SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 863

PROCEDURALNO GENERIRANJE TERENA

Marko Prosenjak

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 863

PROCEDURALNO GENERIRANJE TERENA

Marko Prosenjak

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

ZAVRŠNI ZADATAK br. 863

Pristupnik:	**Marko Prosenjak (0036535883)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Proceduralno generiranje terena**

Opis zadatka:

Proučiti grafički programski pogon Unity. Posebice proučiti mogućnosti generiranja terena. Proučiti mogućnosti generiranja terena temeljem stvarnih podataka. Razraditi postupke izrade terena s proceduralno generiranom vegetacijom uz korištenje stvarnih visinskih mapa. Diskutirati utjecaj različitih parametara. Načiniti ocjenu rezultata i implementiranih algoritama. Izraditi odgovarajući programski proizvod. Koristiti grafičku programski pogon Unity. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 9. lipnja 2023.

SADRŽAJ

1. Uvod

Proceduralno generiranje je tehnika koja se često primjenjuje u računalnim igrama te animiranim filmovima. Najčešće se koristi za generiranje velikih površina, 3D objekata, te se također može koristiti za izradu animacija. Prednost ove tehnike je velika kontrola nad onime što se generira. U slučaju kada se koristi za izradu animacija, ovaj postupak omogućava kontrolu uz pomoć koje animacije izgledaju prirodnije te fluidnije, te je moguće postići veliku varijaciju između određenih animacija uz samo par promjena određenih parametara. Kada bi se za postizanje jednakog rezultata koristila ručna tehnika, bilo bi potrebno mnogo više vremena, zbog čega je ova tehnika toliko popularna. Također se koristi kod generiranja raznih površina, te rasporeda za instanciranje objekata na određenu površinu, na što se ovaj rad fokusira. Osim velike kontrole, prednost je ta što je ručno postavljanje objekata u 3D sceni teška i iscrpljujuća radnja koja iziskuje puno vremena, pogotovo ako je 3D scena velika. Osim velike količine vremena koja je potrebna za organizaciju scene, pojavljuje se i mogućnost ponavljanja određenih segmenata zbog nedovoljne kreativnosti, ili nedostatka ideja. Ti problemi dovode do gubljenja faktora uvjerljivosti, s obzirom na to da ništa u prirodi nije savršeno te je sve naizgled nasumično, odnosno u velikom broju slučajeva je teško odmah pronaći nekakav uzorak. Problem bi također bio kada bi objekti u prostoru bili previše nasumični, tj. kada između njih ne bi postojala nikakva veza. Iz tih razloga je proceduralno generiranje idealan odabir za ovakav tip problema.

2. Proceduralan šum

2.1. Općenito o šumu

Šum je nasumičan i nestrukturiran obrazac koji se koristi gdje god postoji potreba za izvorom opširnih detalja kojoj ipak nedostaje evidentna struktura [3]. Pridjev proceduralni koristi se u računalnoj znanosti za razlikovanje entiteta koji su opisani programskim kodom umjesto strukturama podataka. Proceduralne tehnike su segmenti koda ili algoritmi koji specificiraju neke karakteristike računalno generiranog modela ili efekta. U računalnoj grafici se šum odnosi na pseudoslučajne funkcije koje se koriste za izradu tekstura. Neke od poznatijih funkcija za izradu šuma su: Perlinov šum, Worleyev šum (eng. Worley noise), jednostavan šum (eng. Simplex noise) [8].

2.2. Općenito o generatoru pseudoslučajnih brojeva

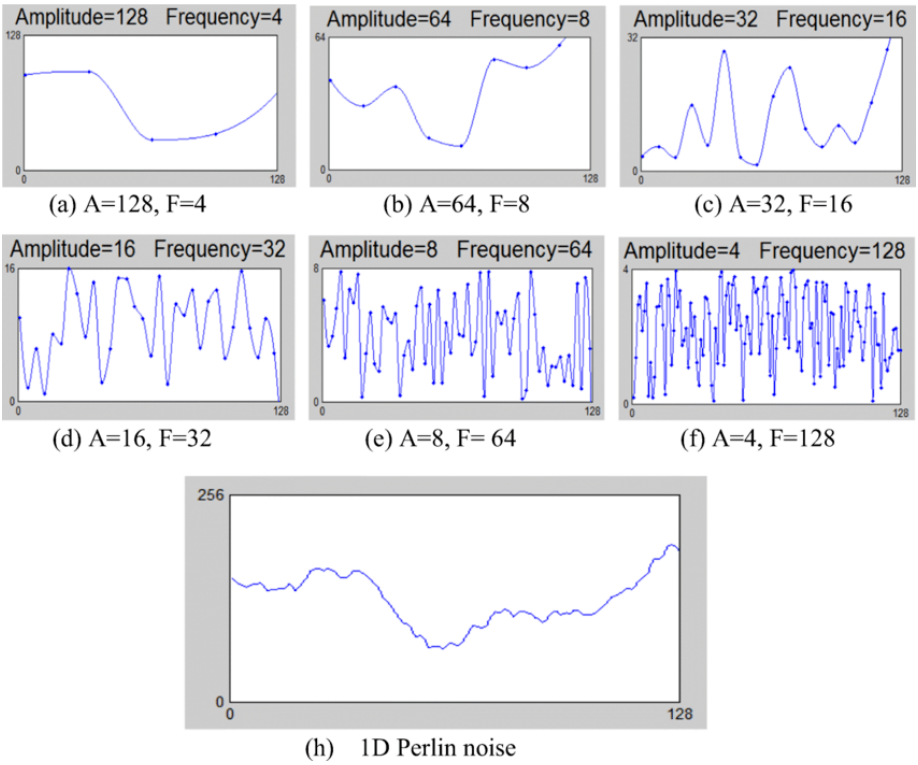
Generator pseudoslučajnih brojeva (PRNG) je algoritam koji služi za generiranje niza brojeva čija su svojstva slična svojstvima slučajnih brojeva. PRNG zapravo ne daje istinski slučajne rezultate, nego je određen inicijalnom vrijednošću koja se zove *seed* [4]. Generatori pseudoslučajnih brojeva se najčešće koriste u simulacijama, videoigrama i kriptografiji. Pseudoslučajan niz brojeva je onaj niz koji se čini statistički slučajnim, unatoč tome što je proizveden uz potpuno deterministički ponavljajući proces.

