SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2878

PROCEDURALNO GENERIRANJE ŠPILJSKOG SUSTAVA

Danijel Bajlo

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2878

PROCEDURALNO GENERIRANJE ŠPILJSKOG SUSTAVA

Danijel Bajlo

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

DIPLOMSKI ZADATAK br. 2878

Pristupnik:	**Danijel Bajlo (0036505511)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Proceduralno generiranje špiljskog sustava**

Opis zadatka:

Proučiti osnovne gradivne elemente u špiljama i podzemnim prostorima. Proučiti mogućnosti proceduralnog generiranja takvih prostora. Razraditi generiranje neograničenih prostora. Programski implementirati rješenje za proceduralno generiranje podzemnih neograničenih sustava špilja. Načiniti testiranje na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Koristiti programski alat Unity i programski jezik C#. 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: 27. lipnja 2022.

Hvala mentorici prof. dr. sc. Željki Mihajlović na brzim odgovorima i kontinuiranoj podršci. Hvala mom prijatelju Ivanu na pomoći sa snimanjem demonstrativnog videa.

Sadržaj

Uvod

Proceduralno generiranje daje beskonačno mnogo rezultata i raznovrsnosti, bez zauzimanja puno memorije. Najveći izazovi proceduralnog generiranja su stvaranje interesantnog sadržaja, jer se generirani sadržaj često ponavlja s neprimjetno malim varijacijama. Također, generiranje velike količine sadržaja za vrijeme igranja predstavlja veliko računsko opterećenje pa je znatan izazov omogućiti korisniku glatko iskustvo bez prekida i naglih skokova. Ovaj rad neće se baviti proceduralnim generiranjem jedne scene, već će naglasak biti na dinamičkom generiranju neograničeno velikog terena.

Špilje su oduvijek interesirale i privlačile ljude. Istraživanje pećina je hobi mnogima, a neke od njih su i turističke atrakcije. Špilje nastaju na različite načine pa imaju raznovrsne teksture i oblike koje je izazov reproducirati. Istraživanjem i proučavanjem špilja bavi se znanost speleologija.

U ovom radu najprije ću istražiti različite vrste i karakteristike špiljskih sustava, kako bih odredio specifikacije toga što će biti generirano. Zatim ću prikazati nekoliko različitih tehnika za generiranje špilja. U radu biti će korištena tehnika Perlinovog crva u kombinaciji s algoritmom pokretne kocke, kako bi se teren vizualizirao.

Istražiti ću i grafove sjenčara koje nudi alat Unity te objasniti stvaranje grafa triplanarnog sjenčara. Također ću prikazati tehniku miješanja više tekstura i njihovih normala kako bi se smanjila monotonija i uklonilo ponavljanje tekstura.

Objasniti ću proces dinamičkog generiranja komada terena te metode kojima će se generirati špilje koje prolaze kroz veći broj komada.

Na kraju ću se baviti podešavanjem parametara kako bi teren i špilje izgledao što ljepši. Također ću opisati tehnike optimizacije kojima se generiranje može ubrzati.

1. Svojstva špiljskih sustava

Prije nego se započne s proceduralnim generiranjem, potrebno je definirati specifičnosti i ograničenja onoga što se generira te na temelju tih informacija odabrati odgovarajući pristup.

Špilja je prirodna šupljina u zemlji dovoljno velika da u nju uđe čovjek. Iako se otvori čija je širina manja nego dubina ponekad nazivaju špiljama, oni spadaju u endogene špilje. Takvi oblici lako se mogu generirati trodimenzionalnim šumom pa fokus ovog rada nije na njima, već egzogenim špiljama čija dubina je veća od širine otvora.

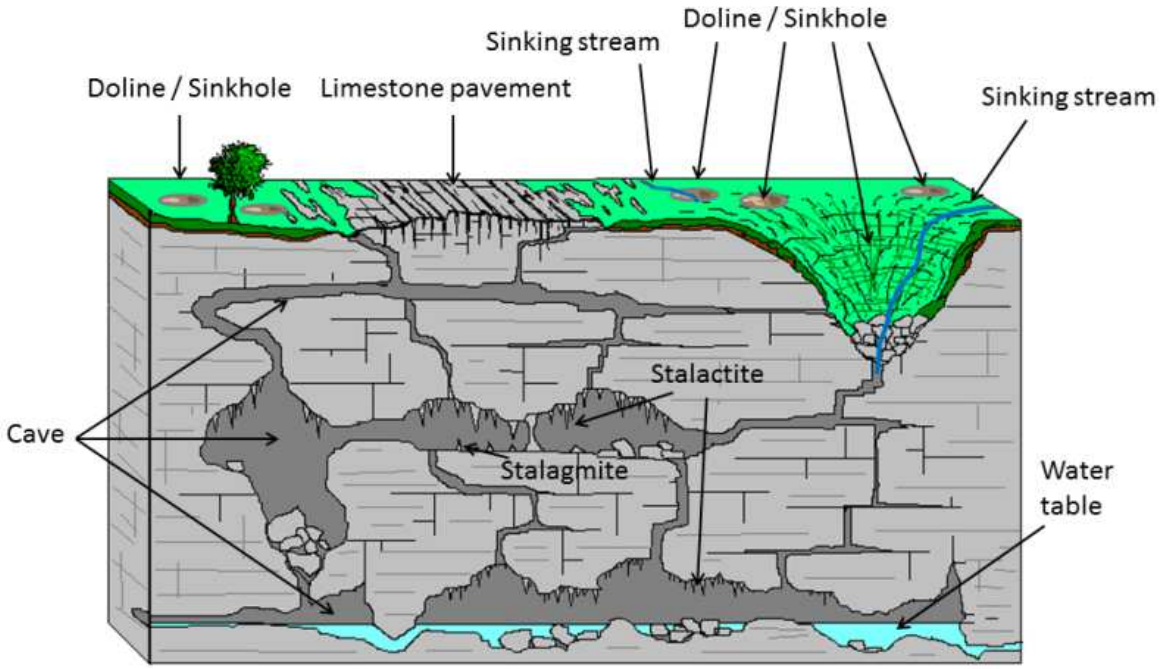
1.1. Nastanak špilja

Postoji više tipova špilja na temelju nastanka. Osnovna podjela je na primarne, koje su nastale u isto vrijeme kao i okolne stijene, te sekundarne, koje su nastale nakon što je stijena već formirana. Primarne špilje većinom nastaju oko vulkana za vrijeme hlađenja toka lave. Formiranje stalaktita u jednoj takvoj pećini prikazuje Slika 1.1.

