SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 229

Proceduralno generiranje objekata za otočni krajolik

Ivan Vlahov

Zagreb, lipanj 2021.

SADRŽAJ

1. Uvod

Ubrzani razvoj tehnologije u zadnjih 20 godina doveo je do velikih napredaka u području računalne grafike. Računala u stvarnom vremenu mogu izvoditi sve više operacija, pa iz tog razloga nije čudno što su moderne igre sve realističnije. Uz razvoj hardvera razvija se i područje proceduralnog generiranja koje se bavi algoritamskim kreiranjem podataka.

U računalnoj grafici, proceduralno generiranje podrazumijeva programe koji stvaraju 2D i 3D objekte, i teksture. Mnoge *open-world* igre koriste tehnike proceduralnog generiranja kako bi igračima omogućile praktički beskonačan broj različitih nivoa (primjer je igra No Man's Sky koja nudi $1.8 \cdot 10^{31}$ različitih svjetova)[9]. Osim toga, i razvojnim timovima koji rade na igrama je često lakše, brže i jeftinije napraviti program koji će brzo generirati nivoe nego trošiti previše resursa na ručnu izradu.

Jedna od čestih primjena proceduralnog generiranja jest izrada terena, pa tako i otoka. U ovom radu bit će predstavljene metode i konkretne implementacije generiranja objekata za otočni krajolik, a koje uključuju generiranje terena otoka, generiranje mora, te ostalih objekata koji se mogu u takvim krajolicima pronaći.

Prvi dio rada govorit će o generiranju terena. Bit će objašnjen algoritam Perlinovog šuma, razlozi korištenja te funkcije, metoda i algoritam izrade 3D objekta, bojanje tog objekta te neke nadogradnje na osnovni algoritam kojima možemo postići bolje rezultate. Drugi dio govorit će o generiranju mora, te će se objasniti potrebni koraci za ostvarivanje teksture mora i efekata naknadne obrade na kameri kojima se može postići kvalitetniji izgled podmorja. U trećem dijelu predstavit će se metode generiranja ostalih objekata poput stabala i morske trave. Četvrti dio govorit će o algoritmu postavljanja tih objekata u scenu kroz pregled nekoliko razreda potrebnih za to. Peti dio govorit će o algoritmu za upravljanje kamerom, odnosno za kretanje scenom. U šestom dijelu povezat će se dosad predstavljene komponente u jednu cjelinu, odnosno dovršeni program. U sedmom dijelu zaključit će se rad te će se dati kratak osvrt na daljnji rad, moguća poboljšanja i sl.

Čitav programski sadržaj ovoga rada napisan je u programskom jeziku C# za po-

kretanje u *game engineu* Unity 3D i Cg jeziku za sjenčanje, a osim toga je za izradu nekih 3D modela korišten i besplatan alat za 3D modeliranje Blender. Dovršeni program bit će objavljen na GitHub Pages stranici autora koristeći biblioteku WebGL, a programski kod će biti dostupan u GitHub repozitoriju autora[7].

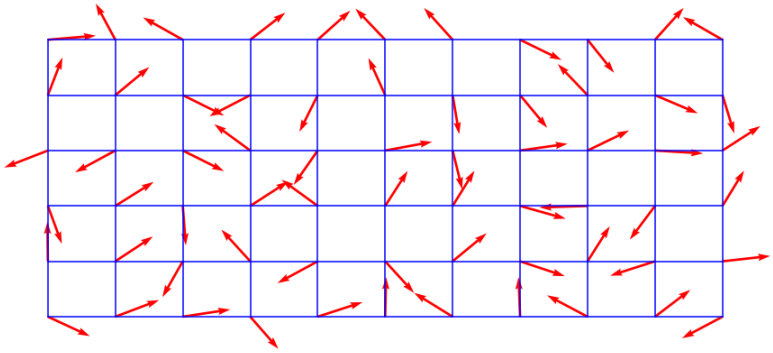
2. Generiranje terena

Ovo poglavlje bavi se prvim korakom pri proceduralnom generiranju otočnog krajolika - generiranjem terena, odnosno samog otoka. Dio ovog dijela rada inspiriran je video-serijalom Sebastiana Laguea objavljenog na YouTubeu 2019. godine [3].

2.1. Perlinov šum i visinska mapa

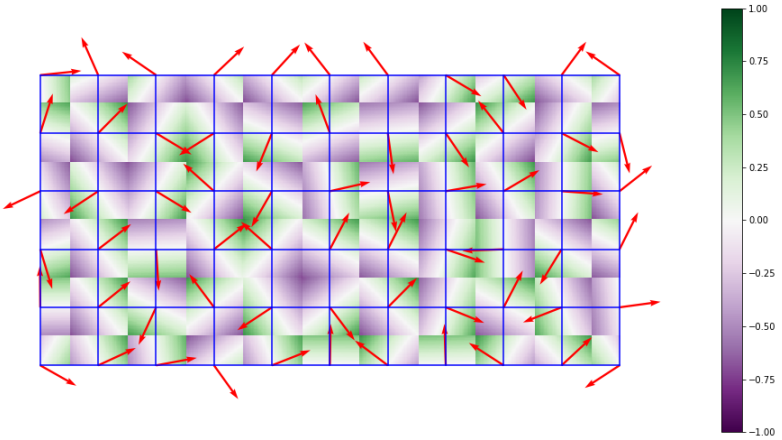
Proceduralno generiranje objekata često zahtijeva generiranje pseudoslučajnih brojeva. Promatranjem krajolika možemo zaključiti da se npr. visina planina ne ponaša baš skroz nasumično, već kao neki organizirani nasumični sustav. Upravo zbog toga za generiranje terena nećemo koristiti uniformno nasumično generiranje pseudoslučajnih brojeva, već moramo koristiti funkciju koja je globalno slučajna, a lokalno predvidljiva.

Jedna od takvih funkcija je Perlinov šum, nazvana po Kenu Perlinu koji ju razvio 1983. godine dok je radio na animiranom filmu Tron [4]. Vrijednost funkcije Perlinovog šuma za *n* ulaznih varijabli izračunava se tako da se u prvom koraku napravi n-dimenzionalna mreža te se na svaki vrh te mreže postavi jedan nasumično izabrani n-dimenzionalni normirani vektor koji predstavlja gradijent te funkcije u toj točki (kad je $n = 1$ svakom vrhu se pridružuje skalar).



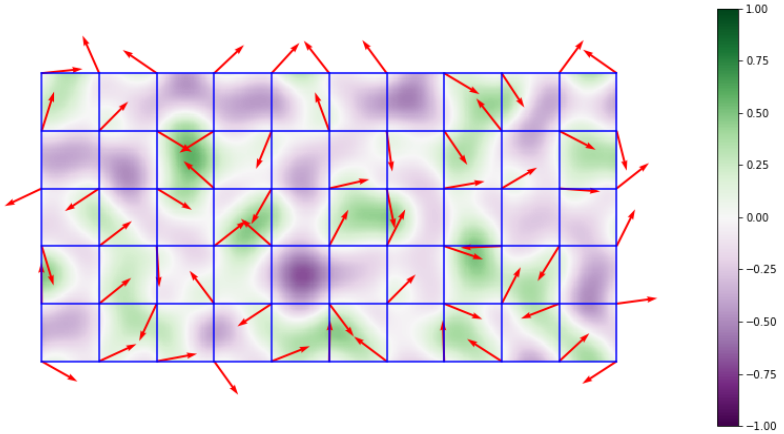
Slika 2.1: Vektori gradijenta na 2D mreži, Izvor: Wikipedia[8]

U drugom koraku se za svaku točku u tom prostoru izračuna skalarni umnožak te točke i gradijenta u njoj najbližem vrhu mreže.



Slika 2.2: Prikaz skalarnih umnožaka, Izvor: Wikipedia[8]

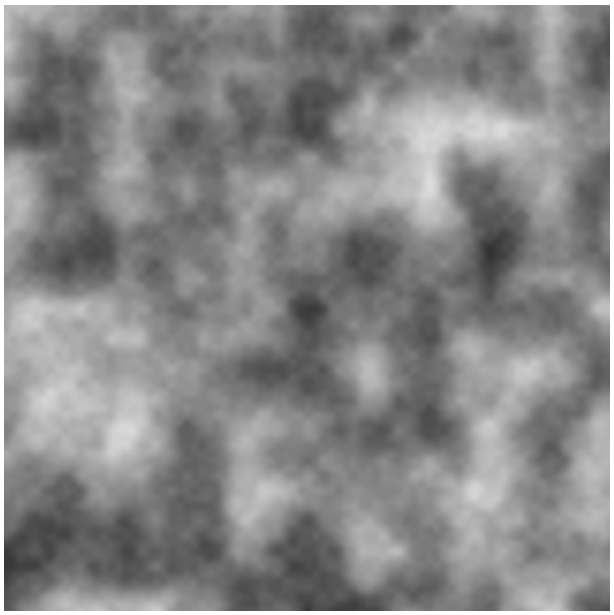
U trećem koraku se interpoliraju vrijednosti koristeći neku interpolacijsku funkciju kojoj su prva i druga derivacija jednake nuli, te se dobije glatka funkcija prepoznatljivog izgleda.