2.3. Perlinov šum

Perlinov šum je popularan algoritam proceduralnog generiranja koji je razvio Ken Perlin. Najčešće se koristi u videoigrama te u filmovima. Koristi se za generiranje tekstura, terena, oblaka, efekata vatre i sl. proceduralnim postupcima, bez potrebe da ih ručno rade dizajneri ili umjetnici. Ovaj algoritam najčešće se koristi kod generiranja objekata ili efekata kod kojih se želi zadržati organski izgled. Objekti u prirodi

nisu u potpunosti nasumičnih oblika, nego imaju određeni uzorak te puno detalja koje bi ručno bilo jako teško replicirati. Iz tog razloga je potrebno koristiti proceduralne algoritme koji su naizgled nasumični, no zapravo su deterministički. Objašnjenje u nastavku odnosi se na rad 2D Perlinovog šuma, no Perlinov šum može biti proizvoljnih dimenzija.

Ulazne vrijednosti funkcije nalaze se unutar mreže koja je sastavljena od kvadrata dimenzija 1x1, čije su koordinate vrhova cijeli brojevi. Donji lijevi vrh mreže ima koordinate (0,0), dok gornji desni vrh ima koordinate (n,n). Nakon što se ustanovi u kojem kvadratu se nalazi ulazna vrijednost (x,y), izračunaju se pseudoslučajni vektori gradijenta u vrhovima kvadrata unutar kojeg se ulazna vrijednost nalazi. S obzirom na to da je taj vektor pseudoslučajan, njegova vrijednost će naizgled biti slučajna, no zapravo nije, nego je stalna [1]. Za iste vrijednosti ulaza, odnosno koordinata vrhova, vektor gradijenta će uvijek imati istu vrijednost. Nakon što su izračunate vrijednosti vektora gradijenata u vrhovima, potrebno je izračunati vektore kojima je početak u vrhu kvadrata, a završetak u točki određenoj koordinatama ulazne vrijednosti. Ti vektori nazivaju se vektorima udaljenosti, te za razliku od vektora gradijenata nisu stalni, nego ovise o ulaznoj vrijednosti. Kako bi se izračunao konačan utjecaj ulazne vrijednosti, potrebno je izračunati skalarni umnožak vektora gradijenta te vektora udaljenosti u svakom vrhu kvadrata te potom interpolirati između dobivenih vrijednosti [2]. Za interpolaciju se koristi *krivulja ublažavanja (eng. "ease curve")* čija je formula: $6t^5 - 15t^4 + 10t^3$. U svrhu dodavanja detalja te postizanja prirodnijeg izgleda Perlinovog šuma, koriste se oktave. Iako se uz pomoć Perlinovog šuma može postići organski izgled kakav je moguće vidjeti u prirodi, uz ovakvo jednostavno korištenje funkcije nije moguće postići sve male nepravilnosti koje bi dodatno ojačale svojstvo uvjerljivosti. Iz tog razloga se funkcija Perlinovog šuma izvede više puta uz korištenje različitih frekvencija i amplituda, čije su inicijalne vrijednosti jednake 1. U svakoj oktavi se frekvencija poveća 2 puta, dok se amplituda množi s varijablom *postojanosti (eng. persistency)*, koja je obično realan broj između 0 i 2. Konačan rezultat je zbroj rezultata svih oktava, pri čemu svaka oktava ima sve manji utjecaj na konačan rezultat.

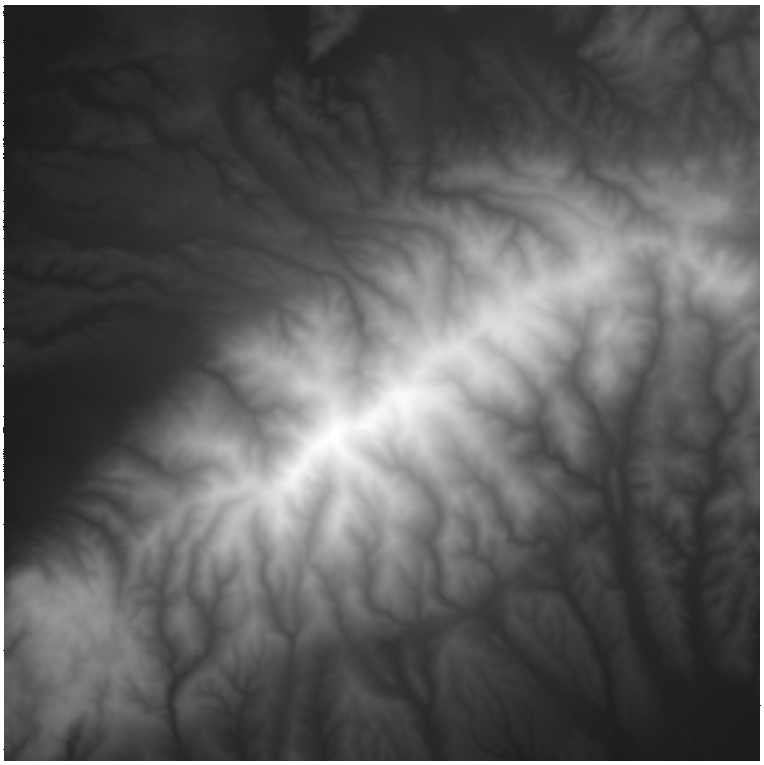


Slika 2.1: Zbroj oktava (izvor: `https://www.researchgate.net/publication/327042430/figure/fig8/AS:941729649803289@1601537253835/1D-Perlin-noise-noise-from-a-to-f-with-halved-amplitude-\and-doubled-frequency-to-the.png`)

3. Generiranje terena

3.1. Visinska mapa

Visinska mapa je tekstura koja se koristi za generiranje terena u sceni. Slika je crno-bijele boje, čime se označava visina terena na određenim segmentima. Svjetliji dijelovi su viši, dok su tamniji niži. Čista bijela boja označava vrh, odnosno najvišu točku terena, a čista crna boja označava najnižu točku terena, dok nijanse sive označavaju vrijednosti između najviše i najniže točke na terenu, ovisno o udjelu bijele i crne boje. Kako bi se generirao teren, potrebno je proći pojedinačno kroz svaki piksel te iz njega iščitati udio bijele i crne boje kako bi se odredilo koja je visina terena na mjestu koje odgovara poziciji piksela na teksturi. Slika visinske mape koja je korištena u zadatku, generirana je uz pomoć "Cities: Skylines map generator" (direktna poveznica: https://heightmap.skydark.pl/). Na stranici je bilo potrebno odabrati površinu čija bi se visinska mapa htjela preuzeti. Odabrano je područje Medvednice (Longitude: 15.96593°, Latitude: 45.89655°), nakon čega je bilo potrebno pritisnuti opciju "Download PNG height map", koja se nalazi s lijeve strane prozora, kako bi bilo pokrenuto preuzimanje visinske mape. Sliku je potom spremljena u novostvoreni direktorij "Textures" u Unity projektu.



