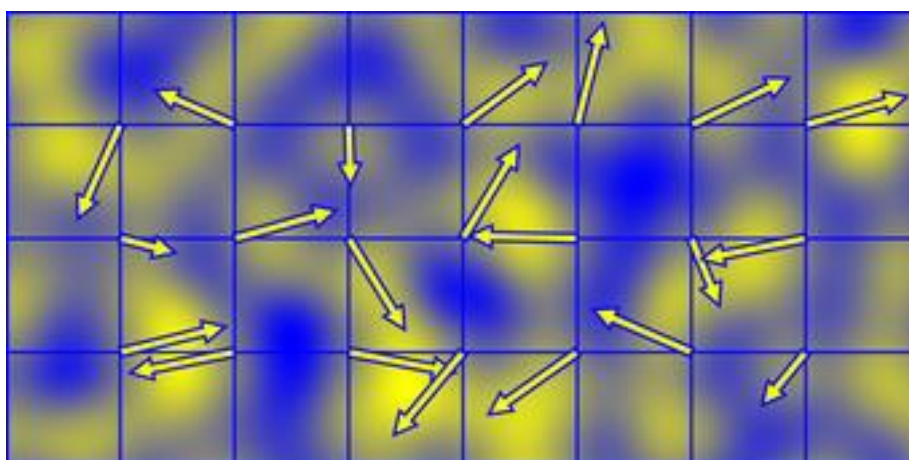
1.2. Gradijentni šum

Šum (engl. noise) je bilo koja vrsta slučajnog signala. Za razliku od običnog, gradijentni šum (engl. gradient noise) dobiven je matematičkim postupkom i služi za stvaranje naizgled nasumičnih, ali kontroliranih uzoraka [2]. U suprotnosti s potpuno nasumičnim vrijednostima normalnog šuma, gradijentni šum proizvodi glatke prijelaze između susjednih točaka [2] zbog čega je pogodan za modeliranje prirodnih pojava poput terena, vegetacije, oblaka... Ova vrsta šuma se dobiva stvaranjem nasumičnih gradijenata, te interpolacijom njihovih skalarnih produkata kako bi se dobili glatki prijelazi između točaka. Postoji više vrsta gradijentnih šumova, među kojima su Perlinov šum, Simplex šum i OpenSimplex šum, no zbog svoje jednostavnosti i brzine, Perlinov šum se danas najčešće koristi u svrhu generiranja terena.

1.3. Generiranje Perlinovog šuma

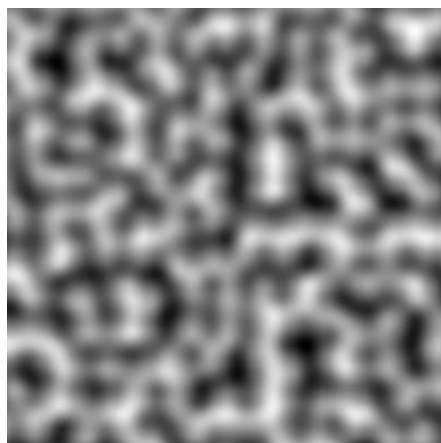
Perlinov šum je vrsta šuma kojeg je izmislio Ken Perlin 1982. godine zbog potrebe za teksturama pri izradi filma Tron [3]. Za potrebe ovog rada korištena je dvodimenzionalna implementacija Perlinovog šuma, no lako se može pokazati kako se prelazi na više dimenzije.

Algoritam je preuzet od A. Biagioli [4]. Kao prvi korak, prostor treba podijeliti na mrežu kvadrata tako da gornji lijevi kut ima koordinate (0, 0) dok donji desni kut ima koordinate (1, 1). Za svaki od tako napravljenih kvadrata, u svakom vrhu izračunati pseudo nasumični vektor. Taj vektor predstavlja gradijent u toj točki, kao što je prikazano na slici (Slika 1.1). Kažemo da je gradijent pseudo nasumičan jer se za isti set ulaznih podataka dobije isti rezultat. Kasnije će nam to omogućiti ponavljanje već prije napravljenog šuma. Kada smo generirali gradijente, moramo proći kroz sve točke u prostoru te za svaku odrediti kojem kvadratu pripada. Kada znamo kojem kvadratu pripada točka, treba izračunati skalarni produkt točke i svakog od gradijenata pripadnog kvadrata. Na kraju se ta 4 skalarna umnoška interpoliraju kako bi se dobila konačna vrijednost te točke.



Slika 1.1 Vektori gradijenata i dobiveni šum [4]

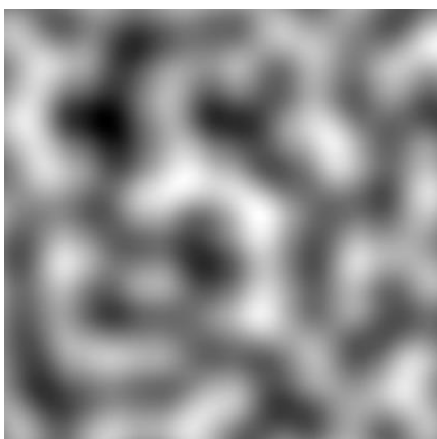
Postupak se ponavlja za svaku točku u prostoru. Kod interpolacije važno je napomenuti da se uz linearnu interpolaciju koristi i *krivulja ublažavanja* (engl. *ease curve*) kako bi se dobili realističniji prijelazi između točaka. Na slici (Slika 1.2) vidimo primjer Perlinovog šuma dobivenog ovim postupkom.



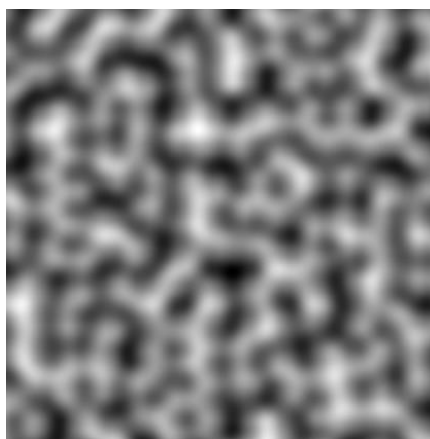
Slika 1.2 Perlinov šum

Postoje razni parametri koji se gledaju pri izradi Perlinovog šuma. Jedan od parametara kojeg možemo mijenjati je frekvencija generiranog šuma. Frekvencija označava razmak između točaka prostora od kojeg računamo šum. Drugim riječima, veća frekvencija uzrokuje "uvećanom" slikom i obrnuto. Kao što vidimo na slici (Slika 1.3) je šum s manjom frekvencijom nego šum na slici (Slika 1.2).

Inicijalna vrijednost generatora (engl. seed) je parametar koji označava jednu od ulaznih vrijednosti kod računanja gradijenata. Ova vrijednost nam dopušta ponavljanje izlaza ako imamo istu vrijednost te promjena ove vrijednosti rezultira potpuno drugačijim šumom. Na slici (Slika 1.4) vidimo šum s drugačijom početnom vrijednosti i primjećujemo da je potpuno drugačiji u odnosu na šum sa slike (Slika 1.2).