Slika 2.3: Interpolirane vrijednosti, Izvor: Wikipedia[1]

Razred `Mathf` u biblioteci Unity 3D-a ima već definiranu metodu `PerlinNoise` koju možemo koristiti da bismo generirali visinsku mapu. Razredi `MapGenerator` i `Noise` poslužit će nam za tu svrhu.

U razredu `MapGenerator` možemo definirati duljinu i širinu mape, maksimalnu visinu, *seed* vrijednost pseudoslučajnog generatora brojeva. Statički razred `Noise` sadrži metodu `GenerateNoiseMap` koja vraća polje brojeva s pomičnom točkom koji odgovaraju vrijednostima funkcije Perlinovog šuma u zadanim koordinatama.



Slika 2.4: Generirana 2D mapa na intervalu $x, y \in [0, 1]$, crna boja predstavlja vrijednost 0, a bijela 1

2.2. Smanjivanje rubova

S obzirom da je cilj ovoga rada generiranje otočnih krajolika, bilo bi dobro na neki način utjecati na funkciju Perlinovog šuma, tako da za vrijednosti bliže rubovima mape funkcija polako počne opadati, te da budemo sigurni da na samim rubovima ima vrijednost jednaku nuli. To možemo postići pomoću mape za smanjivanje rubova. Takvu mapu možemo generirati koristeći sljedeću funkciju:

$$f(x) = \frac{x^a}{x^a + (b - bx)^a}$$

gdje a i b predstavljaju parametre koje sami izabiremo. Ako koordinate generirane mape mapiramo na interval $[-1, 1]$ nekom funkcijom map, onda koristeći funkciju $f(x)$ možemo napraviti mapu za smanjivanje rubova:

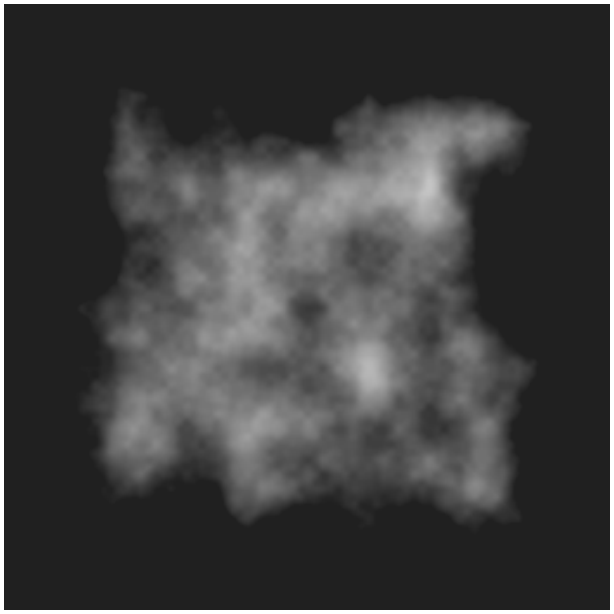
$$s(x, y) = max(f(|map(x)|), f(|map(y)|))$$

Slika 2.5: Mapa za smanjivanje rubova

Nakon što uspješno generiramo mapu za smanjivanje rubova, za svaki x i y možemo izračunati nove vrijednosti visinske mape sljedećom funkcijom:

$$v(x, y) = max(0, p(x, y) - s(x, y))$$

gdje p(x, y) predstavlja vrijednost Perlinovog šuma u točki (x, y).

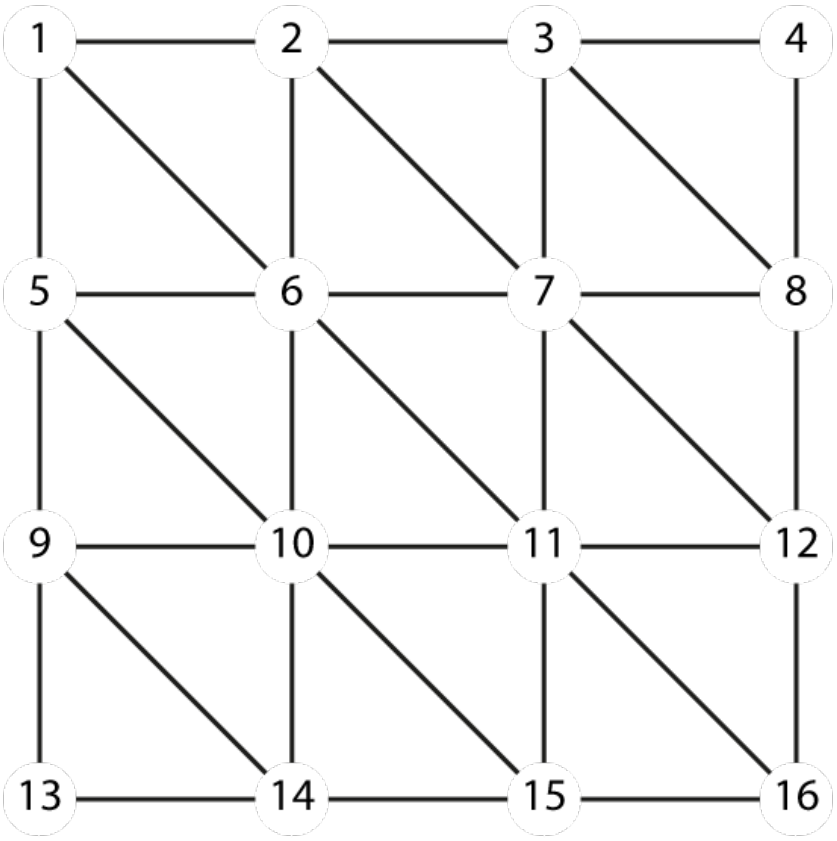


Slika 2.6: Visinska mapa nakon smanjivanja rubova

2.3. Plašt terena

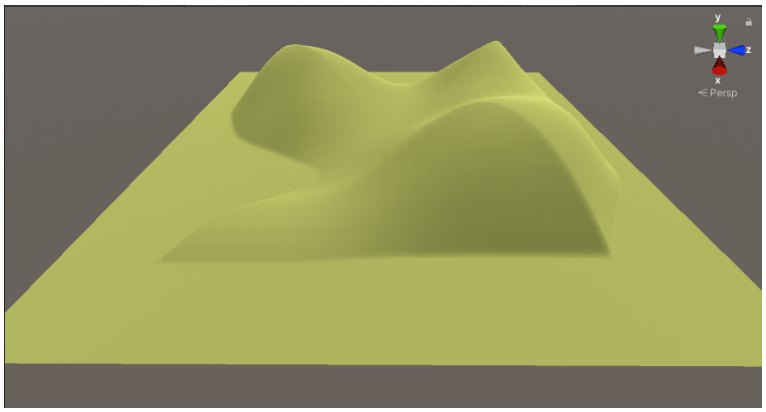
Nakon uspješnog generiranja visinske mape, možemo je iskoristiti za generiranje 3D objekta terena. Plašt terena generiramo u metodi `GenerateTerrainMesh` statičkog razreda `MeshGenerator`. Ta metoda vraća objekt tipa `MeshData` koji sadrži polje vrhova, polje trokuta i polje UV koordinata. `MeshData` također sadrži i metode za dodavanje trokuta i kreiranje `Mesh` objekta iz spremljenih podataka.

Metoda `GenerateTerrainMesh` prima visinsku mapu i faktor skaliranja visinske mape. Vrhovi plašta terena su numerirani kao na slici 2.7, slijeva nadesno po redovima. Funkcija iterira kroz sve elemente visinske mape te u polje vrhova dodaje vrh čije x i z koordinate izračuna na temelju trenutne pozicije u petlji koja iterira kroz visinsku mapu, a y koordinatu izračuna kao umnožak faktora skaliranja i trenutne vrijednosti visinske mape. Ako trenutni element visinske mape nije u posljednjem stupcu niti u posljednjem redu, u polje trokuta objekta tipa `MeshData` dodajemo dva trokuta, trokut $(i, i+w+1, i+w)$ i $(i+w+1, i, i+1)$, gdje je i indeks vrha kojeg smo upravo izračunali, a w broj vrhova u jednom redu plašta. Poredak vrhova je bitan, jer bi u protivnom trokuti bili prozirni s krive strane.