Slika 3.1: Visinska mapa Medvednice (preuzeto sa stranice: `https://heightmap.skydark.pl/`)

3.2. Terrain tools paket

Za pomoć pri generiranju terena korišten je video s Youtube kanala Ketra Games [9]. Teren na kojem su instancirani objekti generiran je uz pomoć besplatnog paketa "Terrain Tools", preuzetog iz Unity-ovog upravitelja paketa. Nakon preuzimanja paketa, potrebno je kliknuti na padajući izbornik "Window", u kojem je ponuđena opcija "Terrain" -> "Terrain toolbox". U novootvorenom prozoru potrebno je unijeti parametre za generiranje terena. Parametri koji se trebaju unijeti su visina, širina te najviša točka terena. Oni su zapisani na stranici uz pomoć koje se generirala visinska mapa, te je njih potrebno prepisati. Radi jednostavnosti i poboljšanja performansi, svi parametri su umanjeni 100 puta. Na kraju je potrebno označiti opciju "Import Heightmap", odabrati visinsku mapu koja se nalazi u projektu, te kliknuti na gumb "CREATE" kako bi se generirao teren korištenjem visinskih mapa iz stvarnog svijeta.

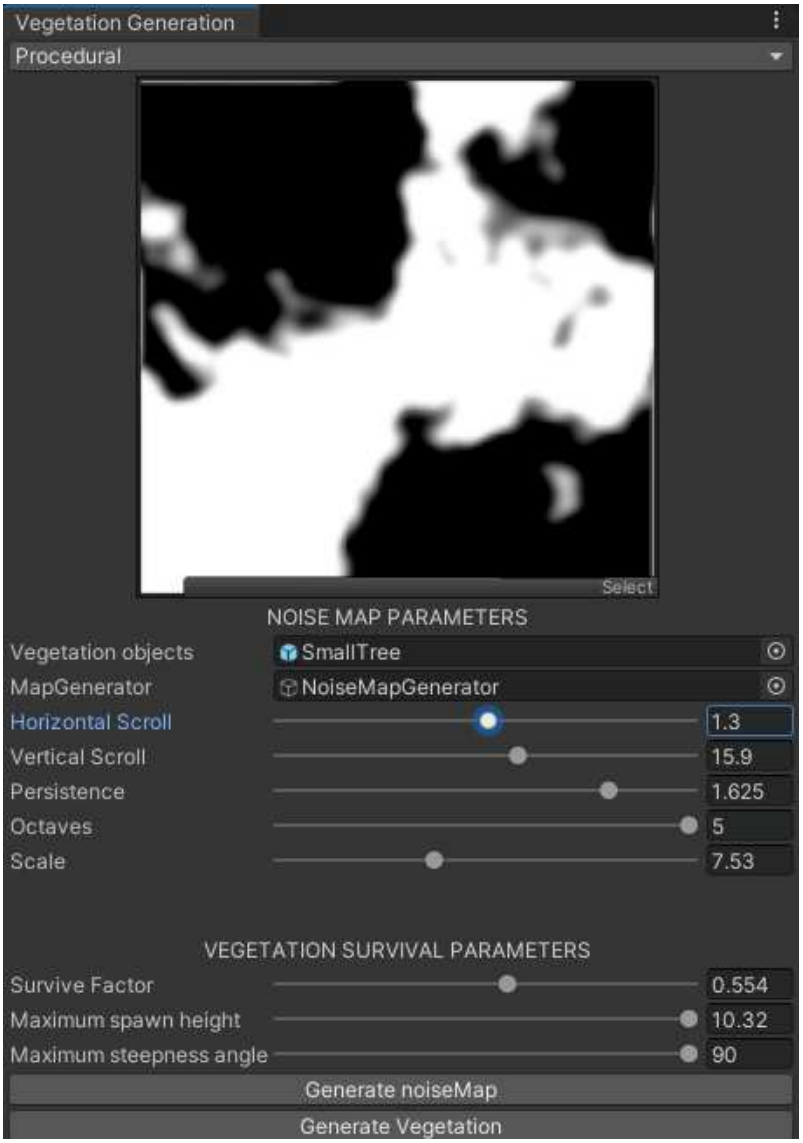
4. Izrada prozora

Svrha prozora je korištenje parametara za jednostavniju kontrolu prilikom generiranja vegetacije. S obzirom na to da su ljudi vizualna bića, puno je jednostavnije i prihvatljivije mijenjati parametre uz pomoć interaktivnog prozora koji koristi klizače i gumbe nego ručno mijenjati vrijednosti unutar koda. Kod za izradu prozora nalazi se u skripti *ProceduralGenerationWindow.cs*. Klasa *ProceduralGenerationWindow.cs* nasljeđuje *EditorWindow*, čime je naznačeno da se radi o elementu koji je tipa prozor. Ovime je omogućena izrada vlastitog interaktivnog prozora u Unity-u. Na početku skripte nalazi se oznaka

```
[MenuItem("Window/VegetationGenerator")]
```

čime je stvorena stavka izbornika, te je dodana u glavni izbornik pod opcijom "Window" [7]. *OnGUI* funkcija služi za crtanje *GUI* elemenata kao što su gumbi, labele i slično. Na vrhu prozora nalazi se padajući izbornik koji služi za odabir tehnike uz pomoć koje će se generirati vegetacija. Ispod padajućeg izbornika nalazi se tekstura. Kod tehnike korištenja satelitske slike, ona se odabire iz mape *Assets/Textures*, dok se kod tehnike korištenja Perlinovog šuma tekstura generira uz pomoć odgovarajućih parametara. Kod tehnike korištenja Perlinovog šuma, uz parametre za generiranje Perlinovog šuma postoje i parametri koji služe za donošenje odluke o tome koje pozicije su pogodne za instanciranje objekata. To su takozvani parametri preživljavanja. Kod tehnike korištenja satelitske slike, koriste se samo parametri preživljavanja. Svi parametri imaju svoje početne vrijednosti koje se mogu mijenjati uz pomoć klizača. Parametri također imaju rubne vrijednosti u oba smjera koje se ne mogu premašiti. Primjer takve rubne vrijednosti je maksimalan broj oktava koji je jednak 5. Razlog ograničavanju na 5 oktava su bolje performanse, te to što je s 5 oktava moguće ostvariti dovoljnu razinu detalja. Parametri koji su zajednički za obje tehnike generiranja su *Survive Factor, Maximum spawn height* i *Maximum steepness angle*. Parametar *Survive Factor* određuje minimalan intenzitet sive, tj. zelene boje koji piksel treba imati, *Maximum spawn height* određuje visinu iznad koje nije moguće instancirati objekte, *Maximum steep-*

ness angle određuje najveći nagib pod kojim objekti mogu biti instancirani. Nakon što je korisnik zadovoljan s odabranim parametrima, potrebno je pritisnuti na gumb *Generate Vegetation* (kod korištenja Perlinovog šuma, potrebno je prvo kliknuti na gumb *Generate noiseMap* kako bi se stvorila tekstura). Time se poziva funkcija *GenerateVegetation* te započinje postupak instanciranja. U slučaju da nije odabrana tekstura koja će se koristiti prilikom generiranja vegetacije, neće se moći kliknuti na gumb *Generate Vegetation* jer će biti onemogućen (eng. disabled).