Slika 1.3 Perlinov šum sa manjom frekvencijom



Slika 1.4 Perlinov šum sa drugom početnom vrijednosti generatora

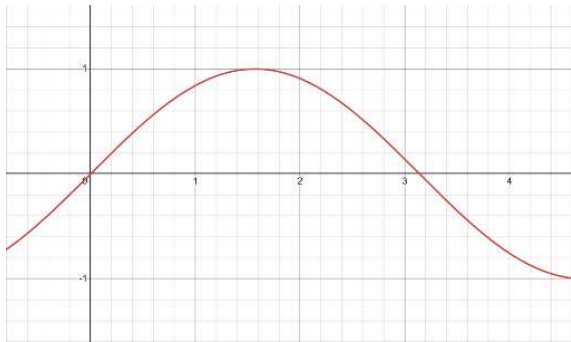
Ovakvim postupkom se generira zadovoljavajući izgled, no korištenjem oktava može se dodatno povećati realističnost generiranog šuma. Oktave su dodatni šumovi koje "slažemo" jedno preko drugog kako bi dobili veći broj detalja u šumu. Dodatne slojeve dobivamo tako što ćemo svakom uzastopnom sloju povećavati frekvenciju, a smanjivati amplitudu. Tako dobivene šumove ćemo zbrojiti. Radi lakše vizualizacije, efekt oktava možemo jednostavno prikazati korištenjem sinus funkcije (Slika 1.5, Slika 1.6, Slika 1.7, Slika 1.8).

$$f_0(x) = \sin(x) \quad (1)$$

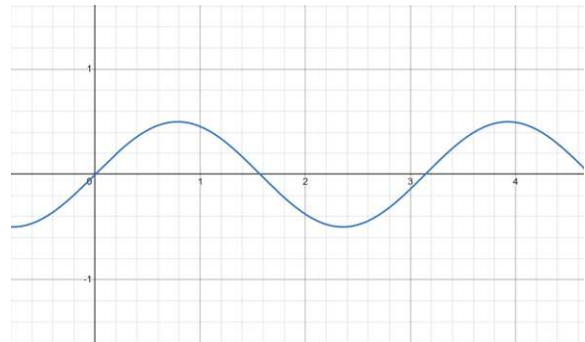
$$f_1(x) = 0.5 * \sin(2x) \quad (2)$$

$$f_2(x) = 0.25 * \sin(4x) \quad (3)$$

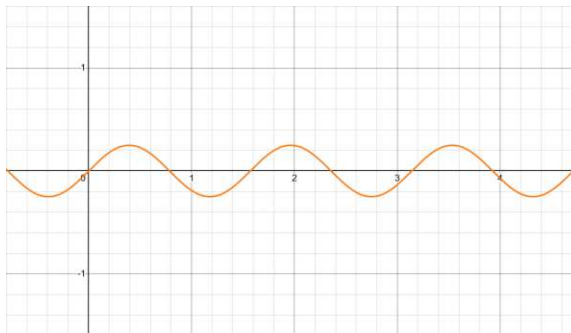
$$f_3(x) = 0.125 * \sin(8x) \quad (4)$$



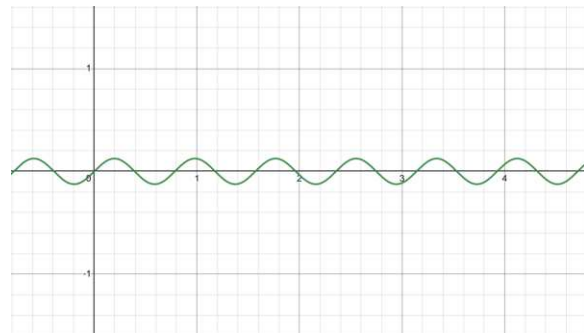
Slika 1.5 Graf funkcije f_0



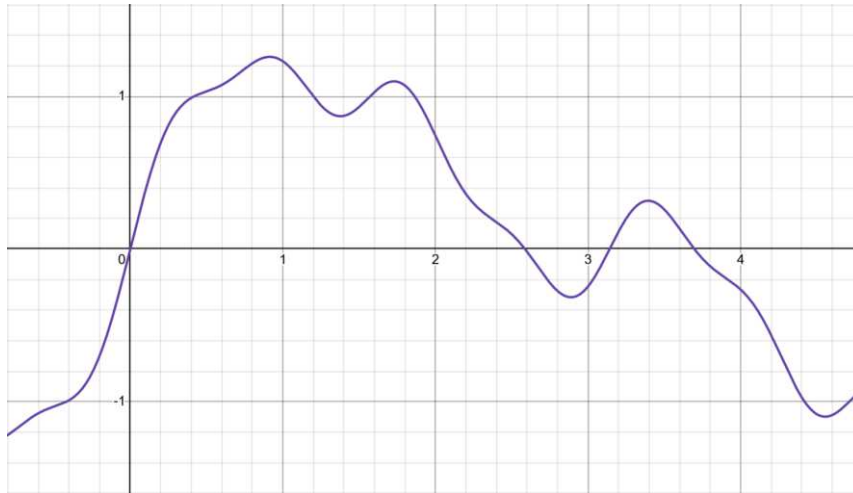
Slika 1.6 Graf funkcije f_1



Slika 1.7 Graf funkcije f_2



Slika 1.8 Graf funkcije f_3



Slika 1.9 Graf funkcije $f_0 + f_1 + f_2 + f_3$

Na slikama (Slika 1.5, Slika 1.6, Slika 1.7, Slika 1.8) vidimo prikaz funkcija od (1) do (4) te na slici (Slika 1.9) vidimo rezultat zbrajanja tih funkcija. Možemo primijetiti da završna funkcija podsjeća na neku vrstu terena, odnosno izgleda puno realističnije nego "obični" sinus. Isti princip možemo primijeniti na Perlinov šum.

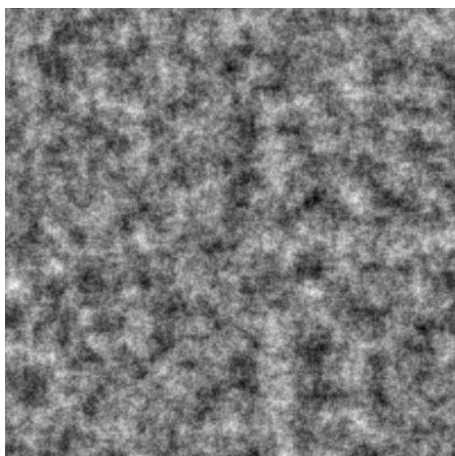
Funkciju oktave možemo zapisati kao:

$$f_n(x) = g^n * noise(l^n * x, l^n * y) \quad (5)$$

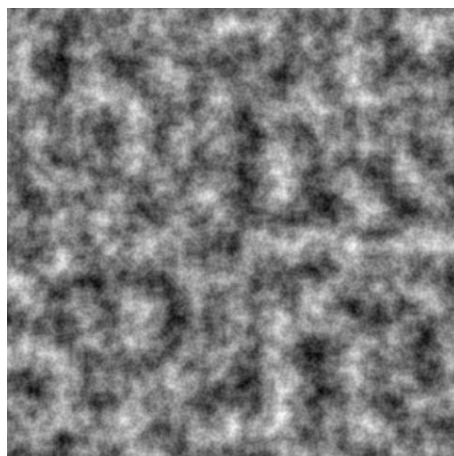
Kao što vidimo iz formule (5) postoje 2 parametra koji upravljaju kako se oktave ponašaju. To su dosljednost (engl. gain) i lakunarnost (engl. lacunarity).