Slika 2.7: Izgled plašta s 4 reda i 4 stupca

Nakon izračunavanja svih potrebnih podataka, generirani objekt tipa `MeshData` šaljemo razredu `MapDisplay` koji sadrži metodu `DrawMesh`, koja zatim prikazuje objekt u Unity Editoru.



Slika 2.8: 3D objekt terena

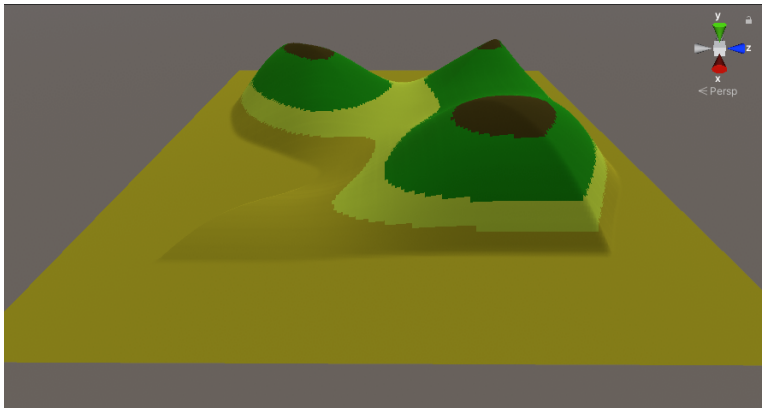
2.4. Bojanje plašta

Bojanje plašta može se izvesti na jako jednostavan način, također koristeći već izračunatu visinsku mapu. U razred `MapGenerator` dodat ćemo polje objekata tipa `TerrainType`, koji u sebi sadrže informacije o boji kojom trebaju obojiti objekt i o intervalu visine na kojem trebaju obojiti vrh u tu boju. Na temelju tih podataka se za svaki element visinske mape izračunava odgovarajuća boja, te se nova mapa boja šalje razredu `TextureGenerator`.

Metoda `TextureFromColourMap` razreda `TextureGenerator` prima polje boja i koristeći ugrađene funkcije Unity 3D-a generira i vraća odgovarajuću teksturu.

Slika 2.9: Tekstura generirana za visinsku mapu sa slike 2.6

Metodi `DrawMesh` razreda `MapDisplay` također možemo poslati i odgovarajuću teksturu koju će onda primijeniti na generirani 3D objekt.



Slika 2.10: 3D objekt terena s dodanom teksturom

2.5. Slojevi slučajnosti terena pomoću oktava

Iako se s dosad prikazanim metodama može generirati 3D objekt, ostalo je otvoreno dosta mjesta za poboljšanje. Jedno od takvih poboljšanja je dodavanje tzv. oktava. Oktave predstavljaju dodatne slojeve slučajnosti i nepravilnosti koji se dodaju generiranoj visinskoj mapi kako bi poboljšali realističnost samog objekta i postigli da objekt terena nije pregladak.

Dodavanje oktava možemo prikazati na jednostavan način funkcijom sinus. Definirajmo sljedeće funkcije:

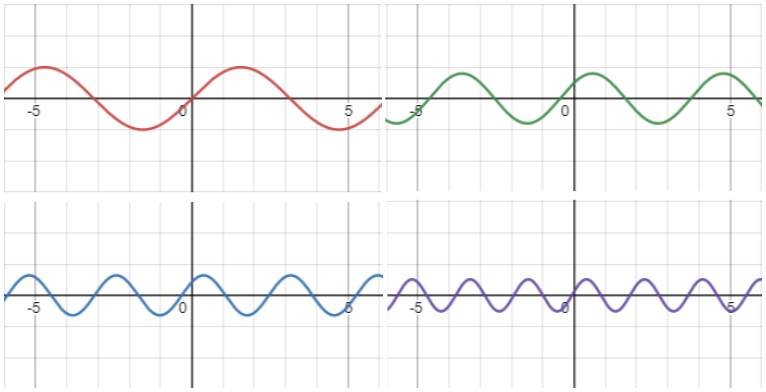
$$f_0(x) = sin(x)$$

$$f_1(x) = 0.8 \cdot sin(1.5 \cdot x + 0.68)$$

$$f_2(x) = 0.64 \cdot sin(2.25 \cdot x + 0.73)$$

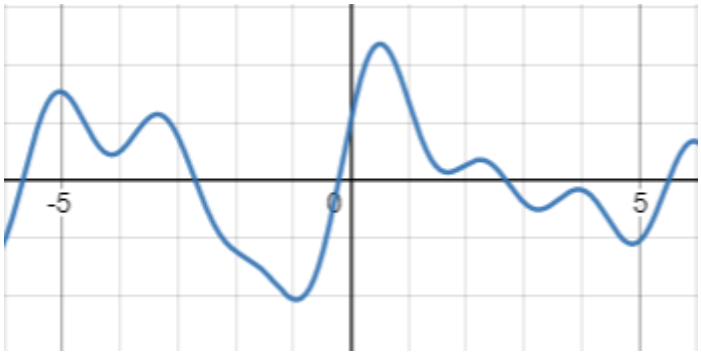
$$f_3(x) = 0.512 \cdot sin(3.375 \cdot x + 0.24)$$

Faktori uz sin su izabrani kao potencije broja 0.8, faktori uz x kao potencije broja 1.5, a pribrojnici u funkcijama 1 do 3 su izabrani nasumično.



Slika 2.11: Grafovi funkcija slijeva nadesno po redovima

Pribrojimo li vrijednosti ovih funkcija jedne drugima, dobit ćemo novu funkciju koju možemo nazvati f_s. Kada pogledamo graf te funkcije, vidjet ćemo da smo ostvarili upravo ono što smo htjeli - smanjili smo glatkoću grafa. Iste ove parametre možemo primijeniti na funkciju Perlinovog šuma u dvije dimenzije.



Slika 2.12: Graf sume funkcija

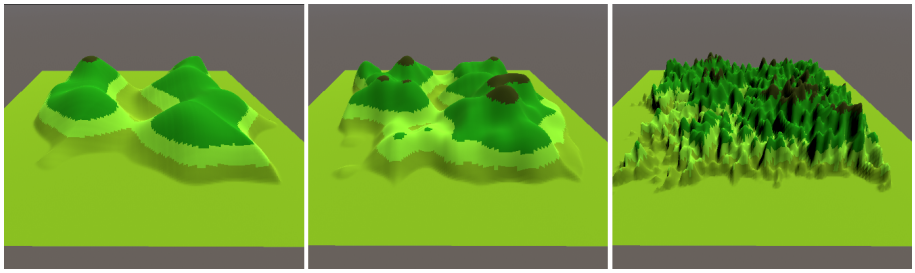
Funkciju i-te oktave na funkciji Perlinovog šuma možemo napisati kao:

$$f_i = d^i \cdot p(l^i \cdot x + rand(), l^i \cdot y + rand())$$

gdje je $p(x, y)$ funkcija Perlinovog šuma, $rand()$ je funkcija koja generira pseudoslučajan broj, a parametre d i l biramo sami. Visinsku mapu izračunavamo zbrajanjem prvih n oktava.

2.6. Kontrola udubljenja terena (lakunarnost)

Fiksiramo li vrijednost parametra d na vrijednost 0.75, možemo promatrati što se događa kad mijenjamo vrijednost parametra l.



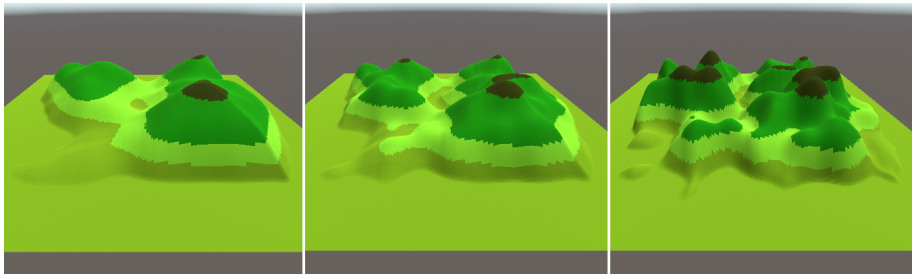
Slika 2.13: Usporedba parametra l za vrijednosti 1.5, 2.0 i 4.0

S obzirom da ovaj parametar utječe na količinu jama u terenu, nazvat ćemo ga lakunarnost (engl. *lacunarity*, lat. *lacuna*, jama, rupa)[11].