Slika 4.1: Prozor za kontroliranje parametara pri generiranju vegetacije

5. Generiranje vegetacije na teren

5.1. Generalan princip instanciranja objekata na teren

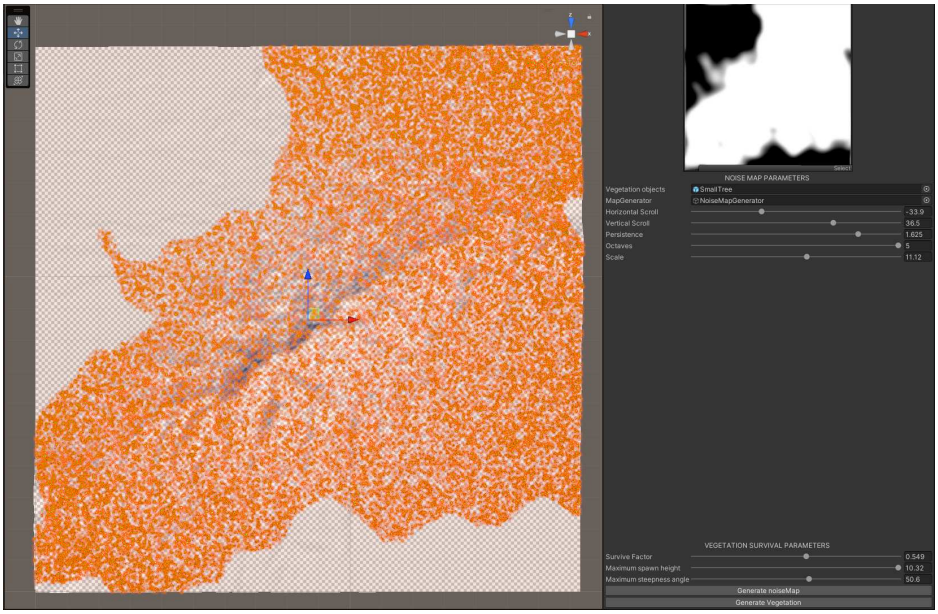
Kod za instanciranje vegetacije na teren nalazi se u skripti *ProceduralGenerationWindow.cs*, koja služi za stvaranje prozora spomenutog u prijašnjem poglavlju. Veći dio logike instanciranja nalazi se u funkciji *GenerateVegetation*, koja kao parametre prima teren na kojem će se vegetacija instancirati, teksturu koja određuje raspored vegetacije na terenu, te *bool* vrijednost koja daje informaciju o tome treba li se koristiti polje šumova ili ne. Ta *bool* vrijednost je potrebna kako bi bilo jasno koji postupak se koristi (Perlinov šum ili satelitska slika). Na početku funkcije odredi se *objectUpOffset* vrijednost koja označava visinu na kojoj se nalazi točka središta tijela (eng. "pivot point"). Ta vrijednost bit će potrebna u kasnijem koraku, te će njena svrha biti kasnije objašnjena. Također je potrebno napraviti novi *Transform* objekt koji će služiti kao roditelj svim instanciranim objektima. Nakon toga je potrebno proći po svim pikselima teksture koja se koristi za instanciranje objekata na teren. Za svaki piksel na teksturi se poziva funkcija *getParameters*, koja vraća polje elemenata tipa *float*. Polje sadrži informacije o visini i nagibu točke na terenu, koja odgovara poziciji promatranog piksela na teksturi, te informaciju o intenzitetu sive, odnosno zelene boje promatranog piksela. Boja čiji se intenzitet promatra, ovisi o tome koja tekstura se koristi prilikom instanciranja. Potom se provjeravaju uvjeti koji moraju biti zadovoljeni kako bi mogao započeti postupak instanciranja objekta na teren. Da bi uvjeti bili zadovoljeni, visina na kojoj bi objekt trebao biti instanciran (*height*), te nagib terena na području na kojem bi objekt trebao biti instanciran (*angle*), moraju biti manji ili jednaki zadanim vrijednostima *maxSpawnHeight*, odnosno *maxSteepness*. Vrijednosti varijabli *height* i *angle* dobivene su uz pomoć ugrađenih funkcija *GetInterpolatedHeight* i *GetSteepness*. Te funkcije kao argumente primaju normirane koordinate terena x i y (interval [0, 1]), koje predstavljaju poziciju točke na terenu u x0z ravnini, te vraćaju visinu, odnosno nagib terena u traženoj točki. Također je potrebno usporediti intenzitet boje piksela s varijablom *surviveFactor*. Ako je piksel zadovoljio sve uvjete, kreće postupak instan-

ciranja. U postupku instanciranja potrebno je napraviti vektor *position* tipa *Vector3*, koji predstavlja poziciju na terenu na kojoj će se instancirati objekt. Taj vektor kao x i z koordinate prima vrijednosti koje predstavljaju poziciju na terenu u x0z ravnini (duljina i širina), dok y koordinata predstavlja visinu terena na kojoj objekt treba biti instanciran. Vrijednosti visine terena na poziciji za instanciranje potrebno je pribrojiti vrijednost varijable *objectUpOffset*. Ako bi se kao y komponenta varijabli *position* predala samo visina terena na poziciji za instanciranje, pola visine objekta bi se nalazilo iznad, a pola ispod terena. Razlog tome je to što se prilikom instanciranja objekta, na poziciju, koja je predana kao argument funkciji *Instantiate*, postavlja središte objekta. U slučaju kada je y koordinata varijable *position* na koordinatama (x, y, z) jednaka visini terena na (x, z), središte objekta će biti postavljeno na površinu terena, te će teren na taj način sjeći objekt. Nakon instanciranja objekta na poziciji *position*, *MeshFilter* komponenta objekta se dodaje u *childrenMeshFilter* listu, te se instanciranom objektu pridjeljuje roditelj stvoren na početku funkcije.

```
for (int j = 0; j < textureHeight; j++)
{
   for (int i = 0; i < textureWidth; i++)
   {
      float[] parametersValues = getParameters(terrain,
         greenSurface, j, i, spawnTexture);
      float height = parametersValues[0];
      float pixelColorValue = parametersValues[1];
      float angle = parametersValues[2];
      bool condition = useNoiseMap ?
         (pixelColorValue >= (1 - surviveFactor) & height <=
            maxSpawnHeight & angle <= maxSteepness) :
         (pixelColorValue <= surviveFactor & height <=
            maxSpawnHeight & angle <= maxSteepness &
            greenSurface[i, j].spawnValue == 1);
      if (condition)
      {
         Vector3 position = new Vector3(i /
            (float)textureWidth * terrain.terrainData.size.x,
            height + objectUpOffset, j / (float)textureHeight
            * terrain.terrainData.size.z);
         GameObject plantToSpawn = Instantiate(objectToSpawn,
            position, Quaternion.identity);
```

```
            childrenMeshFilters.Add(plantToSpawn.GetComponent
            <MeshFilter>());
            plantToSpawn.transform.SetParent(parent);
        }
    }
}
```