Dosljednost predstavlja omjer amplitude trenutačne oktave u odnosu na prošlu oktavu. Odnosno koliko će svaka sljedeća oktava imati utjecaja u odnosu na prošle. Što je omjer veći to će više oktave doprinositi završnom šumu i obrnuto (uobičajeno se koristi dosljednost od 0.5). Na slici (Slika 1.10) vidimo šum s većom dosljednosti u odnosu na šum sa slike (Slika 1.2).

Lakunarnost predstavlja faktor množenja frekvencije trenutačne oktave u odnosu na prošlu oktavu. Drugim riječima koliko sitnije želimo detalje u svakoj sljedećoj oktavi. Što je lakunarnost veća to su detalji sitniji i obrnuto (uobičajeno se koristi lakunarnost od 2.0). Na slici (Slika 1.11) vidimo šum s većom lakunarnosti u odnosu na šum sa slike (Slika 1.2).



Slika 1.10 Perlinov šum s večom dosljednosti



Slika 1.11 Perlinov šum s večom lakunarnosti

2. Implementacija generiranja terena u Godotu

2.1. Grafički sustav Godot

Godot je grafički sustav (engl. game engine) otvorenog koda napravljen za razvoj 2D i 3D računalnih igara. Iako je sustav napravljen ranih 2000-tih svoju popularnost dobiva u zadnjih 5 godina i trenutačno je jedan od najkorištenijih grafičkih sustava.

Radi lakšeg praćenja rada, potrebno je objasniti osnovne pojmove pri izradi projekata u Godotu. Glavne strukture koje se koriste u radu su čvorovi (engl. node) i scene (engl. scene). Prikazivanje svih elemenata se odvija pomoću hijerarhije čvorova. Svaki čvor može biti čvor roditelj i čvor dijete nekom drugom čvoru te svaki čvor ima neku ulogu, ovisno o vrsti kojoj pripada (npr. Objekt u 2D ili 3D prostoru, element korisničkog sučelja, kamera, svijetlo...). S druge strane, scene predstavljaju samostalne strukture sastavljen od jednog ili više čvorova. Također scene mogu sadržavati druge scene, odnosno svaka scena se može instancirati u drugoj sceni. Ponašanje čvora se dodatno može promijeniti dodavanjem skripti. Skripta se može pridružiti svakom čvoru te u njoj definiramo funkcije i dodatnu logiku koju čvor može obavljati. Godot podržava više programskih jezika poput C++ i C#, no za potrebe ovog rada će se koristiti GDScript, namjenski jezik posebno izmišljen za rad u Godotu.

2.2. Izrada terena

Kod proceduralnog generiranja terena mogu se koristiti dva osnovna pristupa. Prvi pristup podrazumijeva generiranje cijelog terena unaprijed, pri čemu se cijeli teren nalazi u memoriji i pri pokretanju je odmah spreman za prikazivanje. Ovaj način je koristan kod manjih terena koji se ne mijenjaju tijekom izvođenja. Drugi pristup podrazumijeva podjelu prostora na manje cjeline koje se zasebno generiraju. Ovakav pristup je koristan kod dinamičkih i beskonačnih terena zato što se ne mora generirati u cijelosti. Za potrebe ovog rada je izabran drugi pristup zbog beskonačne prirode proceduralnog šuma.

Kako bi izveli taj pristup potrebno je podijeliti prostor na manje jedinice proizvoljne veličine. Te jedinice ćemo nazvati ćelije (engl. chunk). U Godotu je moguće napraviti scenu koja predstavlja jednu ćeliju, koju kasnije možemo instancirati prilikom izvođenja programa. Glavni čvor te scene bit će čvor `MeshInstance3D`, kojemu je zadaća prikazati zadanu mrežu poligona (engl. mesh) u trenutačnoj sceni. Kako bi to ostvarili u atribut `mesh`

pridodaje se preddefinirani resurs `PlainMesh`. Ta ravnina predstavlja osnovu terena koja će se kasnije oblikovati pomoću vrijednosti šuma. Ideja za ovaj pristup je preuzeta od codata [5].

Kako bi se odredilo gdje se pojedina ćelija nalazi unutar globalne scene, svakoj ćeliji je potrebno dodijeliti koordinate u mreži. Ćelija se raspoznaje prema svojim x i z koordinatama u mreži, koje također određuju i njezin položaj u globalnom prostoru. Osim za određivanje položaja, te će se koordinate koristiti i prilikom izračuna vrijednosti šuma. Tako svaka ćelija može dobiti jedinstveni dio generiranog šuma, koji se nadovezuje na okolni teren.

2.3. Generiranje visinske mape

Visinska mapa (engl. height map) je slika koja određuje visinu svake točke terena. Svakoj točki je pridružena vrijednost od 0.0 do 1.0 koja predstavlja visinu, pri čemu svjetlije (veće vrijednosti) označavaju više regije, dok tamnije (manje vrijednosti) označavaju niže regije. Za generiranje visinske mape koristit ću Perlinov šum kako bi dobio neravan i prirodni oblik terena. Štoviše, Godot sadrži ugrađenu klasu `FastNoiseLite` koja omogućuje generiranje vrijednosti šuma bez potrebe za vlastitom implementacijom algoritma, a također omogućuje i rad s oktavama. Potrebno je samo postaviti odgovarajuće parametre klase kako bi se dobio šum.

U skripti glavne scene instanciramo klasu `FastNoiseLite`, zatim postavimo početne parametre i kao vrstu šuma odabiremo Perlinov šum. Zatim za svaku ćeliju generiramo visinsku mapu njezinog dijela prostora. To ćemo postići tako da uzimamo vrijednosti šuma koje smo pomaknuli za koordinate ćelije u mreži ćelija. Priloženi kod (Kôd 2.1) je napravljen po uzoru na RedBlobGames blog [6] i na S. Laguea [7], no prilagođen potrebama ovog rada.

Osnovna ideja algoritma je da prolazi po svakom pikselu slike, koju smo prethodno stvorili, i računa vrijednost šuma za tu točku u prostoru. Kako bismo dobili točnu koordinatu šuma prvo se moramo pomaknuti za koordinate ćelije u globalnom prostoru (umnožak koordinata u mreži i veličine ćelije) te zatim za lokalne koordinate unutar ćelije (pomićemo se za veličinu piksela slike). Tako dobivenu vrijednost pridjeljujemo na točno određenu lokaciju u slici. Nakon što se proces izrade slike završi, slika se pretvara u teksturu i prosljeđuje instanci ćelije koja je kasnije prosljeđuje sjenčaru.

```

var tex_res = 128
var img = Image.create(tex_res, tex_res)

var step = chunk_size / (tex_res - 1)

for y_i in range(tex_res):
    for x_i in range(tex_res):
        var chunk_x = chunk_size * x + x_i * step
        var chunk_z = chunk_size * z + y_i * step

        var value = perlin_noise.get_noise_2d(chunk_x, chunk_z)

        . . .

        img.set_pixel(x_i + 1, y_i + 1, Color(value, 0, 0,
1.0))

```

Kôd 2.1 Generiranje Perlinovog šuma pomaknutog za koordinate ćelije

2.4. Sjenčari

Sjenčari su manji programi, specijalizirani da se izvode na grafičkoj kartici, koji kontroliraju kako se objekti iscrtavaju [8]. Iako su se u početku koristili prvenstveno za određivanje načina osvjetljenja i prikaza 3D objekata, danas omogućuju mnogo širu primjenu. Pomoću njih moguće je kontrolirati način na koji grafički sustav obrađuje geometriju i pojedine piksele na ekranu [8]. Zato što grafičke procesorske jedinice mogu istovremeno izvršavati velik broj operacija, način rada sjenčara razlikuje se od klasičnih programa. Sjenčar se izvršava paralelno za pojedini vrh ili fragment modela, zbog čega se svaki takav element obrađuje neovisno o ostalima [8]. U Godotu se koristi Godot Shading Language koji je vrlo sličan GLSL-u, ali je više prilagođen Godotovom grafičkom sustavu. Jedna od tipova sjenčara u Godotu je `spatial_shader`, koji se koristi za 3D objekte. Uz to što omogućuju bojanje objekata, također omogućuju i pomicanje vrhova. Pomoću njega ćemo modelirati spomenuti `PlaneMesh` kako bi dobili oblik terena.