Ručnim testiranjem može se zaključiti da se najbolji rezultati dobivaju kad je parametar l u intervalu $[2.0, 2.5]$.

2.7. Dosljednost amplituda

Na sličan način možemo provjeriti utjecaj parametra d na izgled terena. Fiksirat ćemo vrijednost parametra l na vrijednost 2.0, te mijenjati vrijednost parametra d.



Slika 2.14: Usporedba parametra d za vrijednosti 0.25, 0.5 i 1.0

Iz ovoga možemo zaključiti da ovaj parametar utječe na lokalne varijacije u amplitudi, pa ga možemo nazvati dosljednošću amplituda (engl. *persistence*).

Ručnim testiranjem ovog parametra možemo zaključiti da daje najbolje rezultate kada je u intervalu $[0.4, 0.6]$.

3. Generiranje mora

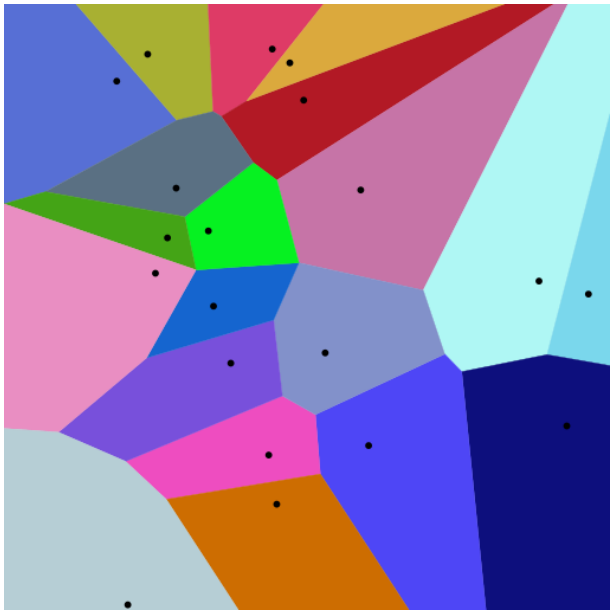
Da bismo generirali more, potrebno je napraviti tri stvari. Prva stvar je generiranje oblika morske površine, druga je bojanje površine, a treća je dodavanje učinka na prostor projekcije koji se uključuje kad se kamera spusti ispod površine mora.

More ćemo prikazati kao dvije ravnine koje se nalaze na istoj lokaciji, s tim da je jedna od njih rotirana za 180 stupnjeva po x-osi, čime postižemo vidljivost teksture s obje strane.

Implementacija generiranja mora dijelom je inspirirana video uratkom s kanala Brackeys na YouTubeu, međutim, korišteni su drugačiji alati da bi se postigli isti rezultati [1].

3.1. Voronoi sjenčar

Za generiranje površine možemo iskoristiti teksturu temeljenu na dvodimenzionalnom Voronoi dijagramu. Dvodimenzionalni Voronoi dijagram predstavlja način particioniranja ravnine tako što se izabere neki broj nasumičnih točaka na toj ravnini. Dvije točke ravnine pripadaju istoj particiji ako i samo ako dijele najbližu točku iz skupa nasumično izabranih točaka na toj ravnini, a svaka particija je obojena u svoju boju. Za potrebe ovog rada prikazat ćemo kako postići takvu teksturu koristeći sjenčare te kako uz minimalne promjene postići željenu teksturu površine mora [2].



Slika 3.1: Crnom bojom su prikazane nasumično izabrane točke, a ostalim bojama su prikazane particije, Izvor: Wikipedia[10]

Ideja algoritma ovog sjenčara jest podjela ravnine na mrežu veličine $6 \cdot 6$, te se koristeći pseudoslučajan generator brojeva generira 9 točaka - po jedna za svako susjedno polje u mreži, i jedna za samo polje u kojem se trenutna točka nalazi. Za isto polje u istom vremenu će svi njegovi susjedi generirati istu točku. Nakon što se generiraju točke, usporedi se udaljenost trenutne točke od svake od njih, te se izabire ona nasumično izabrana točka koja je najbliža točki koju trenutno gledamo. Točka se zatim oboji u odgovarajuću boju, a ako je točka jako blizu nasumično generirane točke, oboji se u crno kako bismo vidjeli gdje se generirane točke nalaze.

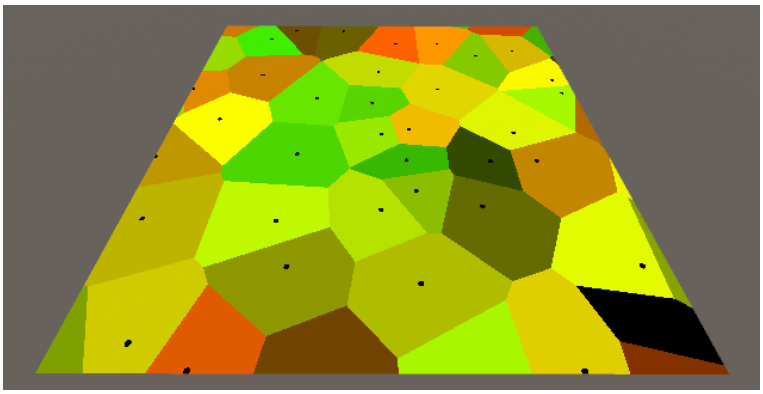
```
fixed4 frag(v2f i) : SV_Target {
  fixed4 col;
  float2 uv = i.uv;
  uv *= 6.0;
  float2 iuv = floor(uv);
  float2 fuv = frac(uv);
  float minDist = 1.0;
  for (int y = -1; y <= 1; y++) {
    for (int x = -1; x <= 1; x++) {
      float2 neighbour = float2(float(x), float(y));
      float2 pointv = random2(iuv + neighbour);
      pointv = 0.5 + 0.5*sin(_Time.z / 2.0 + 6.2831*pointv);
      float2 diff = neighbour + pointv - fuv;
      float dist = length(diff);

```

```
16          if(dist < minDist) {
17            col = float4(random2(iuv + neighbour), 0.0, 1.0);
18            minDist = dist;
19          }
20        }
21      }
22
23      if (minDist < 0.03) col = float4(0.0, 0.0, 0.0, 1.0);
24      return col;
25    }
```

Isječak koda 3.1: Algoritam generiranja Voronoi dijagrama

U retku 12 postižemo mijenjanje teksture kroz vrijeme pomicanjem nasumično izabranih točaka koristeći funkciju sinus.

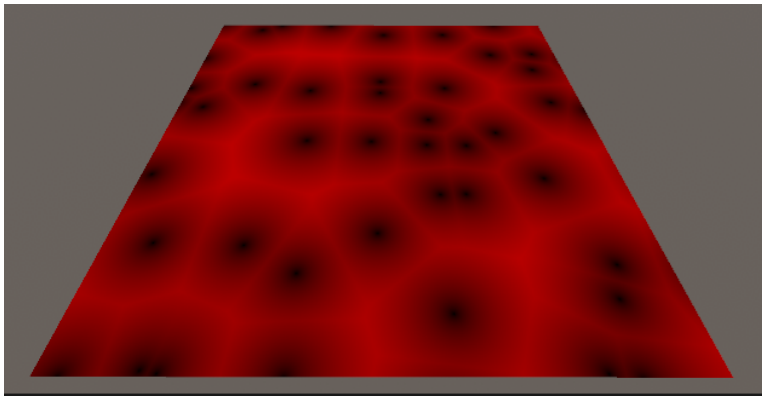


Slika 3.2: Ravnina s teksturom generiranom Voronoi sjenčarom

Varijablu `minDist` možemo iskoristiti kako bismo napravili gladak prijelaz između različitih particija ovog dijagrama. Uklonimo li liniju 17 i 23 iz Isječka koda 3.1, a prije naredbe `return` dodamo sljedeću liniju, dobit ćemo puno zanimljiviju teksturu.

```
1   col = float4(minDist * 0.5, 0.0, 0.0, 1.0);
```