Nakon što su instancirani svi objekti, počinje korak optimizacije, nakon kojeg dolazi do uništavanja roditelja, zajedno sa svom njegovom djecom.



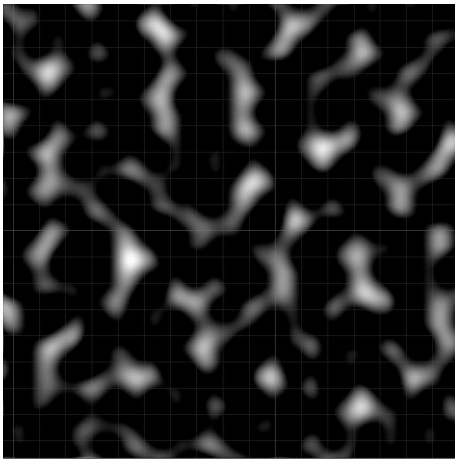
Slika 5.1: Generirana vegetacija na terenu

5.2. Korištenje Perlinovog šuma

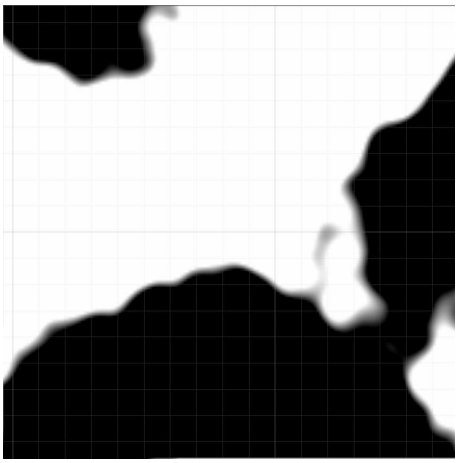
Dio koda napravljen je po uzoru na video Sebastiana Laguea [5]. U ovom postupku se izrađuje tekstura koja se koristi za generiranje pozicija za instanciranje objekata. Funkcija *makeNoiseMap* služi za stvaranje 2D polja u koje se spremaju izlazne vrijednosti funkcije Perlinovog šuma, te se to polje koristi za izradu teksture. U Unity3D postoji implementirana funkcija za generiranje Perlinovog šuma, stoga je potrebno samo odrediti vrijednosti koje će biti predane funckiji. Funkcija Perlinovog šuma prima parametar x i parametar y, koji predstavljaju x i y koordinatu uzorka. Bitno je naglasiti da su x i y varijable tipa float, na intervalu [0, 1], stoga ih je potrebno podijeliti sa širinom, odnosno duljinom terena. Kako bi se dobila veća varijacija i kontrola nad teksturom, dodani su parametri za skaliranje, vertikalan i horizontalan pomak, frekvenciju i amplitudu, koji se mogu mijenjati uz pomoć prozora za kontrolu parametara. Ti se parametri koriste kako bi se izračunale varijable *tempX* i *tempY*, koje se kao argumenti šalju funkciji Perlinovog šuma. Rezultat funkcije Perlinovog šuma množi se s amplitudom, te se rezultat umnoška pridodaje varijabli *totalPerlinNoise*, koja se nakon posljednje oktave sprema u 2D polje vrijednosti na poziciju (x, y). Nakon svake oktave frekvencija se uvećava dva puta, a amplituda se množi s varijablom *persistence*.

```
tempX = (x - widthCenter) / (width * scale * frequency) *
   100 + horizScroll;
tempY = (y - heightCenter) / (height * scale * frequency)
   * 100 + vertScroll;
perlinValue = Mathf.PerlinNoise(tempX, tempY) * 2 - 1;
totalPerlinNoiseValue += perlinValue * amplitude;

frequency *= 2f;
amplitude *= persistence;
```

Dobivena lista koristi se u funkciji *drawNoiseMap*, koja se nalazi u skripti *DrawMap.cs*, za izradu teksture. Funkcija iterira po svim pikselima novostvorene teksture te dohvaća vrijednost Perlinovog šuma iz polja *noiseMap*, s pozicije koja odgovara poziciji piksela na teksturi. Dobivena vrijednost se koristi za interpoliranje između bijele i crne boje uz pomoć ugrađene funkcije *Color.Lerp*. Rezultat se sprema u listu boja. Nakon što su petlje završile sa svim prolazima, lista boja se primjenjuje na piksele teksture. Ta se tekstura pridjeljuje komponenti *Renderer* objekta *PlaneTexturedMap* kao njegova glavna tekstura, kako bi rezultat bio vidljiv u scenskom prikazu.