2.5. Izrada 3D terena pomoću sjenčara

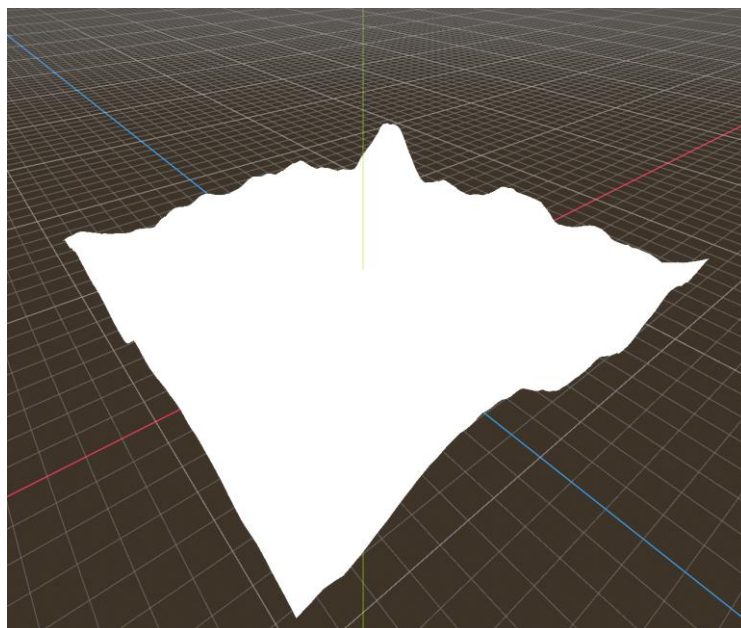
Kako bi mogli koristiti funkcionalnosti `spatial_shadera` potrebno je dodati skriptu sjenčara pod `material` atribut `PlainMesh`a. Za oblikovanje terena bit će nam potrebne dvije glavne funkcije sjenčara, `vertex` i `fragment`. Prvo ćemo pogledati kako smo implementirali oblikovanje vrhova `PlainMesh`a pomoću `vertex` funkcije.

Funkcija `vertex` izvršava se za svaki vrh (engl. `vertex`) modela te u njoj možemo mijenjati položaj tih vrhova. Budući da je početni `PlaneMesh` ravan, svaki njegov vrh potrebno je pomaknuti po `y` osi ovisno o vrijednosti visinske mape. Za svaki vrh dohvaćamo odgovarajuću vrijednost iz visinske mape te je ju pridodajemo trenutnoj visini vrha. Kako bi znali koju vrijednost trebamo dohvatiti, koristit ćemo `UV` koordinate. `UV` koordinate su dvodimenzionalne koordinate koje određuju položaj određene točke na površini modela. Njihove vrijednosti se kreću od 0.0 do 1.0, gdje vrijednost 0.0 predstavlja početak, a vrijednost 1.0 kraj površine. U našem slučaju, `UV` koordinate omogućuju nam da za svaki vrh odredimo gdje se on nalazi u mreži vrhova i na temelju toga dohvatimo odgovarajuću vrijednost visine. Kao što vidimo iz koda (Kôd 2.2), prvo smo teksturu generiranog šuma podijeliti na najmanje jedinice, `texele` (engl. `texture element`). Kako bi kasnije mogli izračunati pravilne normale terena, bilo je potrebno da pri računanju visinske mape generiramo mapu s dodatnim marginama s vrijednostima susjednih terena. Iz tog razloga je potrebno napraviti `UV` koordinate samo za središnji dio teksture šuma koje smo nazvali `new_uv`. Pomoću tih `UV` koordinata možemo dohvatiti ispravne podatke o visini trenutnog vrha.

```
void vertex() {
    vec2 texel = 1.0 / vec2(textureSize(noise, 0));
    vec2 new_uv = clamp(UV, texel * 1.5, vec2(1.0) - (texel *
    1.5));

    float height = texture(noise, new_uv).r;
    VERTEX.y += height * height_scale;
}
```

Kôd 2.2 `Vertex` funkcija u sjenčaru



Slika 2.1 PlainMesh s pomaknutim vrhovima

Kao što se vidi iz slike (Slika 2.1) vrhovi su dobro pomaknuti, ali Godot sustav ne zna kako računati sjene za ovaj objekt jer i dalje "misli" da je to samo ravna površina. Kako bi to popravili potrebno je zadati nove normale, što možemo izvesti pomoću funkcije `fragment`.

Funkcija *fragment* izvršava se za svaki fragment, odnosno svaki piksel (engl. pixel) koji pripada površini modela. Prvenstveno se koristi za određivanje konačnog izgleda površine poput boje, texture i osvjetljenja. Budući da smo promijenili položaj vrhova PlainMesha potrebno je za svaki piksel zadati novu normalu kako bi Godot sustav znao kako pravilno osvjetlit površinu. Za izračun su potrebne vrijednosti visina susjednih točaka trenutnog vrha kako bi izračunali nagib. Kao što vidimo iz koda (Kôd 2.3), pomoću nagiba računamo vektore tangenti u x i z smjeru, te zatim računamo vektorski umnožak tangenti kako bi dobili vektor normale okomit na te vektore. Na kraju kada imamo vektor normale, transformiramo ga iz lokalnog prostora modela u prostor scene pomoću matrica `MODEL_MATRIX` (matrica transformacije u globalni prostor) i `VIEW_MATRIX` (matrica transformacije u prostor kamere). Izračunati vektor normale pridružimo varijabli `NORMAL`. Na slici (Slika 2.2) vidimo poboljšanje sjenčanja u odnosu na teren sa slike (Slika 2.1)

```

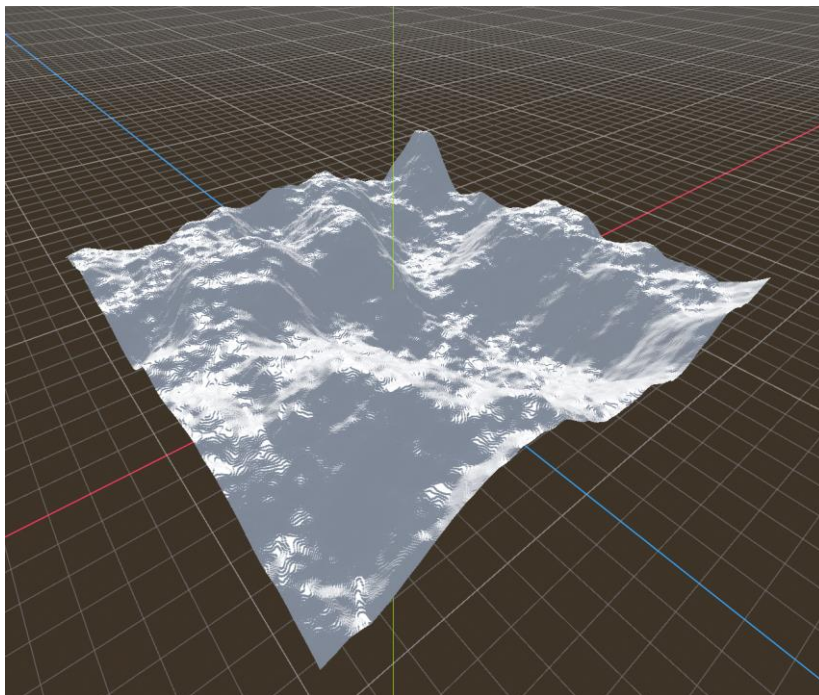
void fragment() {
    float h_l = texture(noise, shrunk_uv - vec2(texel.x, 0.0)).r;
    float h_r = texture(noise, shrunk_uv + vec2(texel.x, 0.0)).r;
    float h_u = texture(noise, shrunk_uv - vec2(0.0, texel.y)).r;
    float h_d = texture(noise, shrunk_uv + vec2(0.0, texel.y)).r;

    vec3 tangent_x = vec3(
        2.0 * terrain_size * texel.x,
        (h_r - h_l) * height_scale,
        0.0
    );
    vec3 tangent_z = vec3(
        0.0,
        (h_d - h_u) * height_scale,
        2.0 * terrain_size * texel.y
    );

    vec3 n = normalize(cross(tangent_z, tangent_x));
    NORMAL = (VIEW_MATRIX * (MODEL_MATRIX * vec4(n, 0.0))).xyz;
}