Slika 1.1 Pećina koja nastaje u lavi [1]

Za nastanak sekundarnih špilja potrebna je neka vanjska sila koja erodira ili otapa stijenu. Stoga postoje obalne špilje koje nastaju erozijom valova, tektonske špilje, pa čak i eolske

špilje koje stvara utjecaj vjetra. Neke špilje ni ne moraju nastati u stijeni. Na primjer, ledene špilje nastaju otapanjem ledenjaka. Ipak, najčešći tip špilje na svijetu je krški, koji je dobio ime po krškom reljefu u kojem nastaje. Stvara se utjecajem vode i organskih kiselina koji otapaju slojeve topljivih stijena poput vapnenca i dolomita. Trošenjem stijene kroz vrijeme, malene pukotine prerastaju u špilje i špiljske sustave. Karakteristike krškog reljefa prikazuje Slika 1.2. Gornja granica podzemne vode (engl. *Water table*) predstavlja razinu ispod koje će svi dijelovi špilje biti zasićeni vodom (engl. *Saturated*) odnosno potopljeni.



Slika 1.2 Karakteristike krškog reljefa [2]

Ovaj rad bavit će se generiranjem špilja po uzoru na krški tip.

1.2. Veličina špilja

Zbog nepredvidivog procesa nastanka koji može trajati milijunima godina, špilje imaju izuzetno veliku raznolikost oblika i veličina. Upravo to ih mnogim ljudima čini uzbudljivim za istraživanje. Volumenom najveća špilja je Hang Sơn Đoòng u Vijetnamu koja je na najvećem dijelu čak 200 metara visoka i 150 metara široka, a prikazuje ju Slika 1.3. Navedena špilja nastala je otapanjem vapnenca prije 2 do 5 milijuna godina, a u njoj se nalaze divovski stalagmiti, od kojih su najveći čak 70 metara visoki. [3]

Slika 1.3 Volumenom najveća pećina na svijetu [3]

Najdulja pećina na svijetu je američki nacionalni park Mammoth cave s duljinom od 675.9 kilometara. [20]

Jame također spadaju u špilje pa se može mjeriti i njihova dubina. Tako je najdublja pećina Veryovkina u Gruziji s dubinom od 2212 metara. Valja spomenuti i da se u Hrvatskoj nalazi jedna od najdubljih pećina na svijetu, a to je Lukina jama, koja seže u dubinu 1431 metar. [21]

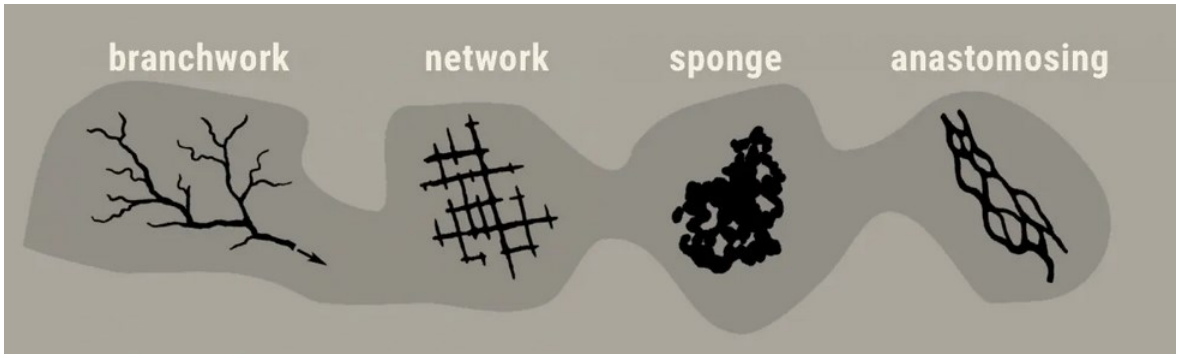
Iz ovih podataka vidljivo je da špilje mogu biti ekstremnih dimenzije. Zbog ograničenja veličine terena te iz praktičnosti, u ovom radu ću se baviti generiranjem špilja duljine u desecima ili stotinu metara te širine od nekoliko metara.

1.3. Oblici špilja

U osnovi postoje 4 oblika špiljskih sustava koji su određeni načinom i uvjetima pod kojima je špilja nastala. [4] Slika 1.4 prikazuje navedene oblike.

1. Razgranate špilje (engl. Branchwork) nastaju tokom vode te su najučestaliji oblik koji pronalazimo
2. Mrežaste špilje (engl. Network) nastaju kemijskom erozijom pukotina
3. Spužvaste špilje (engl. Sponge) nastaju podizanjem gornje granice podzemne vode te kemijskim otapanjem

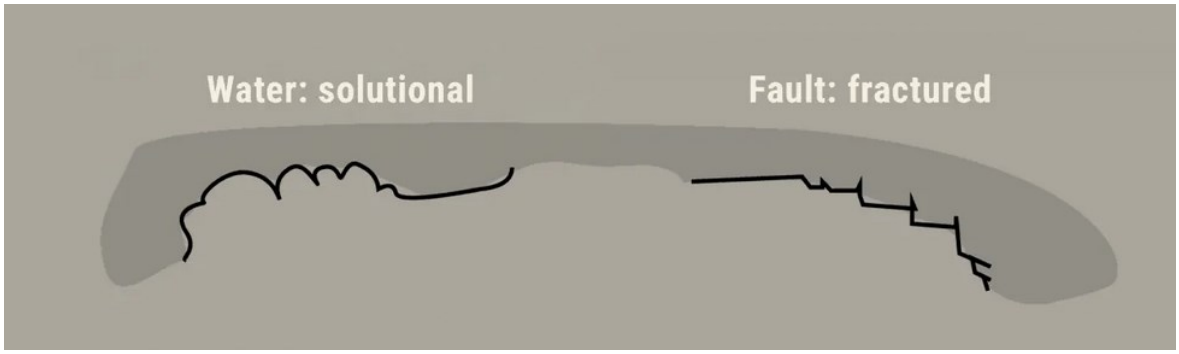
4. Anastomozne špilje (engl. Anastomosing) nastaju unutar jednog sloja te su pretežno horizontalne, a nalikuju pletenicama



Slika 1.4 Oblici špiljskih sustava[4]

S obzirom da su razgranate pećine najčešće, u ovom radu ću se fokusirati na generiranje takvog oblika sustava.