Slika 5.2: Prikaz prve oktave

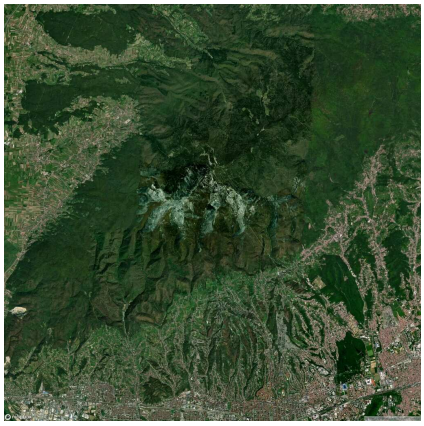


Slika 5.3: Prikaz pete oktave

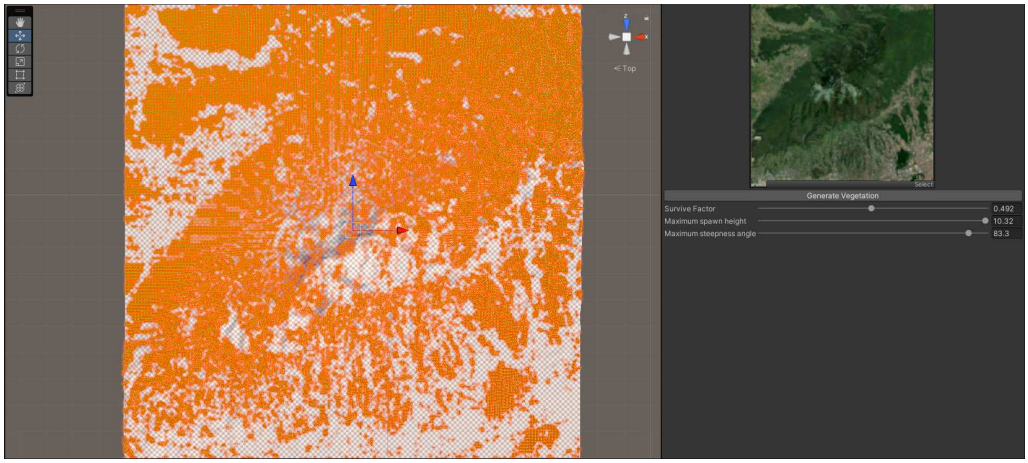
5.3. Korištenje satelitske slike

Za provedbu ovog algoritma potrebna je satelitska slika područja na temelju koje bi se htjela generirati vegetacija. Ideja je generirati vegetaciju na onim dijelovima terena koji odgovaraju zelenim pikselima na satelitskoj slici. Sa stranice `https://heightmap.skydark.pl/` je preuzeta satelitska slika područja Medvednice u boji, te je spremljena u projekt u mapu "Textures". Radi postizanja što boljih performansi, uz gubitak što manje detalja, korištena rezolucija slike je 1k. Ta tekstura bila je korištena za određivanje pozicija na terenu na kojima će se generirati vegetacija. Prvi korak u generiranju vegetacije ovim principom je proći kroz sve piksele teksture, te za svaki piksel provjeriti radi li se o pikselu zelene boje. Za iteriranje kroz sve piksele teksture korištena je funkcija *scanSatelliteImage*. Za svaki piksel teksture se poziva funkcija *greenColorChecker*, kako bi se utvrdilo je li promatrani piksel zelene boje.

Funkcija *greenColorChecker* prima boju piksela u RGB formatu i pretvara ju u HSV format. Razlog korištenja HSV formata je taj što su na njemu jasno definirani intervali, odnosno kutevi unutar kojih određena boja pripada, dok kod RGB formata nije jednostavno definirati koju kombinaciju crvene, zelene i plave komponente treba koristiti kako bi se dobile sve nijanse zelene boje. Ako je boja zelena, funkcija vraća vrijednost svjetline piksela. Vrijednost se predaje konstruktoru strukture "pixelInfo" te se novostvoreni objekt sprema u 2D polje. Funkcija *GenerateVegetation* koristi podatke iz 2D polja *greenSurface* kako bi se oktrio intenzitet zelene boje promatranog piksela. To je jedan od uvjeta koji treba biti zadovoljen kako bi se započelo s procesom instanciranja objekata.



Slika 5.4: Satelitska slika Medvednice (preuzeto sa stranice: `https://heightmap.skydark.pl/`)



Slika 5.5: Generirana vegetacija uz pomoć satelitske slike

6. Optimizacija programa

6.1. Osnovni pojmovi

U računalnoj grafici poziv crtanja (eng. "draw call") je naredba koja se šalje GPU-u kako bi se prikazao ili nacrtao određeni objekt ili skup objekata na zaslonu. Svaka naredba crtanja sadrži informaciju koju grafički API treba kako bi iscrtao objekt na zaslon. U pripremi za naredbu crtanja, CPU postavlja resurse te mijenja unutarnje postavke na GPU-u. Te postavke se zovu stanje prikaza (eng. "render state"). S obzirom na to da su promjene stanja prikaza resursno zahtjevne operacije, potrebno ih je optimizirati. Glavni način za optimizaciju je smanjenje njihovog broja [6]. Postoje dva načina kako bi se to moglo izvesti. Prvi je smanjenje ukupnog broja poziva crtanja. Kada se smanji broj poziva crtanja, također se smanjuje broj promjena stanja prikaza između njih. Drugi način je organizacija poziva crtanja tako da se smanjuje broj promjena stanja prikaza. Ako grafički API može koristiti isto stanje prikaza kako bi se obavilo više poziva crtanja, onda može grupirati pozive crtanja. U Unity-u postoji nekoliko metoda za optimizaciju i smanjenje broja poziva crtanja i promjena stanja iscrtavanja. Metoda korištena u ovom radu je metoda spajanja objekata u jedan objekt ("combining meshes") uz pomoć *Mesh.CombineMeshes* funkcije. Ovime je postignuto da Unity za iscrtavanje kombiniranog objekta koristi ukupno jedan poziv iscrtavanja umjesto jednog poziva iscrtavanja po objektu. Pojam koji je također bitan za ovaj rad je paketna obrada (eng. batch). Ovaj pojam se odnosi na postupak grupiranja i izvršavanja većeg broja poziva crtanja kako bi se poboljšale performanse.

6.2. Opis problema i rješenje

U slučaju kada je na teren bilo instancirano mnogo objekata, izvršavanjem operacija rotacije pogleda, te pomicanja objekata u sceni, pojavio se problem kašnjenja slike. Taj problem je otežavao rad u sceni, te je bio razlog zbog kojeg su implementirana