```

Kôd 2.3 Fragment funkcija za računanje normala



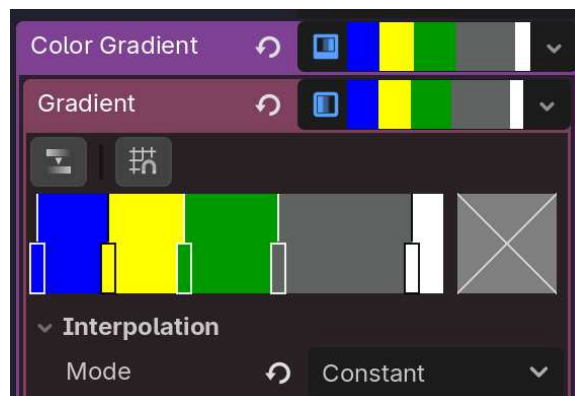
Slika 2.2 PlainMesh s dodanim normalama

3. Vizualizacija terena

3.1. Bojanje terena prema nadmorskoj visini

Kako teren ne bi bio jednolične bijele boje, potrebno ga je obojiti. Najučinkovitijim se pokazalo bojanje terena prema njegovoj visini, pri čemu se različitim rasponima visina pridjeljuje drugačija boja. Najnižim dijelovima terena pridijeljena je plava boja koja predstavlja morsko dno, nakon čega slijedi žuta boja koja predstavlja pješčanu plažu. S povećanjem visine boja postupno prelazi u zelenu, koja predstavlja područja s vegetacijom, zatim sivu koja predstavlja kamenite dijelove terena, te na kraju bijelu koja predstavlja područja prekrivena snijegom.

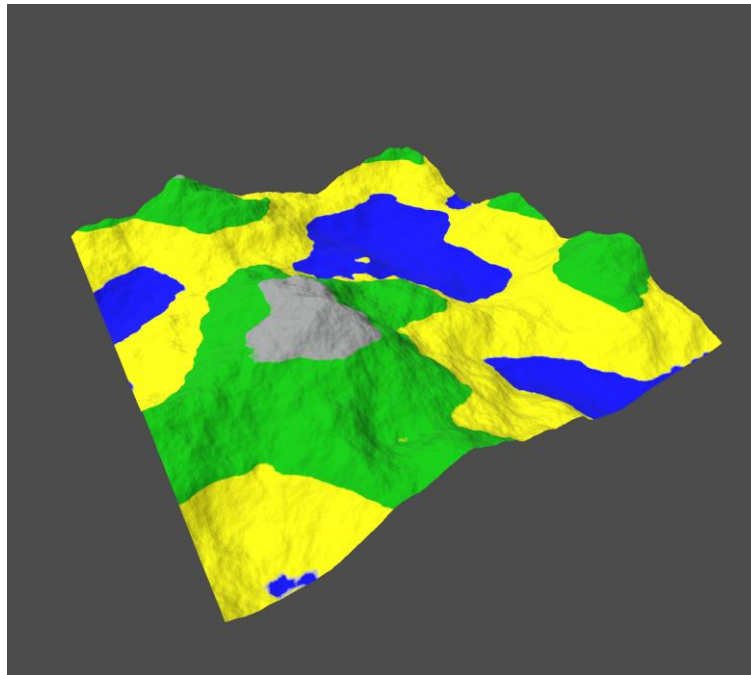
Kako bi postigli taj efekt, u sjenčaru definiramo teksturu `GradientTexture1D` (Slika 3.1). Ona nam omogućuje definiranje skupa boja koji se raspoređuje u jednodimenzionalni gradijent. Zatim sjenčar u `fragment` funkciji (Kôd 3.1) dohvaća visinu svake točke. Ta se vrijednost gledao kao koordinata za dohvaćanje boje iz `GradientTexture1D` teksture. Dobivenu boju se pridjeljuje u `ALBEDO` atribut kako bi točka poprimila tu boju. Kako ne bi dobili čudno miješanje boja na prijelazima terena, možemo isključiti interpolaciju s čime dobivamo nagle prijelaze između boja (Slika 3.2).



Slika 3.1 GradientTexture1D tekstura

```
void fragment() {  
    float height = texture(noise, shrunk_uv).r;  
    ALBEDO = texture(color_gradient, vec2(height, 1.0)).rgb;
```