Isječak koda 3.2: Gladak prijelaz između crne i crvene boje

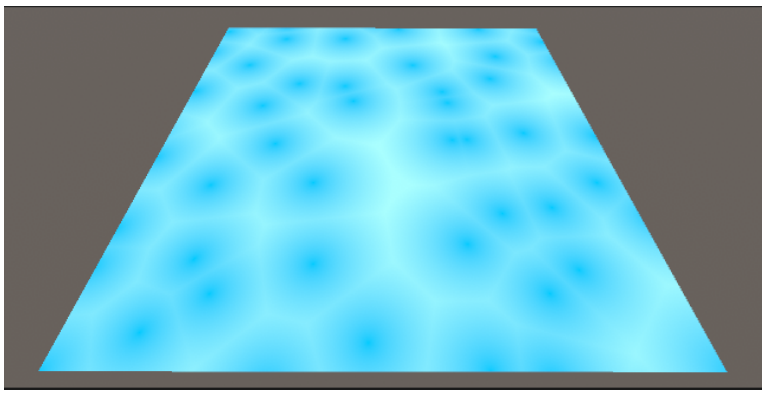


Slika 3.3: Nova tekstura koja prikazuje udaljenost točaka crvenom bojom

Na sličan način možemo izvršiti interpolaciju između dvije boje. Zamijenimo li liniju iz Isječka koda 3.2 sa sljedećom linijom, dobit ćemo željenu teksturu.

```
col = minDist * float4(0.4, 1.0, 1.0, 1.0) + (1.0 - minDist) *
 float4(0.0, 0.6, 1.0, 1.0);
```

Isječak koda 3.3: Interpolacija između različitih nijansi plave



Slika 3.4: Željena tekstura prikazana na ravnini

3.2. Prozirnost

S obzirom da je morska površina uglavnom prozirna, da bismo omogućili prozirnost, u sjenčaru trebamo dodati sljedeći kod, koji omogućuje prikaz alfa vrijednosti RGBA boja.

```
Tags {"Queue" = "Transparent" "RenderType" = "Transparent" }

ZWrite Off
```

```
4   Blend SrcAlpha OneMinusSrcAlpha
```

Isječak koda 3.4: Kod za omogućavanje prozirnosti

Nakon što smo dodali te linije koda, možemo promijeniti alfa vrijednosti boja koje interpoliramo na 0.5 te tako ostvariti prozirnost.

3.3. Oblik morske površine

Nakon što smo generirali teksturu morske površine, trebamo dodati varijacije u y koordinatama vrhova 3D objekta morske površine, te ih mijenjati kroz vrijeme kako bismo ostvarili učinak valova na morskoj površini. Takav učinak možemo ostvariti korištenjem sjenčara vrhova. U postojećem sjenčaru možemo dodati parametre `_DispTex`, `_Amount` i `_DispTexScale`.

```
1  [NoScaleOffset] _DispTex("Displacement Texture", 2D) = "white" {}
2  _Amount("Displacement Amount", Range(0,20)) = 0.5
3  _DispTexScale("Displacement Scale", Range(0, 1)) = 1.0
```

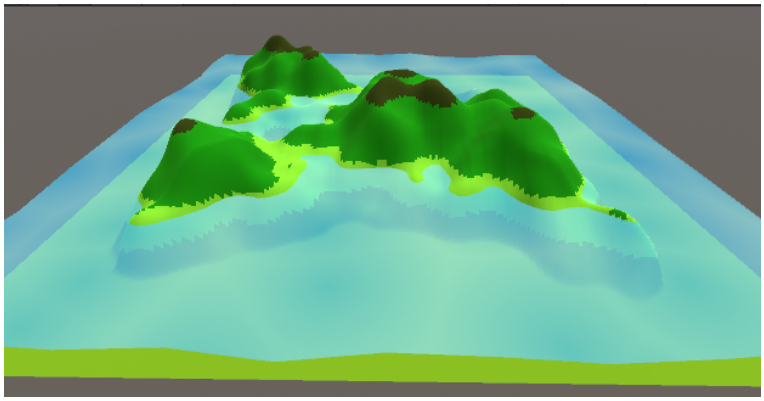
Isječak koda 3.5: Parametri sjenčara

U `_DispTex` spremamo teksturu pomoću koje možemo odrediti koliko će gdje objekt površine mora biti izdignut. `_Amount` je faktor skaliranja y osi, a parametar `_DispTexScale` je faktor skaliranja x i z osi teksture `_DispTex`.

```
1  v2f vert(appdata v)
2  {
3    v2f o;
4    o.vertex = UnityObjectToClipPos(v.vertex);
5    o.uv = v.uv;
6
7    o.vertex.y += (tex2Dlod(_DispTex, float4((_DispTexScale + 0.125
     * sin(0.5 * _Time)) * o.uv.xy, 0, 0)).r - 0.25)  * _Amount;
8    return o;
9  }
```

Isječak koda 3.6: Sjenčar vrhova za morsku površinu

U retku 7 Isječka koda 3.6 se pomoću dane teksture računa pomak po y osi funkcijom sinus koristeći vrijeme kao argument, te se dobije učinak valova.

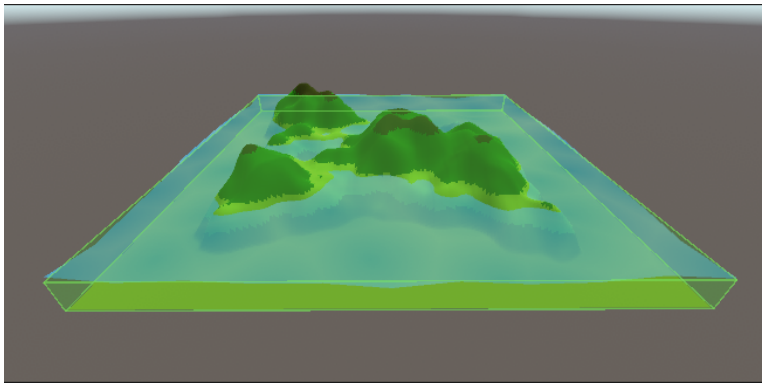


Slika 3.5: Izgled scene nakon dodavanja morske površine

3.4. Naknadna obrada u prostoru projekcije

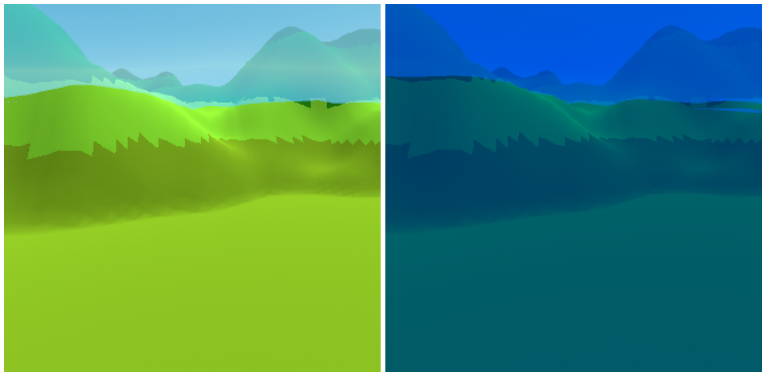
S obzirom da planiramo omogućiti kretanje kamere u sceni, potrebno je omogućiti uključivanje učinka na kameri prilikom spuštanja ispod površine mora. Takav učinak može se ostvariti korištenjem paketa *Post Processing* iz standardne liste dodatnih paketa Unity 3D-a.

Dodavanjem objekta *Post-Processing Volume* u scenu možemo odrediti područje u kojem želimo omogućiti *post-processing* učinke. Skripta unutar ovog objekta provjerava nalazi li se kamera unutar zadanog područja koristeći skriptu Box Collider, te omogućuje učinke u slučaju da se kamera nalazi unutar područja. U ovom slučaju postavit ćemo Box Collider tako da pokriva područje od dna terena do površine mora.



Slika 3.6: Zelenom bojom označeno je područje djelovanja učinka

Na objekt *Post-Processing Volume* možemo dodati učinke koje želimo primijeniti. Učinak Color Grading omogućuje nam utjecanje na boje koje kamera vidi, te njime možemo omogućiti učinak izgleda podmorja. Smanjivanjem temperature boje i pomicanjem boja prema plavim nijansama možemo postići zadovoljavajući učinak.