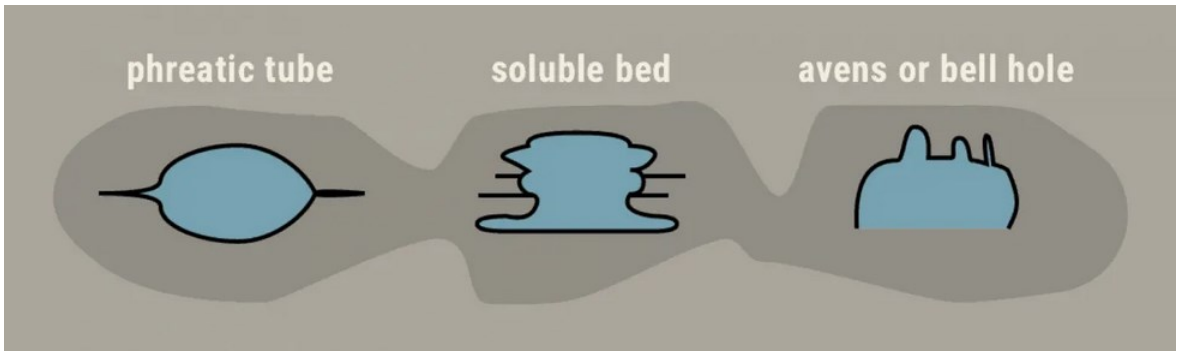
Površine unutar pećine biti će glatke ako je uzrok nastanka voda, a grube i pravokutne ako im je uzrok nastanka lomljenje pod pritiskom. Usporedbu prikazuje Slika 1.5. Ovaj radi bavit će se generiranjem pećina kojima je uzrok nastanka voda pa će površine biti zaobljene i glatke.



Slika 1.5 Oblici površine unutar pećine[4]

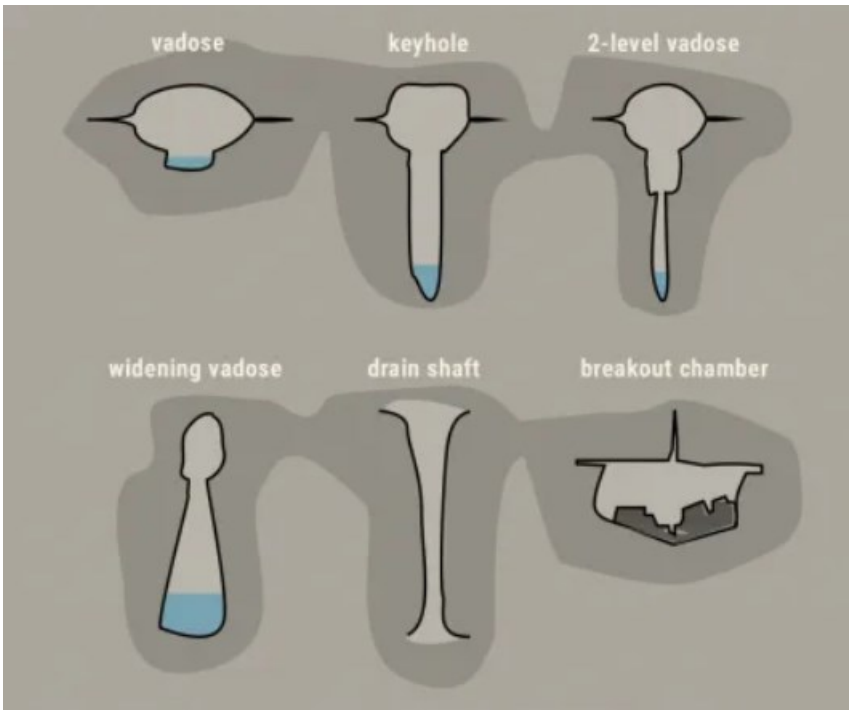
Čak i među pećinama koje su nastale zbog istog uzroka, postoji mnogo različitih oblika poprečnog presjeka. Tako kod špilja uzrokovanih protokom vode postoje freatični i vadozni prolazi.

Freatični prolazi nastaju kada je prolaz u potpunosti ispod razine vode te njime teče voda pod pritiskom. Zbog pritiska, voda će podjednako otapati pod, plafon i zidove prolaza te će njegov oblik biti pretežno eliptičan. Izrazito okrugli prolazi ukazuju na brzi tok vode. Što je tok sporiji, to će prolaz više zadržati izvorni oblik pukotine iz kojeg je nastao.



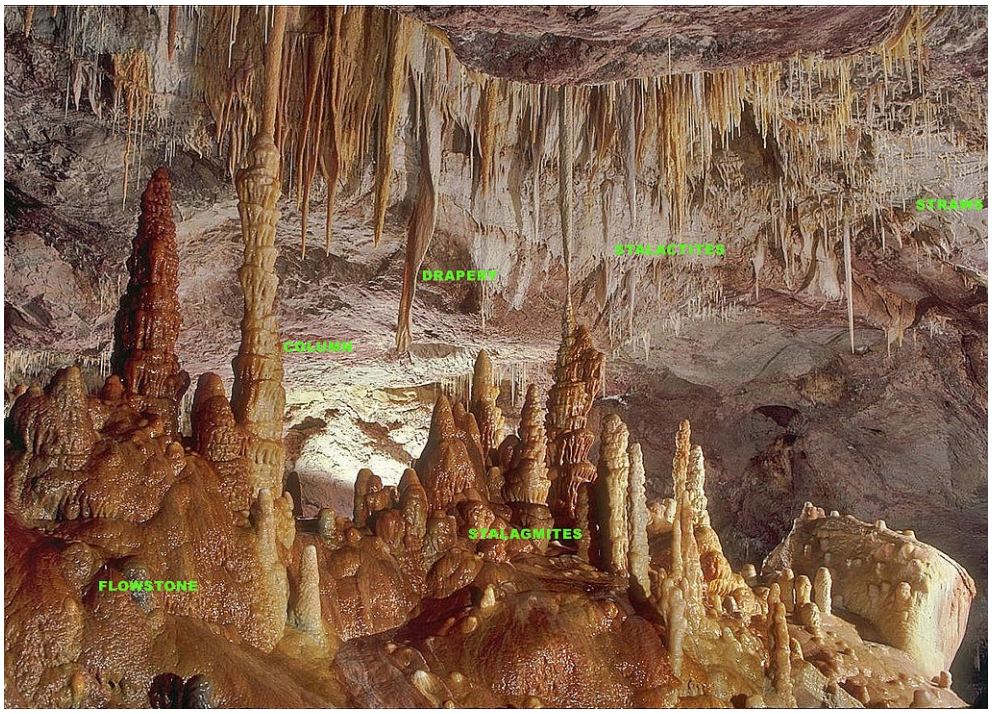
Slika 1.6 Freatični prolazi[4]

Vadozni prolazi nastaju tokom voda iznad koje ima slobodnog prostora u prolazu. Tada pritisak neće biti toliko velik te će najviše erodirati dno prolaza. Iz tog razloga vadozni prolazi imaju oblik kanjona. Često nastaju iz otprije formiranih freatičnih prolaza nakon snižavanja razine vode.