rješenja sa svrhom optimizacije programa. Tehnika koja je korištena u optimizaciji je već spomenuto kombiniranje objekata u jedan objekt. Ovo rješenje je prikladno s obzirom na to da su instancirani objekti u sceni statični, odnosno ne pomiču se u sceni. Kako bi se objekti mogli kombinirati, potrebna je imati informaciju o njihovim *MeshFilter* komponentama. Radi toga je napravljena lista MeshFilter komponenti, koja se zove *childrenMeshFilters*. MeshFilter komponenta svakog objekta koji se instancira sprema se u tu listu. Nakon što je *for petlja* prošla po svim pikselima teksture, poziva se funkcija *combineMeshFilters* kojoj se kao argument predaje roditelj *GameObject*. U funkciji je stvorena lista tipa *CombineInstance* koja ima istu duljinu kao lista *childrenMeshFilters*. Potom je potrebno iterirati po elementima liste *childrenMeshFilters* te spremiti mesh i transform podatke pojedinog elementa u listu *combineInstances*. Za elemente liste *combineInsances* koristi se sharedMesh od elemenata *childrenMeshFilters* liste kako bi materijal bio vidljiv u sceni. Potom je potrebno stvoriti novu *Mesh* komponentu koja će biti kombinirana *Mesh* komponenta svih instanciranih objekata. Potrebno je postaviti indexFormat *combinedMesh* komponente na *UnityEngine.Rendering.IndexFormat.UInt32* jer je ukupan broj točaka objekata koji se spajaju veći od 65536. Nakon što je postavljen odgovarajući indexFormat, napokon se može izvesti operacija spajanja svih *Mesh* komponenata iz liste *combineInstances* u jedan *Mesh*. Potom je potrebno stvoriti novi objekt tipa *GameObject* kojem će se pridijeliti kombinirana *Mesh* komponenta. S obzirom na to da svi objekti sadrže isti materijal, kombiniranom objektu dodijeljen je materijal koji se nalazi na objektu koji se koristi za instanciranje. Nakon povratka iz funkcije, uništava se *GameObject parent*, koji je roditelj svih instanciranih objekata, zbog čega se posljedično i sva djeca unište te ostaje samo kombinirani *GameObject*.

```
private void combineMeshFilters(GameObject parentObject)
{
    CombineInstance[] combineInstances = new
        CombineInstance[childrenMeshFilters.Count];

    for (int i = 0; i < childrenMeshFilters.Count; i++)
    {
        combineInstances[i].mesh =
            childrenMeshFilters[i].sharedMesh;
        combineInstances[i].transform =
            childrenMeshFilters[i].transform.localToWorldMatrix;
    }
```

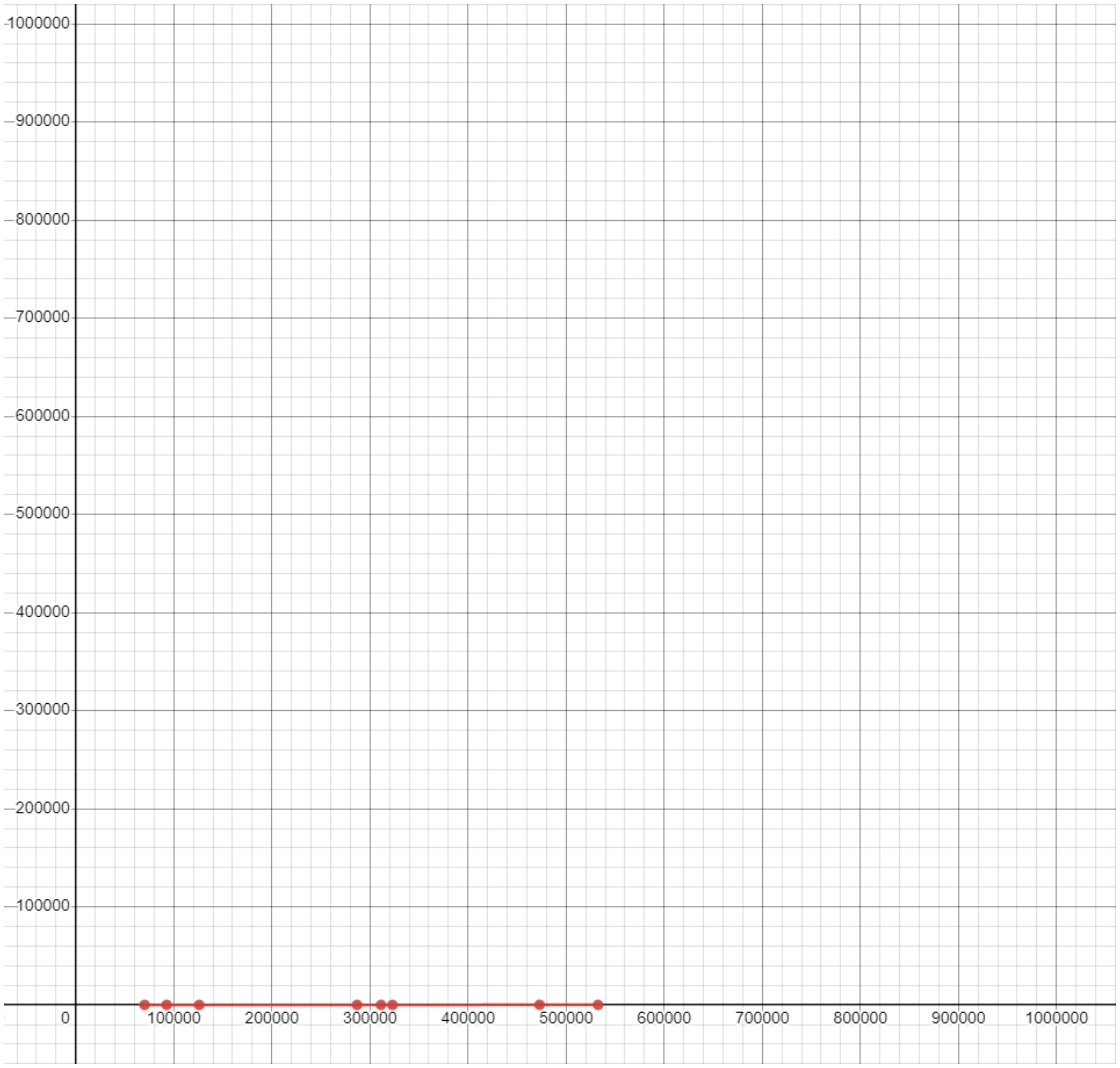
```
        Mesh combinedMesh = new Mesh();
        combinedMesh.indexFormat =
            UnityEngine.Rendering.IndexFormat.UInt32;
        combinedMesh.CombineMeshes(combineInstances, true,
            true);

        GameObject combinedObject = new GameObject("Combined
            Vegetation");
        MeshFilter meshFilter =
            combinedObject.AddComponent<MeshFilter>();
        meshFilter.mesh = combinedMesh;
        combinedObject.AddComponent<MeshRenderer>().sharedMaterial
            =
            objectToSpawn.GetComponent<MeshRenderer>().sharedMaterial;
    }
```