Slika 3.7: Usporedba izgleda scena bez učinka i s učinkom

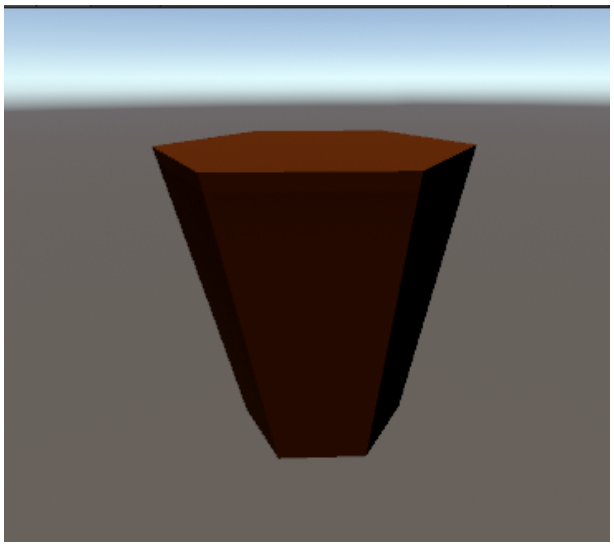
4. Ostali objekti

Ovo poglavlje govori o načinima generiranja ostalih objekata koji se mogu koristiti pri izradi 3D otočnih krajolika, kao što su stabla i morska trava. Svi razredi iz ovog poglavlja nasljeđuju apstraktni razred `GeneratorBase`, o kojem će biti govora u sljedećem poglavlju.

4.1. Stabla

U ovom dijelu ovog poglavlja prikazat ćemo metodu generiranja 3D palmi. Generiranje palme podijelit ćemo u dva dijela - generiranje debla i generiranje krošnje. Metoda `Generate` za generiranje stabla nalazi se u razredu `TreeGenerator`.

Da bismo generirali deblo palme, u Blenderu je izrađen 3D objekt jednog "režnja" debla, pomoću čijih kopija možemo izgraditi čitavo deblo.



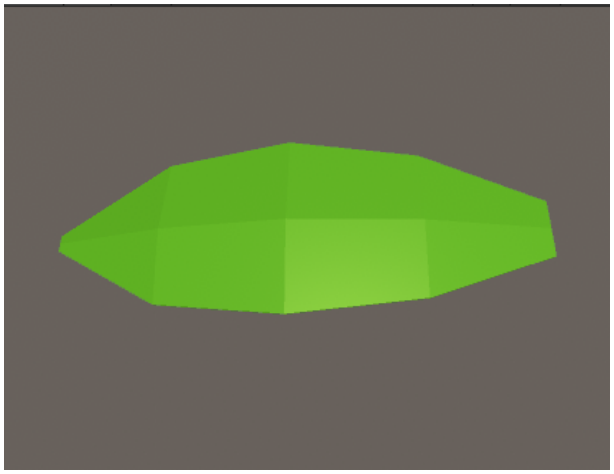
Slika 4.1: 3D objekt režnja debla

S obzirom da želimo da svako stablo izgleda drugačije, debla ćemo generirati tako da budu malo nagnuta, zaobljena i rotirana oko y-osi. Da bismo postigli takav izgled,

trebamo pronaći neku funkciju koja za određeni nasumični parametar daje oblik debla. Ta funkcija se također mora moći lako derivirati kako bismo mogli izračunati rotaciju svakog dodanog režnja u određenim točkama. Također, poželjno bi bilo da funkcija na visini 0 uvijek poprima vrijednost 0. Jedna takva funkcija je funkcija $f(x) = (cx^2)/4$ gdje c predstavlja nasumično izabran parametar. Ubacivanjem vrijednosti y-osi u ovu funkciju možemo izračunati pomak po x-osi. Također, derivacija ove funkcije, $f'(x) = (cx)/2$ je dovoljno jednostavna da pomoću nje lako možemo izračunati rotaciju svakog režnja. Želimo li generirati stablo koje se sastoji od n režnjeva, pozicija i-tog režnja bit će $(f(i/n), i/n, 0)$, a rotacija po x, y i z-osima će biti $(tan^{-1}(f'(i/n)), 0, 0)$.

Slika 4.2: Izgled nasumično generiranog debla

Krošnje stabala ćemo generirati na sličan način kao i debla. U Blenderu je izrađen objekt jedne grane krošnje čije će skalirane i rotirane kopije činiti čitavu krošnju.



Slika 4.3: 3D objekt grane stabla

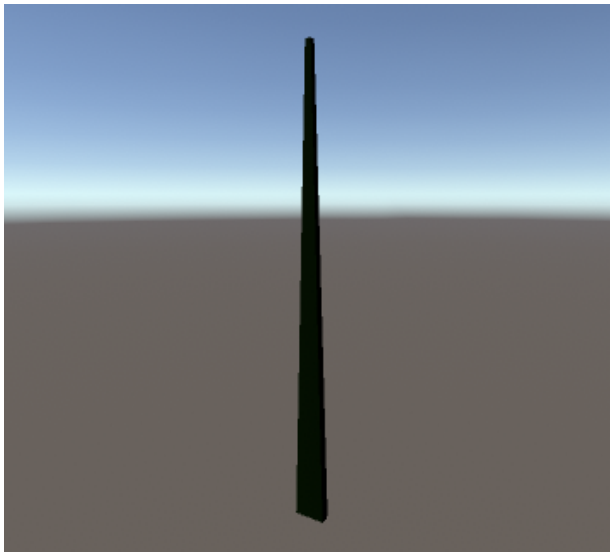
Središte rotacije objekta grane ne nalazi se u sredini, već na vrhu objekta kako bi pozicioniranje i određivanje rotacije učinili lakšom. Želimo li generirati krošnju s n grana, generirat ćemo je tako da napravimo dva reda grana od kojih će se donji red sastojati od $\lfloor n/2 \rfloor$ grana, a gornji od $\lceil n/2 \rceil$ grana. Rotacija i-te grane oko y-osi bit će jednaka $(i + rand()) \cdot 360/m$, gdje $rand()$ predstavlja slučajnu funkciju koja vraća broj u intervalu $[-0.5, 0.5]$, a m predstavlja broj grana u tom redu. Rotacija oko z-osi nasumična je vrijednost u intervalu $[0, 7.5]$ za donji red, odnosno $[15, 30]$ za gornji red. Također, objekti se nasumično skaliraju tako da grane u donjem redu budu veće od onih u gornjem redu.

Nakon što se generira i krošnja, postavi se na vrh debla te se cijeli objekt nasumično zarotira oko y-osi i postavi na poziciju zadanu u funkciji `Generate`.

Slika 4.4: Generirano stablo

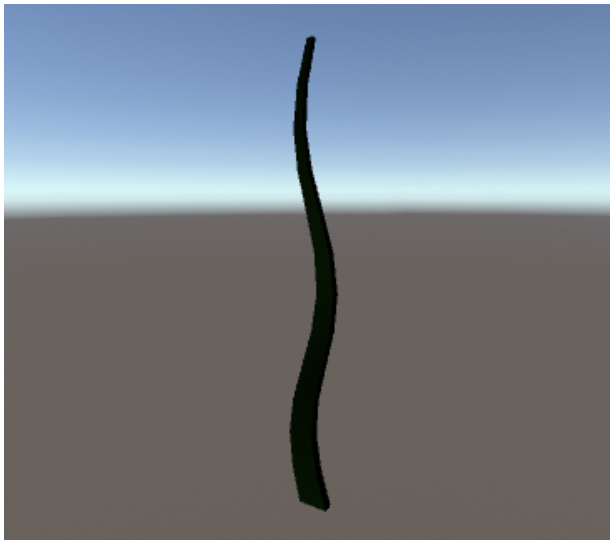
4.2. Morska trava

S obzirom da generiramo otočni krajolik, trebamo dodati i neke objekte u podmorje. U podmorju se često može vidjeti morska trava, tako da ćemo u ovom dijelu baviti proceduralnim generiranjem područja s morskom travom. Za potrebe generiranja jedne travke u Blenderu je napravljen 3D model čijim vrhovima možemo programski manipulirati kako bismo ostvarili travke različitih veličina i oblika. Generiranje morske trave implementirano je u metodi `Generate` razreda `SeaweedGenerator`.



Slika 4.5: Travka prije manipuliranja vrhovima

Pošto su travke morske trave uglavnom zaobljene pod utjecajem morskih struja, u programskom kodu možemo odrediti koliko ćemo koji vrh 3D objekta pomaknuti po x i z-osima kao funkciju vrijednosti y-osi. Kako se radi o morskim strujama i valovima, prigodno je koristiti funkciju sinus za ostvarivanje takvog učinka. Za svaku od x i z-osi definirat ćemo vlastitu funkciju s nasumično izabranim parametrima amplitude i frekvencije. Nakon što odredimo nove koordinate vrhova, postavimo ih u objektu tipa `Mesh`, te nanovo izračunamo normale koristeći metodu `RecalculateNormals` razreda `Mesh`.



Slika 4.6: Travka nakon manipuliranja vrhovima

Nakon uspješnog generiranja jedne travke, u metodi `Generate` dodat ćemo i nasumičnu vrijednost koja označava broj generiranih travki u okolini zadane pozicije. Za svaku od n travki nasumično odredimo poziciju u odnosu na zadanu poziciju te ju generiramo.