Slika 1.7 Vadozni prolazi[4]

U špiljama se također mogu pronaći sekundarne nakupine minerala pod nazivom speleotemi (engl. *Speleothems*). Najpoznatiji speleotemi su stalaktiti i stalagmiti, ali postoje još stalagnati, heliktiti, draperije, slamčice i drugi. Razne speleoteme u jednoj pećini prikazuje Slika 1.8.



Slika 1.8 Razni speleotemi u jednoj pećini[5]

Njihovo generiranje bi moglo biti čitav rad samo za sebe, tako da se ovaj rad neće baviti speleotemima.

2. Metode generiranja špilja

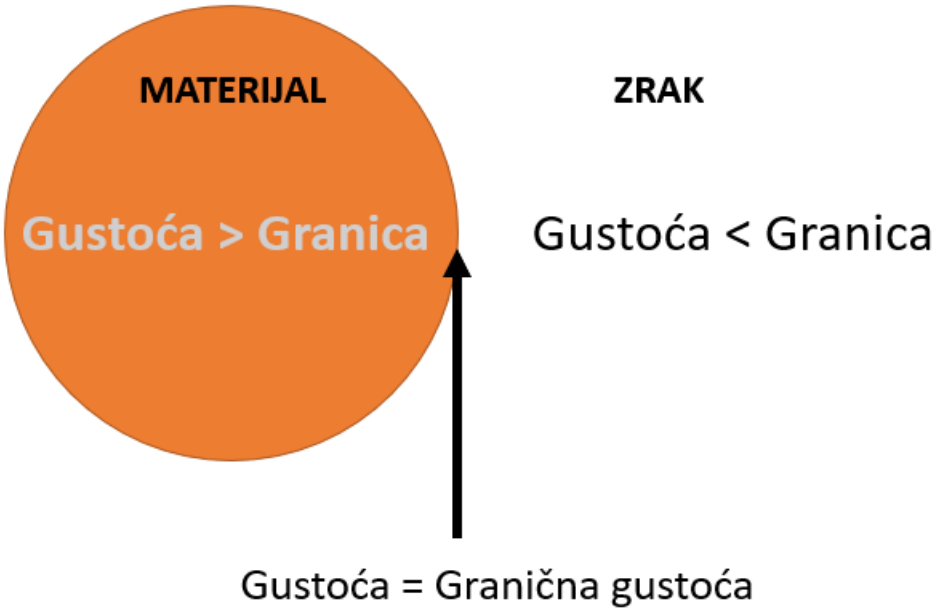
Špilje koje ću u ovom radu generirati biti će krške špilje nastale otapanjem mekih stijena protokom vode. Prolazi će biti pretežno freatični, a sustavi će biti razgranatog oblika. Kako je već spomenuto, duljina špilje biti će u desecima ili stotinu metara, a širine nekoliko metara. S obzirom da je cilj omogućiti korisniku da istražuje sustav špilja, svi prolazi moraju biti dovoljno veliki da objekt igrača prođe kroz njih.

S tim specifikacijama može se započeti tražiti odgovarajuću metodu generiranja.

2.1. Osnove prikaza terena algoritmom pokretne kocke

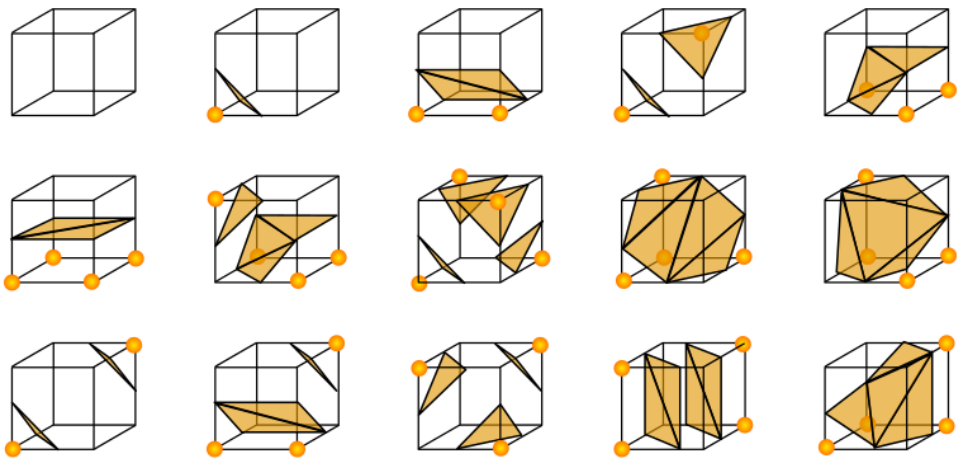
Ovaj rad nastavlja se na moj prijašnji rad na preddiplomskom studiju u kojem sam proceduralno generirao teren putem trodimenzionalnog Perlinovog šuma te koristio algoritam pokretne kocke (engl. *Marching cubes*) za prikaz generiranih podataka. [6] U svrhu boljeg razumijevanja nadolazećih sekcija, ovdje ću kratko ponoviti osnove generiranja i prikaza terena u 3D prostoru.