Kôd 3.1 Bojanje materija unutar fragment funkcije



Slika 3.2 Obojani PlainMesh pomoću jednog gradijenta

3.2. Implementacija bioma

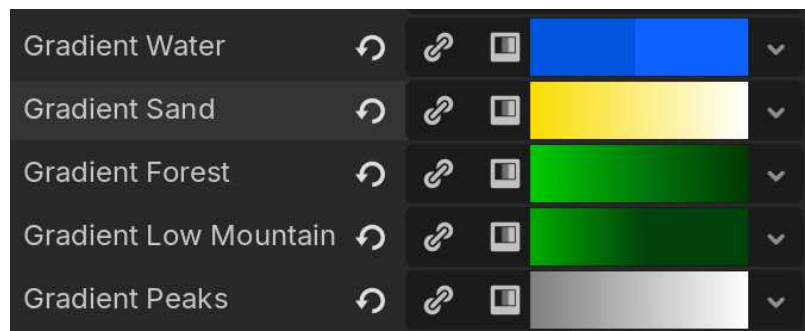
Bojanje terena samo prema visini nije dovoljno za postizanje prirodnog izgleda zato što dijelovi terena iste visine ću uvijek biti iste boje. Kako bi se dobila veća raznolikost, osim visine potrebno je uzeti u obzir još čimbenika. Jedan od čimbenika koje možemo gledati je vlažnost terena. Ideja za vlažnost je preuzeta s RedBlobGames bloga [6]. Visina točke će određivati u koju skupinu bioma teren pripada dok vlažnost će određivati boju unutar te skupine. Na taj način moguće je dobiti različite boje na istoj visini terena. Kako bi postigli taj efekt potrebno je u sjenčaru definirati za svaku visinsku skupinu `GradientTexture1D` teksturu, analogno kao u prošlom poglavlju, s bojama karakterističnim za tu visinu. U fragment funkciji (Kôd 3.2), ovisno o visini točke potrebo je računati boju s drugačijim gradijentom. Za potrebe ovog rada definirano je 5 skupina koje predstavljaju vodu, pijesak, šumu, planinu i snježne vrhove kao što vidimo na slici (Slika 3.3). Vlažnost terena možemo simulirati s još jednim dodatni šumom, koji ima drugačiju frekvenciju i početnom vrijednosti generatora od šuma korištenog za visinsku mapu. Taj šum je potrebno generirati pri inicijalizaciji ćelije terena zajedno s visinskom mapom. Kao što je prikazano na slici (Slika 3.4), različite vrijednosti vlažnosti omogućuju pojavu različitih boja na terenima jednake visine.

```

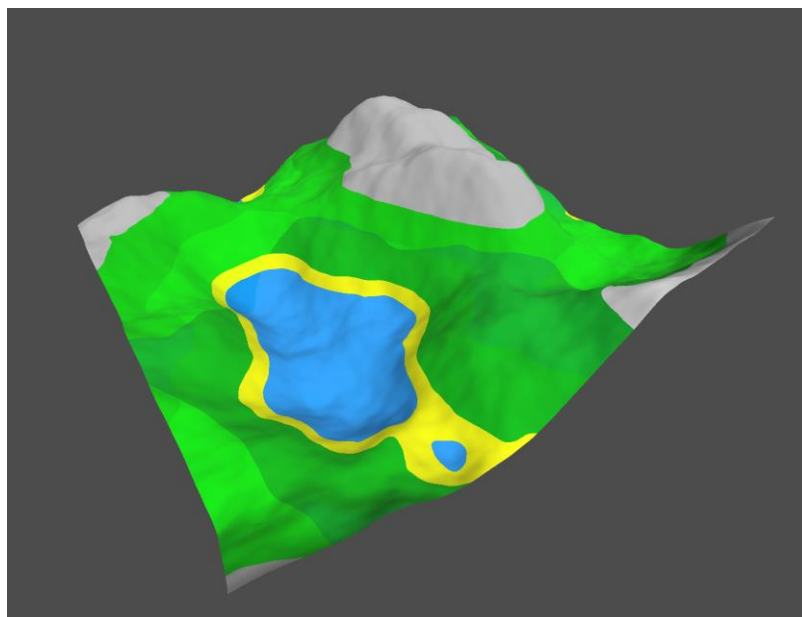
if (height < 0.35) {
    ALBEDO = texture(gradient_water, vec2(moisture, 0.5)).rgb;
} else if (height < 0.45) {
    ALBEDO = texture(gradient_sand, vec2(moisture, 0.5)).rgb;
} else if (height < 0.60) {
    ALBEDO = texture(gradient_forest, vec2(moisture, 0.5)).rgb;
} else if (height < 0.75) {
    ALBEDO = texture(gradient_mountain, vec2(moisture, 0.5)).rgb;
} else {
    ALBEDO = texture(gradient_peaks, vec2(moisture, 0.5)).rgb;
}

```

Kôd 3.2 Bojanje bioma unutar fragment funkcije



Slika 3.3 GradientTexture1D teksture

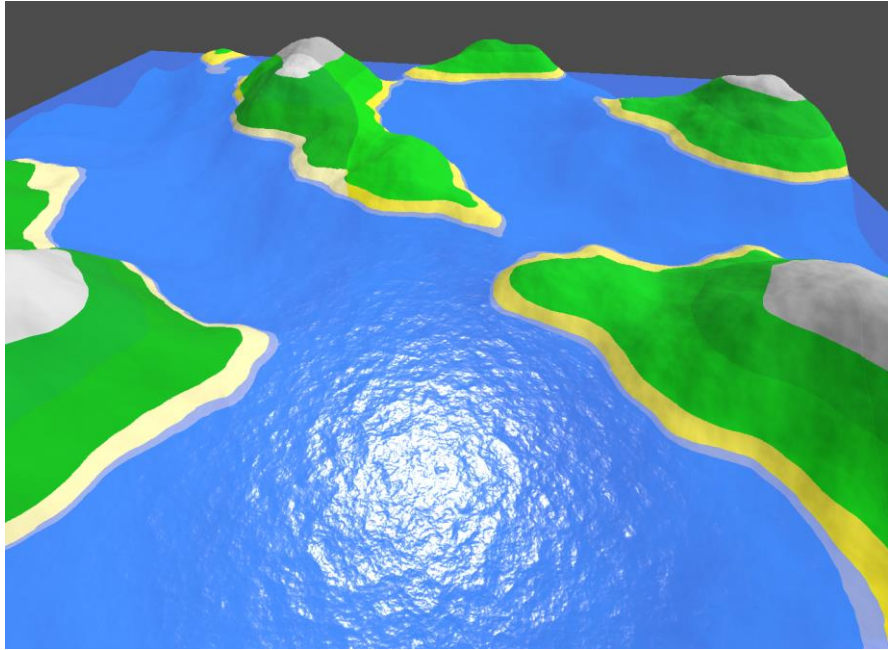


Slika 3.4 Obojani PlainMesh pomoću više gradijenat

3.3. Implementacija razine mora

Za prikaz mora potrebno je u scenu ćelije terena dodati dijete čvor `MeshInstance3D` te kao mesh postaviti `PlainMesh`. Pri inicijalizaciji ćelije, more se postavlja na zadanu visinu (koja je 36 % visine ukupnog terena) kako bi bilo u razini pijeska te dimenzije površine postavljaju kako bi prekrilo cijelu površinu ćelije.

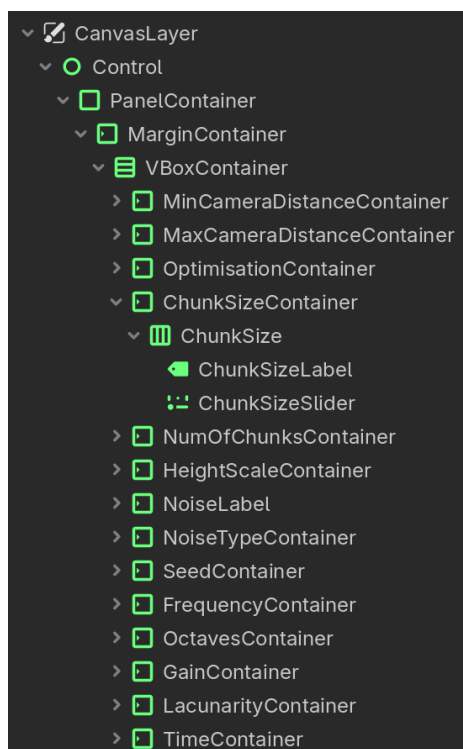
Kako bi postigli izgled mora, materijalu ravnine dodijeljena je plava djelomično prozirna boja. Za dodatno stvaranje dojma valovitosti korištena je mapa normala (engl. normal map) generirana pomoću šuma. Mapa normala je tekstura koja sadrži podatke o smjeru normale površine te omogućuje simuliranje sitnih nepravilnosti na površini bez stvarnog mijenjanja geometrije modela. Na taj način svjetlost drugačije reagira na pojedine dijelove površine, čime se stvaraju prividni valovi i postiže realističniji izgled mora, kao što je prikazano na slici (Slika 3.5). Postupak izrade mora je napravljen po uzoru na video [9].