Slika 4.7: Generirano područje s morskom travom

5. Pozicioniranje objekata

U ovom poglavlju prikazat će se način na koji je ostvareno pozicioniranje objekata u sceni tako što će biti objašnjen rad razreda koji to omogućuju.

5.1. Razred GeneratorBase

Apstraktni razred `GeneratorBase` sadrži apstraktnu metodu `Generate` pomoću koje konkretni generatori generiraju nove objekte na lokaciji `position`.

5.2. Struktura GeneratableObjectParams

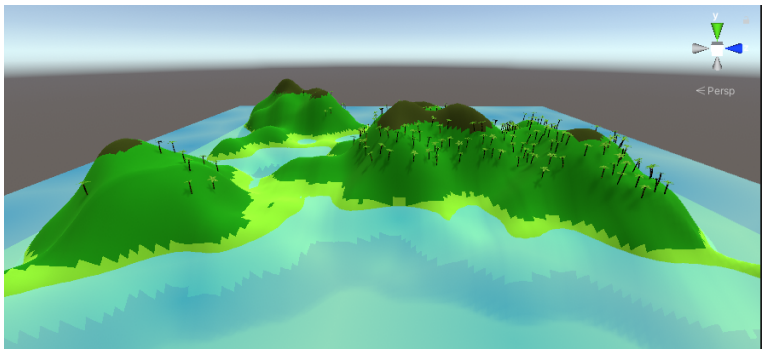
Struktura `GeneratableObjectParams` sadrži parametre potrebne za generiranje objekta koristeći neku od implementacija razreda `GeneratorBase`. Sadrži donju i gornju granicu visine na kojoj se objekt smije generirati (npr. stabla nema smisla generirati u moru, isto kao što morsku travu nema smisla generirati na vrhu planine), faktor skaliranja vjerojatnosti pojavljivanja kojim se omogućuje ograničavanje pojavljivanja nekog objekta na tek par instanci, te *seed* vrijednost generatora pseudoslučajnih brojeva.

Također, ovoj strukturi je dodan atribut `System.Serializable` kojim se omogućuje jednostavno uređivanje njegovih vrijednosti u Unity Inspectoru.

5.3. Razred ObjectPositioner

Razred `ObjectPositioner` sadrži javno polje `objects` u kojem se nalaze objekti tipa `GeneratableObjectParams` čiji broj i vrijednosti unosimo kroz Unity Inspector. U metodi `GenerateAll` ovog razreda napravi se mapa vjerojatnosti Perlinovog šuma te se metodi `Update` prenese informacija da generiranje može započeti. Metoda `Update` poziva metodu `GenerateOne` određen broj puta, te joj šalje x i y

koordinate mape vjerojatnosti. Metoda `GenerateOne` određuje koje objekte može generirati na temelju podataka iz visinske mape. U slučaju da je visina u intervalu koji odgovara nekom od objekata, taj objekt se generira na zadanoj lokaciji s vjerojatnošću umnoška faktora skaliranja vjerojatnosti i vrijednosti iz mape vjerojatnosti. Generiranje objekata raspršeno je kroz više poziva metode `Update` kako bi se smanjila šansa da program uspori prilikom stvaranja objekata. Broj objekata koji se generiraju po pozivu metode `Update` upisan je u javnoj varijabli `objectsPerFrame`.



Slika 5.1: Generirana stabla za interval $[0.4, 0.55]$ i faktor skaliranja vjerojatnosti 0.5

6. Upravljanje kamerom

Upravljanje kamerom omogućeno je razredom `CameraMovement`. Taj razred dodan je kao komponenta objekta MainCamera, a pomoću njega upravlja se korisničkim tipkovničkim ulazom te se ovisno o pritisnutoj tipki kamera rotira ili pomiče. Razred ima dvije javne varijable brojeva s pomičnom točkom: `moveSpeed` i `rotateSpeed` kojima u pregledniku možemo mijenjati vrijednosti. Osim toga sadrži i javni objekt `target` tipa `Transform` - gledište kamere.

6.1. Rotacija kamere

Najjednostavniji način za rotiranje kamere jest rotiranje oko lokalne x i y-osi (y-os je u Unity 3D-u vertikalna os). Pritiskom na tipke `W` i `S` smanjujemo, odnosno povećavamo vrijednosti rotacije oko lokalne x-osi, dok pritiskom na tipke `A` i `D` upravljamo vrijednostima rotacije oko lokalne y-osi.

```
1  if(Input.GetKey(KeyCode.S)) {
2    pitch += rotateSpeed * Time.fixedDeltaTime;
3    if(pitch > 90.0f) pitch = 90.0f;
4    rotationChanged = true;
5  }
```

Isječak koda 6.1: Rotacija kamere oko x osi

U ovom primjeru *pitch* je ime varijable u kojoj je spremljena rotacija oko x-osi, a `Time.fixedDeltaTime` predstavlja fiksni vremenski period između dva poziva funkcije `FixedUpdate` u kojoj se nalazi ovaj kod. Uvjet u trećem retku osigurava da se kamera ne okrene naopako, te je sličan uvjet postavljan u funkciji za smanjivanje rotacije. Kod za rotaciju oko y-osi je sličan (mijenja se varijabla `yaw`), ali u njemu nema nikakvih uvjeta jer se kamera slobodno može rotirati oko lokalne y-osi. Četvrti redak postavlja bool varijablu `rotationChanged` na istinitu vrijednost, tako da se na kraju može promijeniti rotacija `Transform` komponente kamere:

```
1  if(rotationChanged) {
```

```
    transform.eulerAngles = new Vector3(pitch, yaw, transform.
   eulerAngles.z);
  }
```

Isječak koda 6.2: Postavljanje vrijednosti rotacije

6.2. Kretanje kamere

Kretanje kamere omogućeno je na veoma jednostavan način. U djecu objekta MainCamera dodan je prazan objekt Target na poziciju $(0, 0, 1)$. Rotacijom kamere globalna pozicija komponente `Transform` objekta Target se mijenja, međutim, njegova pozicija u odnosu na objekt MainCamera ostaje ista. Ova činjenica može se iskoristiti da bi se s jako malo koda ostvarilo dinamičko mijenjanje vektora smjera kretnje kamere. Oduzimanjem vektora globalne pozicije objekta od pozicije kamere dobije se vektor smjera kretnje, te se nakon normiranja i umnoška s brzinom kretnje dobije vektor brzine kretnje.

```
if(rotationChanged) {
  moveDirection = target.position - transform.localPosition;
  moveDirection = Vector3.Normalize(moveDirection);

  moveDirection *= moveSpeed;

  rotationChanged = false;
}
```

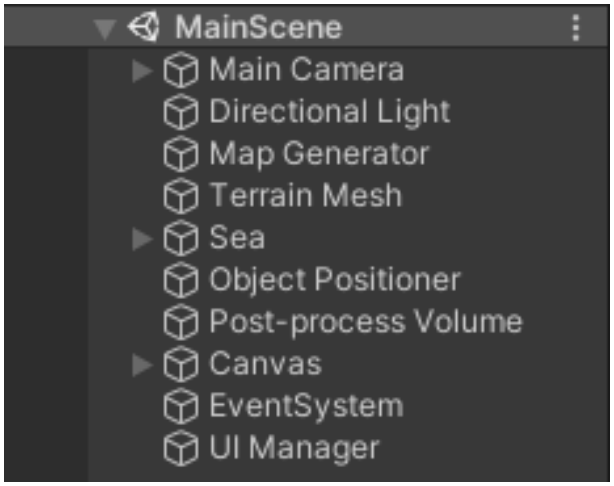
Isječak koda 6.3: Određivanje vektora brzine kretnje

Nakon što izračunamo vektor brzine kretnje, možemo ga iskoristiti kako bismo pomicali kameru. Pritiskom na tipku `K` krećemo se u pozitivnom smjeru vektora brzine, a pritiskom na tipku `L` krećemo se u negativnom smjeru.