Svakoj točki u 3D prostoru potrebno je dodijeliti vrijednost koja predstavlja gustoću. To nije gustoća u znanstvenom smislu omjera mase i volumena, već mjera koliko je ta točka duboko ispod površine, što ilustrira Slika 2.1. Tada je potrebno usporediti gustoću svake točke s proizvoljnom graničnom gustoćom i time odrediti je li ta točka unutar ili izvan terena. [6]



Slika 2.1 Gustoća točke kao granica materijala i zraka[6]

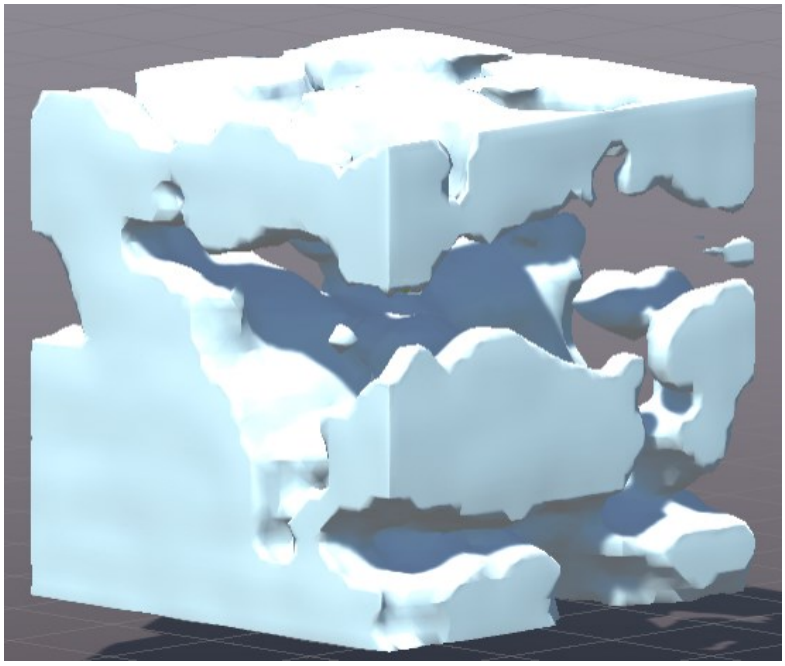
Nakon što je svakoj točki u prostoru dodijeljena gustoća, algoritmom pokretne kocke generiraju se trokuti mreže na granici između zraka i terena. Na taj način može se prikazati teren koji nije samo visinska mapa, već ima konkavne strukture poput tunela i prevjesa. Način na koji algoritam postavlja trokute ovisno o konfiguraciji prikazuje Slika 2.2.



Slika 2.2 Originalnih 15 konfiguracija

2.2. Metoda trodimenzionalnog Perlinovog šuma

Najjednostavniji način generiranja podzemnog prostora je korištenje 3D Perlinovog šuma. U alatu Unity Perlinov šum vraća vrijednosti u rasponu 0 do 1 te je prosječna vrijednost 0.5. Ako se granica između zraka i materijala postavi na tu vrijednost, u prosjeku će 50% prostora biti zemlja, a 50% praznina. To je dobro za generiranje velikih prostora, ali njihov oblik nije nalik na stvarne špilje koje više imaju oblik tunela koji ponekad vode do većih prostorija, što prikazuje Slika 2.3. Također, jedan od očitih problema je postojanje lebdećih elemenata. Iako su špilje prepune „magičnih“ stvari, lebdeće stijene nisu jedna od njih.



Slika 2.3 Teren s podzemnim prostorima generiran Perlinovim šumom

Kada bi se granična gustoća između materijala i zraka pomaknula da bude manje praznina, tada bi dobili mnogo nepovezanih „balončića“ zraka ispod zemlje, a ne zanimljivi sustav kojem korisnik može pristupiti s površine te ga istraživati.

Ovaj pristup bi bio dobar za videoigru s elementima fantazije, ali u svrhu generiranja realističnih pećina potrebno je pronaći drukčiju metodu.

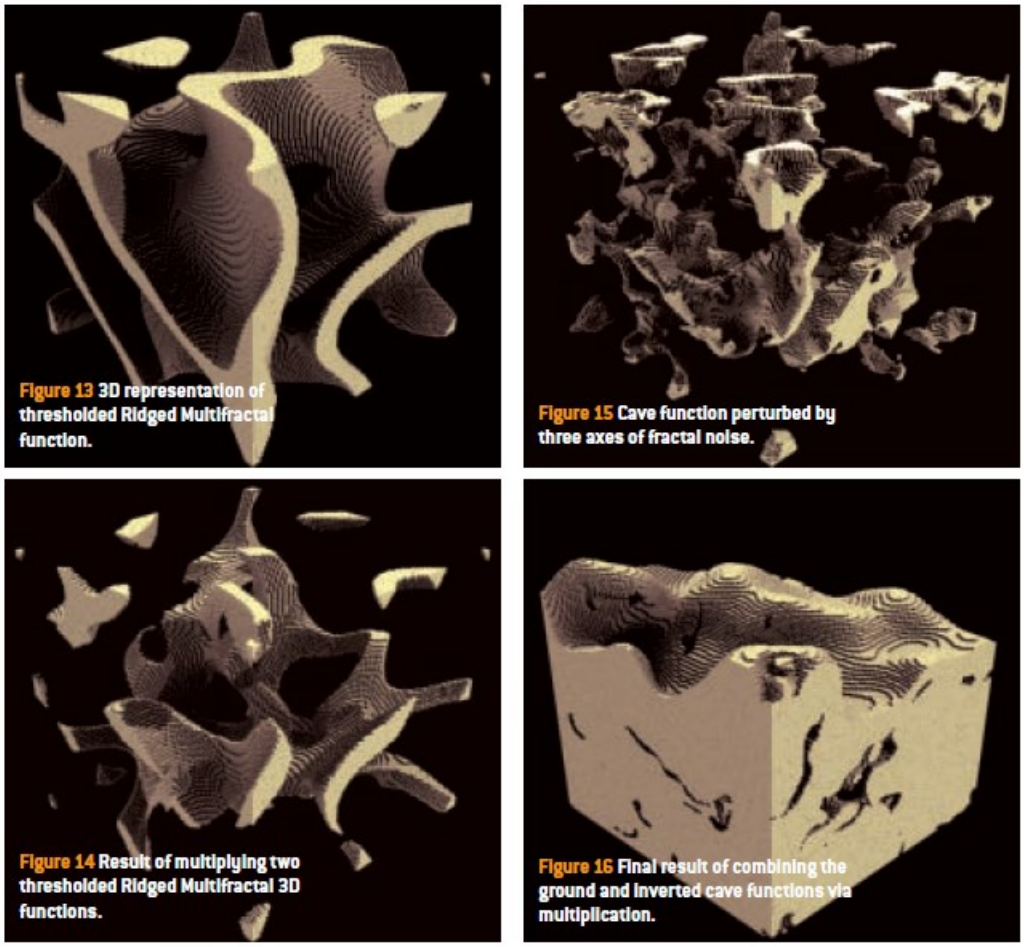
2.3. Metoda grebenastog multifraktalnog šuma