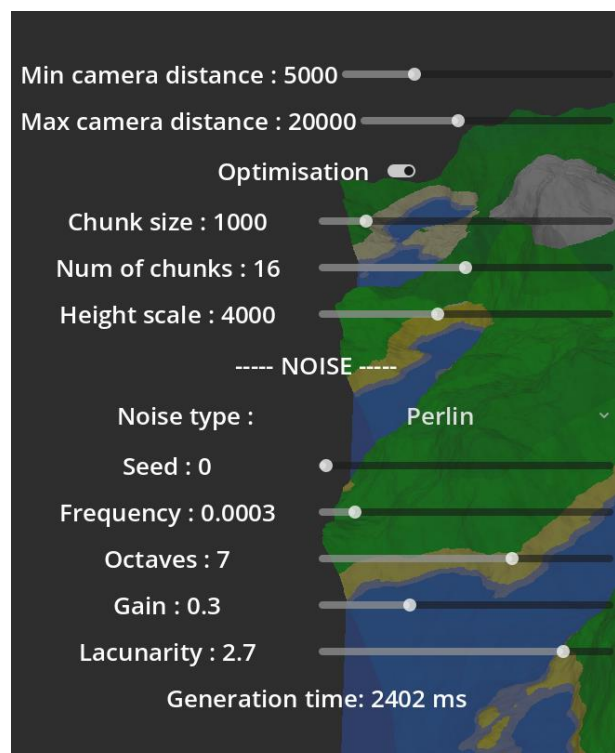
Slika 3.5 Presijavanje svjetlosti preko površine mora

4. Korisničko sučelje i kamera

Korisničko sučelje (Slika 4.2) u Godotu se ostvaruje pomoću `Control` i `CanvasLayer` čvorova. `CanvasLayer` stvara odvojeni 2D sloj za crtanje elemenata, dok `Control` čvorovi se brinu za sve elemente koje pripadaju grafičkom sučelju, odnosno ne utječu nikako s ostalim 2D ili 3D elementima. Struktura korisničkog sučelja organizirana je hijerarhijski kako bi se elementi automatski rasporedili i prilagođavali veličini prozora (Slika 4.1). Glavni `Control` čvor predstavlja osnovu sučelja, unutar kojeg se nalazi `PanelContainer` koji služi kao spremnik za elemente sučelja. Unutar njega nalazi se `MarginContainer`, koji dodaje razmak između ruba panela i njegovog sadržaja, dok se `VBoxContainer` koristi za slaganje elemenata jedan ispod drugoga. Djeca `VBoxContainer` su dodatni spremnici za svaki atribut. Svaki od njih se sastoji od `TextLabel` i `Hslider` čvorova te predstavlja zasebnu cjelinu sučelja za podešavanje određenog parametra (kao što je prikazano na slici (Slika 4.1) ali samo za `chunk_size` parametar). Svaki od njih je povezan sa signalima na glavnu scenu kako bi pokrenuli ponovno crtanje ako se neki atribut promijeni.



Slika 4.1 Hijerarhija čvorova korisničkog sučelja



Slika 4.2 Korisničko sučelje

Kamera je ostvarena pomoću čvora `Camera3D` koja je dijete glavne scene. Kretnja kamera je osmišljena da "kruži" oko generiranog terena. Pomoću tipki A i D se okreće oko globalne y osi, dok pomoću tipki W i S se mijenja njezin nagib po x osi. Udaljenost kamere od terena mijenja se pomoću kotačića miša. Pomicanjem kotačića prema gore kamera se približava terenu, dok se pomicanjem prema dolje udaljava. Udaljenost je ograničena vrijednostima `min_distance` i `max_distance`, čime se sprječava da se kamera približi ili udalji više nego što je predviđeno. Također kamera je postavljena da uvijek gleda prema točki oko koje se kamera okreće tako da teren uvijek ostane na centru ekrana.

5. Optimizacija generiranja ćelija

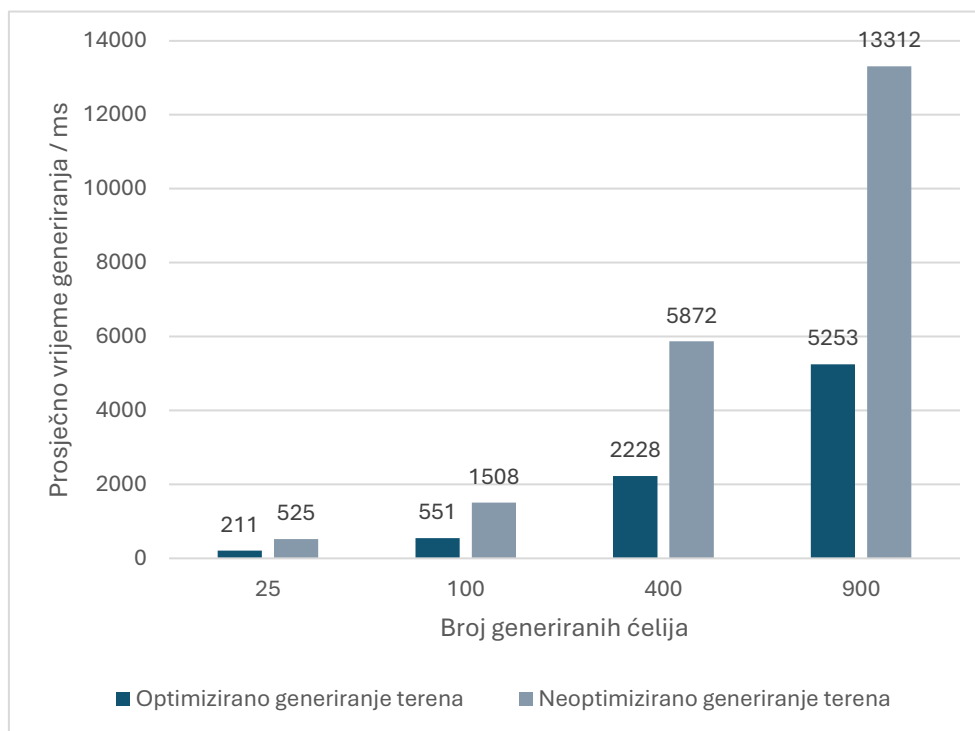
Prilikom generiranja većeg broja ćelija terena primijećeno je zastajkivanje programa. Taj problem se primarno događa zato što se cijeli proces generiranja izvodi na glavnoj dretvi. Za svaku ćeliju je potrebno izračunati vrijednosti dvaju šumova, jednog za visinu drugog za vlažnost terena, nakon čega je ćeliju potrebno instancirati i dodati u scenu. Kod generiranja manjeg broja ćelija zastajkivanje nije primjetano, no povećanjem broja ćelija čekanje se može produžiti i do 15 sekundi. Također tijekom tog vremena glavna dretva ne može odrađivati ostale zadatke te se zbog toga simulacija zamrzne.