7. Povezivanje u cjelinu i izvoz programa

7.1. Glavna scena

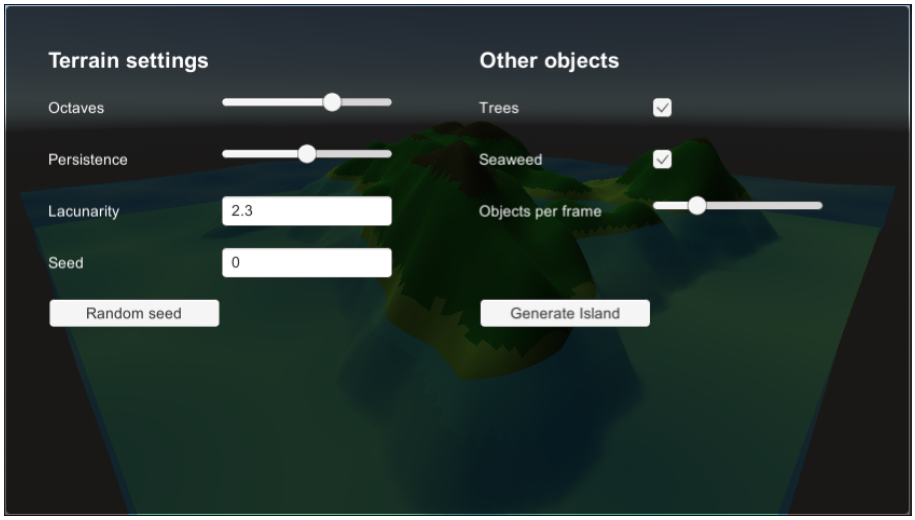
U glavnoj sceni povezani su objekti koji čine cjelinu programskog rješenja proceduralnog generiranja objekata za otočni krajolik. U scenu su dodani svi objekti iz prošlih poglavlja: objekti *Main Camera* i *Directional Light* služe svojoj standardnoj svrsi, objekt *Map Generator* upravlja objektom *Terrain Mesh*, objekt *Sea* se generira koristeći sjenčar vrhova i sjenčar fragmenata, objekt *Object Positioner* upravlja stvaranjem novog objekta *Generated Objects* čija su djeca ostali generirani objekti. *Post-processing Volume* upravlja učinkom u prostoru projekcije, a *Canvas*, *EventSystem* i *UI Manager* upravljaju grafičkim sučeljem za generiranje otoka, te će o njima biti više riječi u sljedećem potpoglavlju.



Slika 7.1: Hijerarhija objekata u glavnoj sceni

7.2. Korisničko grafičko sučelje

Korisničko grafičko sučelje u Unity 3D-u implementira se koristeći objekt *Canvas*. U djecu objekta *Canvas* dodan je prazan objekt imena *SettingsUI*, kojem je dodana komponenta *Horizontal Layer Group*, kako bi se njegova djeca prikazivala u jednom redu jedna za drugom. Ta djeca su objekt *Left* i *Right*. U objekt *Left* dodani su elementi grafičkog korisničkog sučelja za upravljanje parametrima za generiranje terena, a u objekt *Right* su dodani elementi za upravljanje generiranjem ostalih objekata, element za određivanje broja generiranih objekata po pozivu metode `Update` razreda `ObjectPositioner`, te gumb za početak generiranja otočnog krajolika.



Slika 7.2: Izgled korisničkog sučelja

U objektu *UI Manager* omogućena je veza između korisničkog sučelja i objekta *Map Generator* te su napisane potrebne *callback* funkcije za grafičke elemente, koje se pozivaju prilikom promjene na tim elementima. U metodi `Update` objekta *UI Manager* omogućuje se prikazivanje i skrivanje korisničkog sučelja pritiskom na tipku `Space`.

7.3. GitHub

Sav programski sadržaj ovog rada bit će objavljen na GitHub stranici autora, u repozitoriju imena *BSc-Thesis* [7][6].

7.4. WebGL

Da bismo omogućili isprobavanje programskog rješenja bez potrebe za preuzimanjem Unity 3D-a, napravit ćemo izvoz glavne scene u WebGL te objaviti taj *build* na GitHub pages stranici autora[5].

8. Zaključak

Kao što je navedeno u uvodu, cilj ovog rada bio je prikazati metode proceduralnog generiranja različitih objekata koji se mogu koristiti pri izradi scene otočnog krajolika.

Cilj generiranja terena ispunjen je koristeći Perlinov šum kao funkciju visine terena. Objašnjen je algoritam Perlinovog šuma, algoritam izrade plašta 3D objekta te metode za izradu teksture tog objekta. Predstavljene su metode kojima se može dodati još slojeva slučajnosti pri generiranju terena.

Još jedan cilj bio je generirati objekt morske površine i podmorja. Taj cilj ostvaren je koristeći sjenčar vrhova za generiranje oblika morkse površine, te sjenčar fragmenata temeljen na Voronoi dijagramu za generiranje boje morske površine. Izgled podmorja ostvaren je koristeći ugrađene razrede programskog pogona Unity 3D za učinke naknadne obrade.

Od ostalih objekata izrađeni su objekti stabala i morske trave koji se generiraju manipulirajući transformacijama 3D modela dizajniranih u programu Blender.

Upravljanje kamerom ostvareno je koristeći jednostavnu skriptu koja čita ulaz s tipkovnice te ovisno o unesenoj tipki upravlja rotacijom i pozicijom kamere.

Na samom kraju rada prikazana je hijerarhija objekata u glavnoj sceni, predstavljen je izgled i način rada grafičkog sučelja za upravljanje postavkama programa, te je objašnjeno gdje će biti objavljeni programski kod i sam program.

S obzirom da su glavni fokus ovog rada bile metode generiranja objekata, nije stavljen velik naglasak na performanse, te bi se program mogao poboljšati u tom kontekstu. Osim toga, daljnji rad na ovom projektu mogao bi uključivati izradu većih mapa korištenjem različitih razina detalja te rad na dodatnim proceduralno generiranim objektima.

LITERATURA

[1] Brackeys. SIMPLE CARTOON WATER in Unity, 2019. URL https://www.youtube.com/watch?v=Vg0L9aCRWPE.

[2] Joseph Kalathil. Voronoi Noise Shader, 2019. URL https://gist.github.com/josephbk117/a0e06d34aadb43777a1e35ccde508551.

[3] Sebastian Lague. Procedural Terrain Generation, 2019. URL https://www.youtube.com/playlist?list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3.

[4] Ken Perlin. An Image Synthesizer, 1985.

[5] Ivan Vlahov. GitHub pages, 2021. URL https://vlahovivan.github.io/.

[6] Ivan Vlahov. BSc-Thesis GitHub, 2021. URL https://github.com/vlahovivan/BSc-Thesis.

[7] Ivan Vlahov. GitHub, 2021. URL https://github.com/vlahovivan.

[8] Wikipedia. Perlin Noise, 2019. URL https://en.wikipedia.org/wiki/Perlin_noise.

[9] Wikipedia. No Man's Sky, 2021. URL https://en.wikipedia.org/wiki/No_Man\%27s_Sky.

[10] Wikipedia. Voronoi Diagram, 2021. URL https://en.wikipedia.org/wiki/Voronoi_diagram.

[11] Wiktionary. lacuna, 2021. URL https://en.wiktionary.org/wiki/lacuna.

Proceduralno generiranje objekata za otočni krajolik

Sažetak

U ovom radu prikazane su metode generiranja različitih objekata za prikazivanje otočnog krajolika. Objašnjen je algoritam Perlinovog šuma te metoda generiranja terena izradom visinske mape te funkcije. Uveden je koncept oktava koje omogućuju realističniji prikaz terena. Prikazan je rad sjenčara vrhova i sjenčara fragmenata za generiranje morske površine. Na primjeru stabala prikazano je generiranje složenih objekata koristeći instance postojećih 3D modela, a na primjeru morske trave prikazano je generiranje objekata mijenjajući koordinate vrhova postojećeg 3D modela. Objašnjen je rad skripte za upravljanje kretnjom kroz scenu te rad grafičkog sučelja. Krajnji proizvod ovog rada jest interaktivni program u kojem se mogu generirati otočni krajolici, a koji je javno objavljen na Internetu.

Ključne riječi: Proceduralno generiranje, generiranje terena, otočni krajolik, sjenčari

Procedural Generation of Objects for Island Landscape

Abstract

In this thesis several methods of procedural object generation are introduced and explained. Perlin's noise algorithm is described as well as the method of generating terrain using the function's height map. The concept of octaves is introduced to generate more realistic terrain. Vertex and fragment shaders are used to create sea surface. Method of generating complex objects using instances of premade 3D models is shown on the example of tree generation, and the method of generating objects by manipulating vertex coordinates of premade models is shown on the example of seaweed. The operation of the script which controls camera movement and the workings of graphical user interface are explained. The final product of this thesis is an interactive program in which island landscapes can be generated, which has been made public on the Internet.

Keywords: Procedural generation, terrain generation, island landscape, shaders