Perlinov šum izgledom ne podsjeća nimalo na špilje, ali postoje i druge vrste šumova. U članku *Creator of Worlds* u časopisu *Game Developer Magazine* (GDM) [7] opisana je tehnika proceduralnog generiranja špilja korištenjem grebenastog multifraktalnog šuma

(engl. *Ridged multifractal noise*). Takav šum generira se na način sličan Perlinovom, ali se na izlaznu vrijednost svake oktave šuma primjeni funkcija apsolutne vrijednosti. [8] Rezultat prikazuje Slika 2.4. Prikazani šum generiran je koristeći biblioteku FastNoise Lite u jeziku C#. [9] Radi jasnoće, vrijednosti šuma su zaokružene na 1 ili 0 te prikazane bijelom ili crnom bojom. Finalna 2D tekstura jako liči na sustav tunela.

Slika 2.4 Dvodimenzionalni grebenasti multifraktalni šum

Nažalost, kada se ista funkcija prikaže u 3D prostoru, rezultati ne liče na sustav tunela, već zaobljene površine poput ljusaka. Tehnika koja je predložena u GDM članku je množenje dva takva trodimenzionalna šuma s različitim sjemenima (engl. *Seed*) čime bi ostao samo njihov presjek. Njihove rezultate prikazuje Slika 2.5.



Slika 2.5 Metoda generiranja špilja prikazana u GDM travanj 2011

Glavna prednost ove metode je to što je generiranje pećina analogno generiranju ostatka terena. Rezultati umnoška šumova jednostavno se oduzmu od terena i time stvore prolazi. To je jako praktično kod dinamičkog generiranja komada terena (engl. *Chunks*), jer je svaki komad nezavisan o drugima pa nije bitan redoslijed u kojem se generiraju.

Mana ovog pristupa je manja razina kontrole nad izgledom i svojstvima špilja. Dodatno, ispod svakog komada terena bio bi podjednako velik i kompleksan sustav špilja što možda nije poželjno svojstvo. Ovaj pristup također ne garantira odsustvo lebdećih elemenata i špilja koje su samo „balončići“ zraka ispod zemlje.

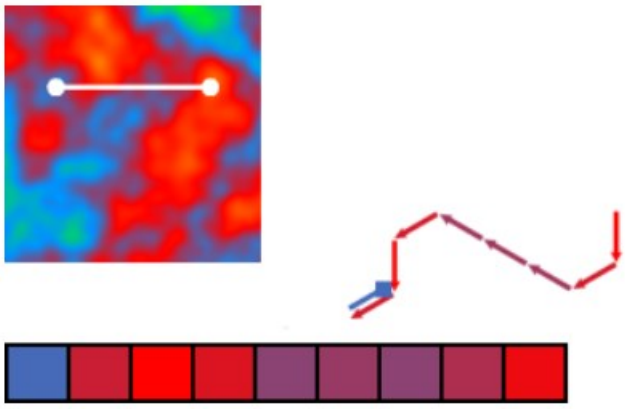
2.4. Metoda Perlinovog crva

Ova metoda često se koristi za generiranje rijeka. S obzirom da rijeke stvaraju špilje, za očekivati je da se uspješno koristi i za generiranje špiljskih sustava. Rijeke i špilje su slične po obliku. Ponekad su ravne, a ponekad vrludaju. Ponekad su horizontalne, a nekad imaju

nagle padove. Negdje su šire, a negdje uže. Često se granaju ili slijevaju u glavni tok. Ono najbitnije je to da imaju početak i kraj. Dosad prikazane tehnike samo su stvarale podzemne prostore, ali oni nisi vodili nikamo, niti imali neki definirani smjer.

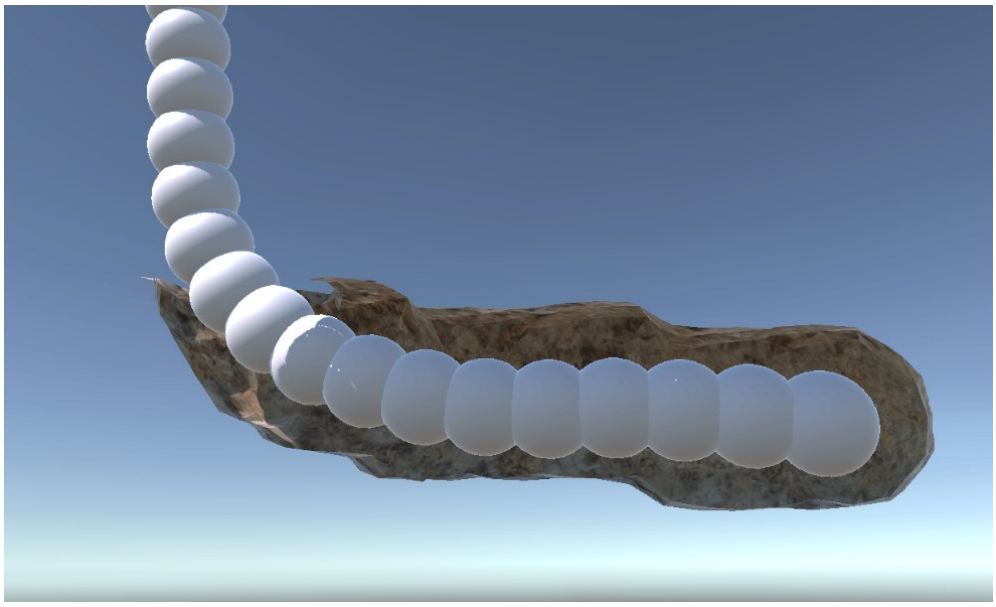
Postoje spekulacije da je metoda Perlinovog crva korištena za generiranje špilja u videoigri Minecraft, ali nisam mogao pronaći pouzdani izvor koji to potvrđuje. Čak i da to nije slučaj, špilje u Minecraft-u dizajnirane su tako da igrača vode nekamo. One nisu samo rupe ispod zemlje, već putevi kojim igrač dolazi do sadržaja. Navedeno svojstvo može se ostvariti koristeći metodu Perlinovog crva.