Jedan od načina kojim možemo optimizirati proces generacije je korištenje pozadinskih dretvi. Budući da se proces izračuna vrijednosti šuma može izvoditi neovisno o drugim ćelijama, taj dio postupka je pogodan za paralelno izvođenje. Međutim, pri korištenju pozadinskih dretvi postoji jedno ograničenje. Scene i čvorovi se ne mogu sigurno instancirati i mijenjati iz pozadinske dretve, već se taj dio postupka mora izvršiti na glavnoj dretvi. Iz tog razloga proces generiranja ćelije možemo podijeliti na 2 dijela. Prvi dio (koji se odvija na pozadinskim dretvama) je generiranje vrijednosti šuma i drugi (koji se odvija na glavnoj dretvi) je instanciranje ćelije u glavnu scenu. Tako će glavna dretva biti korištena samo za pristupanje glavnoj sceni što pak znači da će biti slobodna za ostale zadatke. U Godotu postoji ugrađena klasa `WorkerThreadPool` koja nam omogućuje upravljanje pozadinskim dretvama. Ona nam omogućuje jednostavno dodavanje zadataka pomoću `add_task` funkcije nakon čaga će se oni izvoditi u pozadini. Samo je bitno naglasiti da je potrebno na kraju tog zadatka proslijediti rezultat glavnoj dretvi.

Za testiranje navedenog postupka provedeno je nekoliko mjerenja s različitim veličinama generiranog terena. Jedini parametar koji se mijenjao tijekom testiranja je broj generiranih ćelija terena, dok su svi ostali bili isti. Za svaku veličinu terena je izmjereno prosječno vrijeme generiranja s korištenjem `WorkerThreadPool` i bez njegovog korištenja. Konfiguracija računala na kojem su mjereni rezultati: AMD Ryzen 7 6800HS, 16GB RAM (6400MHz), AMD Radeon(TM) Graphics.

Rezultati optimiziranja pomoću pozadinskih dretvi su prikazani na slici (Slika 5.1). Vidimo da je za sve veličine terene uz korištenja pozadinskih dretvi vrijeme generiranja skoro prepolovljeno u odnosu na ono koje se u potpunosti odvijalo na glavnoj dretvi. Vidimo da je kod svih provedenih testiranja ostvareno smanjenje vremena generiranja od približno 60 %.

Također, korištenjem pozadinskih dretvi smo smanjili zauzetost glavne dretve te se ona mogla brinuti za ostale zadatke poput kretanje kamere.



Slika 5.1 Rezultati optimiziranja pomoću pozadinskih dretvi

Zaključak

U ovom radu implementirana je proceduralno generiranje terena pomoću grafičkog pogonskog sustava Godot. Cilj rada bio je istražiti načine na koje se pomoću proceduralnog šuma može automatizirati stvaranje velikih i raznolikih terena te proučiti način rada programsko alata Godot.

Za generiranje oblika terena korišten je Perlinov šum, čije su vrijednosti iskorištene za izradu visinske mape. Dobivena mapa se zatim koristila uz sjenčare kako bi se pomicali vrhovi terena kako bi teren dobio na trodimenzionalnosti. Za poboljšanje vizualnog izgleda terena su također korišteni sjenčari. Oni su omogućili bojanje terena po njegovoj visini i vlažnosti. Kako bi se omogućilo lako mijenjanje parametra generacije napravljeno je jednostavno korisničko sučelje te jednostavne kontrole kamere. Na kraju rada je objašnjen proces optimizacije generiranja terena koristeći pozadinske dretve.

Dobivenim rezultatom sam zadovoljan jer je ostvaren cilj rada, odnosno izrađen je sustav generiranja 3D terena pomoću proceduralnog šuma. Rad je pokazao da se uz pomoć Perlinovog šuma i sjenčara mogu stvoriti veliki i vizualno raznoliki tereni. Također, upoznao sam se s programskim alatom Godot pogotovo s radom njegovih sjenčara.

Ipak rad ostavlja prostor za daljnji razvoj i unapređenje. Trenutačno se dodatno može optimizirati proces generiranja tako da se ćelije terena koje nisu vidljive ili su predaleko od kamere ne iscrtavaju. Kako bi se dodatno optimizirala simulacija, udaljenim ćelijama terena se može smanjiti količina detalja (engl. level of detail). Također, dodavanjem kolizija na generirani teren bi se omogućilo daljnje korištenje istog u interaktivnim simulacijama i video igrima. Dodavanjem vegetacije, stijena ili drugih objekata bazirano na visini i biomu bi se doprinijelo realističnosti i raznolikosti terena.

Cjelokupni projekt nalazi se na poveznici: <https://github.com/j-ramljak/Procedural-Terrain-Generation>

Literatura

- [1] Wikipedia, »Procedural generation,« Wikipedia, The Free Encyclopedia, [Mrežno]. Available: https://en.wikipedia.org/wiki/Procedural_generation. [Pokušaj pristupa 27. kolovoza 2026].
- [2] Wikipedia, »Gradient noise,« Wikipedia, The Free Encyclopedia, [Mrežno]. Available: https://en.wikipedia.org/wiki/Gradient_noise . [Pokušaj pristupa 27. kolovoza 2026].
- [3] Wikipedia, »Perlin noise,« Wikipedia, The Free Encyclopedia, [Mrežno]. Available: https://en.wikipedia.org/wiki/Perlin_noise. [Pokušaj pristupa 27. kolovoza 2026].
- [4] A. Biagioli, »Understanding Perlin Noise,« [Mrežno]. Available: <https://adrianb.io/2014/08/09/perlinnoise.html> . [Pokušaj pristupa 27. kolovoza 2026].
- [5] codat, »How to Godot: Infinite Procedural Terrain Generation,« Youtube, 29. lipnja 2019. [Mrežno]. Available: <https://www.youtube.com/watch?v=rWeQ30h25Yg&t=1449s> . [Pokušaj pristupa 27. kolovoza 2026].
- [6] RedBlobGames, »Making maps with noise functions,« Red Blob Games, 22. kolovza 2022. [Mrežno]. Available: <https://www.redblobgames.com/maps/terrain-from-noise/>. [Pokušaj pristupa 27. kolovoza 2026].
- [7] S. Lague, »Procedural landmass generation,« Youtube, 31. siječnja 2016. [Mrežno]. Available: https://www.youtube.com/watch?v=wbpMiKiSKm8&list=PLFt_AvWsXl0eBW2EiBt1_sxmDtSgZBxB3. [Pokušaj pristupa 27. kolovoza 2026].
- [8] Godot, »Introduction to shaders,« Godot documentation, [Mrežno]. Available: https://docs.godotengine.org/en/stable/tutorials/shaders/introduction_to_shaders.html. [Pokušaj pristupa 27. kolovoza 2026].

- [9] StayAtHomeDev, »How to Create a Water Shader 2.0 // Godot 4 Tutorial,« Youtube, 20. rujna 2024. [Mrežno]. Available: <https://www.youtube.com/watch?v=TLUt14wHrIQ> . [Pokušaj pristupa 27. kolovoza 2026].

Izjava o korištenju umjetne inteligencije

Osvrt na korištenje umjetne inteligencije: U izradi ovog završnog rada korišten je alat umjetne inteligencije Gemini te je korišten samo u svrhe uređivanja teksta i razumijevanja materije. Cjelokupna ideja i implementacija rada je isključivo vlastiti rad. Alat je korišten za ispravljanje gramatike i čitljivosti već napisanoga teksta te za pojašnjenje pojedinih pojmova, poput principa rada i implementacije sjenčara u Godotu.

Izjava o korištenju umjetne inteligencije: Potvrđujem da sam tijekom izrade ovog završnog rada koristio umjetnu inteligenciju u skladu s Politikom primjerenog korištenja umjetne inteligencije na Fakultetu elektrotehnike i računarstva, te da umjetna inteligencija nije zamijenila moje kritičko razmišljanje niti moj doprinos.