6.3. Rezultati optimizacije

Provedeno je nekoliko mjerenja i usporedbi broja paketnih obrada uz i bez korištenja optimizacije. Za generiranje 532540 objekata bez korištenja optimizacije, korišteno je 827612 paketnih obrada, dok je za generiranje jednakog broja objekata uz korištenje optimizacije korišteno samo 96 paketnih obrada. Očito se radi o velikoj razlici (čak 8620.95 puta manje paketnih obrada pri korištenju optimizacijskog postupka). Za slučaj kada je u sceni generirano 125980 objekata, ako se ne koristi postupak optimizacije, korišteno je 349686 paketnih obrada, dok ih je uz pomoć optimizacije korišteno 46 (7601.86 puta manje). Postupak uništavanja objekata također je ubrzan, jer je za uništavanje jednog po jednog objekta putem petlje potrebno više vremena nego što je potrebno za uništavanje jednog kombiniranog objekta. Na slici 6.2 prikazani su rezultati mjerenja broja paketnih obrada u odnosu na broj instanciranih objekata uz korištenje optimizacije. Broj paketnih obrada (vrijednosti po osi ordinata) kreće se između 45 i 96, dok se broj instanciranih objekata (vrijednosti po osi apscisa) kreće između 70220 i 532540. Kako bi bile vidljive sve točke krivulje, bilo je potrebno umanjiti prikaz (eng. zoom out), zbog čega se dobiva dojam da je dobivena krivulja kojoj je y koordinata u svakoj točki jednaka 0. Grafovi su crtani u alatu Desmos (poveznica: `https://www.desmos.com/calculator`). Konfiguracija računala na kojem

su mjereni rezultati: AMD Ryzen 5 5500, 16GB RAM (3600MHz), RTX 2060 6GB.

Slika 6.1: Rezultati bez optimizacije (os apscisa: broj instanciranih objekata, os ordinata: broj paketnih obrada



Slika 6.2: Rezultati s optimizacijom (os apscisa: broj instanciranih objekata, os ordinata: broj paketnih obrada (vrijednosti između 45 i 96))

7. Zaključak

Cilj rada bio je proceduralno generirati vegetaciju na teren stvoren uz pomoć visinskih mapa iz stvarnog svijeta. Proceduralno generiranje je ostvareno uz pomoć korištenja vrijednosti Perlinovog šuma te uz pomoć korištenja satelitske slike područja iz stvarnog svijeta. Prije generiranja vegetacije potrebno je ubaciti satelitsku sliku u prostor za teksturu, te objekt *NoiseMapGenerator* u polje *MapGenerator* te objekt *SmallTree* u polje *VegetationObjects*.

Najprije je potrebno preuzeti visinsku mapu područja od kojeg se želi napraviti teren s nekog izvora na internetu. Potom je potrebno preuzeti paket *Terrain Tools*, te je uz pomoć paketa i visinske mape generirate teren na koji će se instancirati objekti.

Napravljen je interaktivni prozor kako bi korisniku bila omogućena opcija jednostavnog i intuitivnog mijenjanja parametara uz pomoć kojih bi se generirali različiti rezultati. Moguće je koristiti unaprijed zadane parametre pri generiranju, no kako bi se postigla što veća varijacija, korisniku je omogućeno ručno podešavanje tih parametara uz pomoć klizača koji se nalaze u prozoru. Nakon što su zadani željeni parametri, potrebno je kliknuti na gumb "Generate Vegetation", odnosno najprije na gumb "Generate noiseMap", pa zatim na gumb "Generate Vegetation", ako se koristi tehnika Perlinovog šuma. Nakon toga će objekti biti instancirani na površinu terena.

Program je optimiziran kako bi radio što je moguće brže, čime je poboljšano korisničko iskustvo. Optimizacija je omogućila jednostavnije kretanje u sustavu scene. Daljnji rad mogao bi uključivati veću varijaciju objekata koje je moguće instancirati, primjenjivanje teksture na teren, dodatnu optimizaciju, proceduralno generiranje puteva na terenu i slično. S obzirom na to da je vegetacija trenutno predstavljena cilindrima, mogli bi se koristiti modeli drveća različitih dimenzija, modeli trava, model grmlja i drugo. Cjelokupni projekt nalazi se na poveznici: `https://github.com/NEYMARKO/Zavrsni-Rad`.

LITERATURA

[1] Raouf Touti. *Perlin Noise: A Procedural Generation Algorithm, 2018*. https://rtouti.github.io/graphics/perlin-noise-algorithm, pristupljeno: lipanj 2023.

[2] Adrian Biagioli. *Understanding Perlin Noise, 2014.*. https://adrianb.io/2014/08/09/perlinnoise.html, prisupljeno: lipanj 2023.

[3] A. Lagae. *A Survey of Procedural Noise Functions, 2010*. https://core.ac.uk/download/pdf/34480918.pdf, pristupljeno: lipanj 2023.

[4] Wikipedia contributors. *Pseudorandom number generator* https://en.wikipedia.org/wiki/Pseudorandom_number_generator, pristupljeno: lipanj 2023.

[5] Sebastian Lague. *Procedural Terrain Generation, 2016*. https://www.youtube.com/playlist?list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3, pristupljeno: travanj 2023.

[6] Unity developers. *Unity Manual* https://docs.unity3d.com/Manual/index.html, pristupljeno: lipanj 2023.

[7] Brackeys. *How to make an EDITOR WINDOW in Unity, 2018*. https://www.youtube.com/watch?v=491TSNwXTIg, pristupljeno: svibanj 2023.

[8] Wikipedia contributors. *Noise (spectral phenomenon), 2022*. https://en.wikipedia.org/wiki/Noise_(spectral_phenomenon)#Noise_in_computer_graphics, pristupljeno: lipanj 2023.

[9] Ketra Games. *Create Terrain from a Heightmap (Unity Tutorial), 2022*. https://www.youtube.com/watch?v=bFGXghavsD8, pristupljeno: ožujak 2023.

Proceduralno generiranje terena

Sažetak

Ovaj rad bavi se proceduralnim generiranjem vegetacije na površinu terena koji je generiran uz pomoć visinskih mapa iz stvarnog svijeta. Za generiranje vegetacije koriste se algoritam Perlinovog šuma te algoritam koji koristi satelitsku sliku iz stvarnog svijeta. Napravljen je interaktivni prozor radi jednostavnije kontrole parametara korištenih za generiranje vegetacije. Programski kod je optimiziran kako bi se poboljšale performanse programa. Napravljena je usporedba rada programskog rješenja uz korištenje optimizacije i bez korištenja optimizacije. Programsko rješenje napravljeno je u grafičkom programskom pogonu Unity, korištenjem programskog jezika C#.

Ključne riječi: proceduralno generiranje, Perlinov šum, Unity, visinska mapa, prozor uređivača, teren

Procedural terrain generation

Abstract

This paper deals with the procedural generation of vegetation on the surface of the terrain, which is generated with the help of height maps from the real world. The Perlin noise algorithm and an algorithm that uses a real-world satellite image are used to generate vegetation. An interactive window was created for easier control of the parameters used to generate vegetation. The program code is optimized to improve program performance. A comparison was made between performance of the software solution with and without optimization. The program solution was created in the graphical programming engine Unity, using the programming language C#.

Keywords: procedural generation, Perlin noise, Unity, heightmap, editor window, terrain