Algoritam je jednostavan i intuitivan, a ilustrira ga Slika 2.6. Svaki crv ima svoju početnu poziciju, orijentaciju te maksimalni broj segmenata. Za svaki segment crv se pomakne fiksnu udaljenost u smjeru određenom trenutnom orijentacijom. Uzorkuje se funkcija Perlinovog šuma te se na temelju dobivene vrijednosti ažurira orijentacija. Dobiveni rezultat je crv čiji oblik je određen glatkim šumom. Što je veća frekvencija, to će njegov oblik biti više promjenjiv.



Slika 2.6 Metoda Perlinovog crva[10]

Najbrži način da se od crva stvori špilja je da se u radijusu oko svakog segmenta gustoća postavi na onu koja odgovara praznom prostoru. U tom slučaju će rubovi pećine biti jako grubi zbog naglog skoka u vrijednosti. Za dobivanje glatkih površina sfere, potrebno je interpolirati vrijednost točke između trenutne gustoće i gustoće zraka, na temelju udaljenosti od centra sfere. Ako se tada primjeni algoritam pokretne kocke, dobije se rezultat koji prikazuje Slika 2.7.



Slika 2.7 Kopanje tunela oko Perlinovog crva

Iznos radijusa svakog segmenta može se modificirati vrijednostima šuma te time dobiti blage prijelaze u širini pećine.

Grananje pećine može se ostvariti tako što se za neki nasumično odabrani segment postavi da bude početak novog crva s orijentacijom drukčijom od trenutne.

S obzirom da špilje većinom monotono idu prema dolje, funkcija kojom se određuje orijentacija može se modificirati da poštuje to svojstvo.

3. Teksturiranje špilja

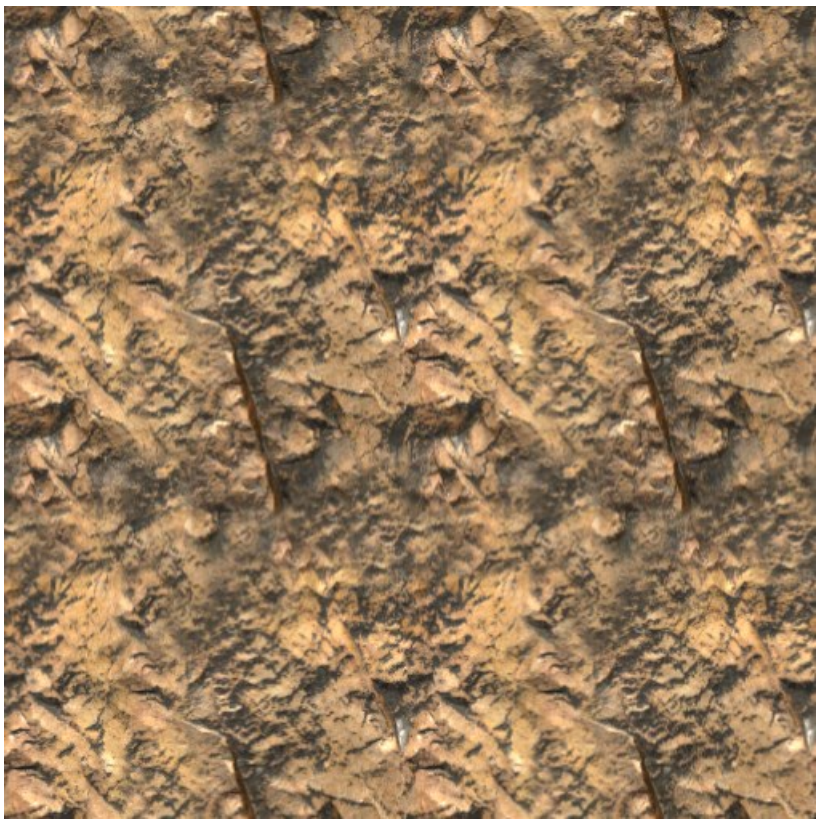
Izgled površina u špiljama iznimno je ovisan o materijalu i uzroku nastanka špilje. Čak i unutar iste špilje mogu postojati dijelovi drukčijeg izgleda. U krškim špiljama većinom dominiraju nijanse smeđe, bijele, žute, narančaste i sive. Speleotemi i dijelovi špilje u kojima su nastali često su svijetle, ponekad čak i bijele boje, iako mogu biti obojani kemijskim spojevima koji se nađu u vodi. Glatki su i vlažni, jer su prekriveni slojem otopljenog vapnenca. Ostali dijelovi pećine često liče više na zemlju i kamenje, sa smeđe-narančastim i sivim nijansama te grubljim teksturama. Treba uzeti i u obzir činjenicu da su špilje u potpunom mraku, pa boje koje u njima vidimo podosta ovise o svjetlu kojim se osvjetljavaju. Primjer boja i tekstura u krškoj pećini prikazuje Slika 3.1.

Slika 3.1 Boje i teksture u krškoj pećini [11]

3.1. Generiranje teksture

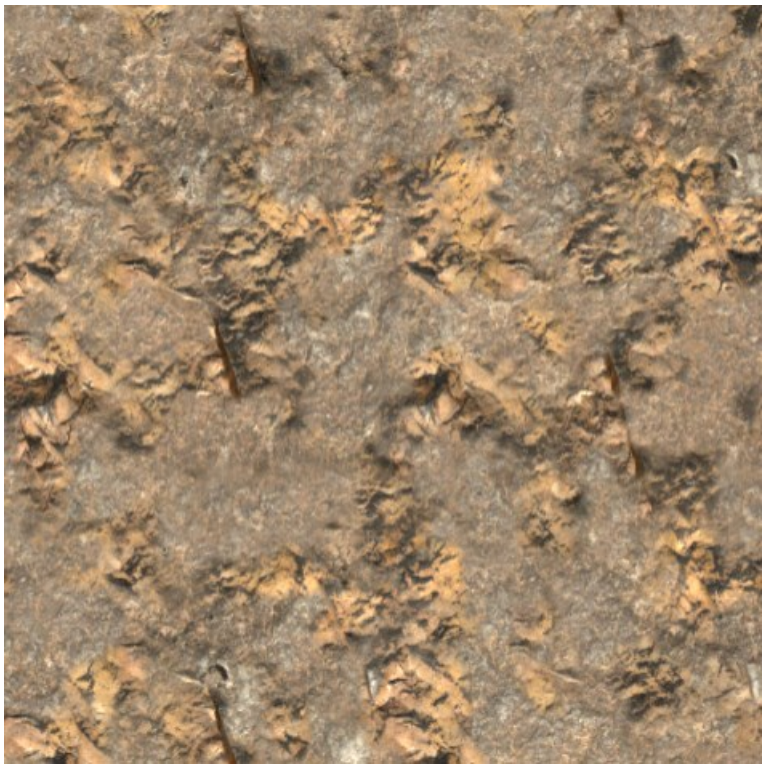
Špilje imaju jako raznovrsne teksture pa ne bi bilo prihvatljivo samo ponavljati istu teksturu bez varijacije. Ponavljanje teksture lako se primijeti od strane korisnika te nije

estetski poželjno u većini primjena. Takav slučaj prikazuje Slika 3.2. Teksture korištene u ovom radu besplatno su preuzete s Unity Asset Store-a. [13]



Slika 3.2 Primjetno ponavljanje teksture

Tehnika za razbijanje monotonije te smanjivanje primjetnosti šavova među teksturama koja je korištena u ovom radu, preuzeta je od Bena Clowarda. [12] Ideja je sljedeća: uzorkovati dvije teksture različitim frekvencijama te blijediti između njih na temelju vrijednosti šuma. Teksturama će se šavovi nalaziti na različitim mjestima zbog različite frekvencije uzorkovanja, a šum uopće nema šavova jer nema ponavljanja. Njihovom kombinacijom dobije se tekstura koja se naizgled ne ponavlja, već samo beskonačno nastavlja. Rezultat prikazuje Slika 3.3.



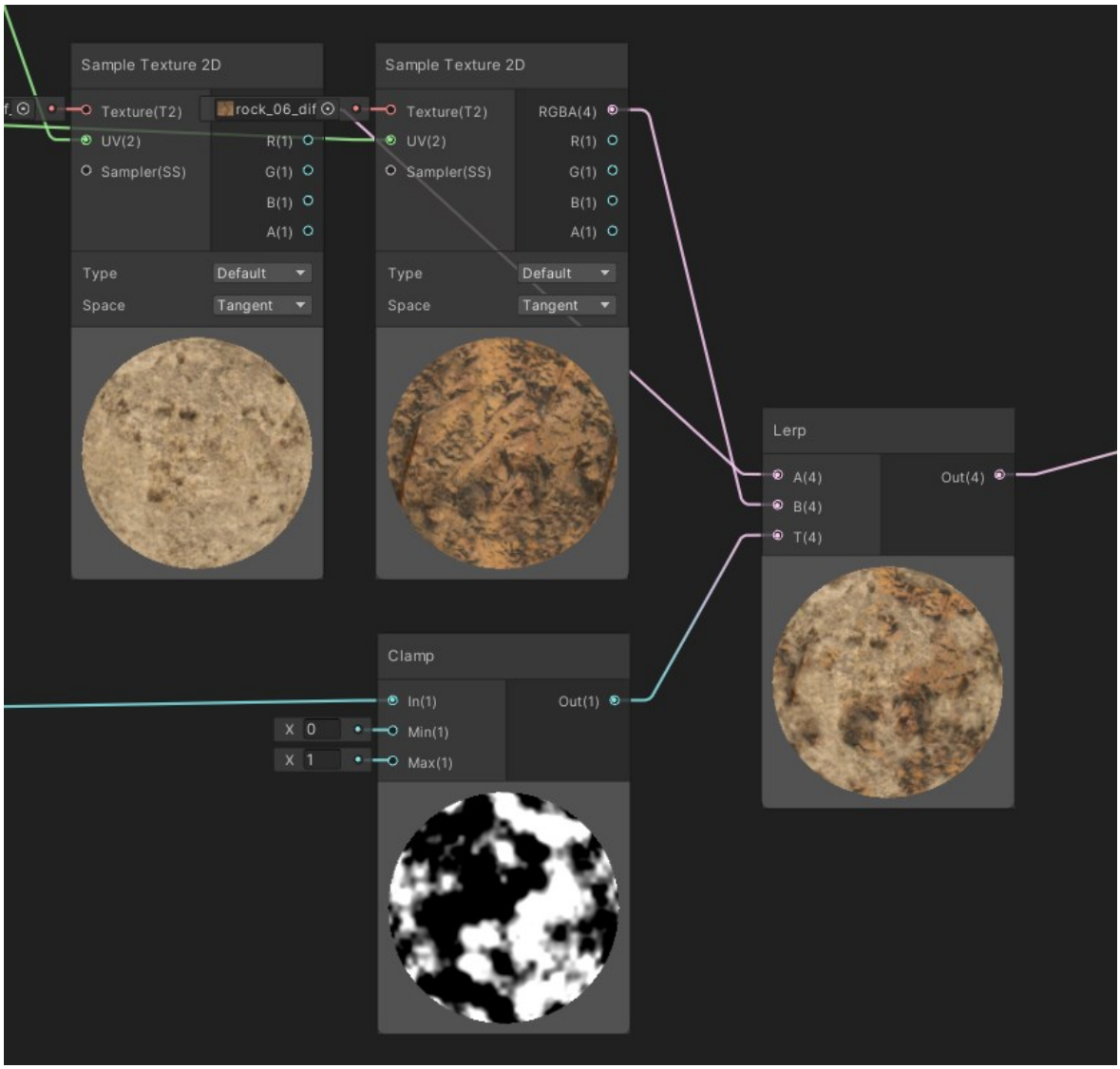
Slika 3.3 Tekstura nastala miješanjem dviju tekstura različitih veličina

U ovom radu sjenčare nisam pisao ručno, već sam koristio jednu od novijih značajki alata Unity pod imenom graf sjenčara (engl. *Shader graph*). Umjesto da se sjenačari vrhova i fragmenata pišu ručno, graf sjenčara omogućava njihovo generiranje povezivanjem čvorova u grafu. Osim što pojednostavljuje proces stvaranja sjenčara, graf ima dodatnu prednost trenutnog odziva. Svaka promjena u grafu odmah je vidljiva u pregledu pa je iteracija puno brža i jednostavnija.

Kako bi se koristila ova funkcionalnost alata Unity, potrebno je preuzeti jedan od paketa *High Definition Render Pipeline* (HDRP) ili *Universal Render Pipeline* (URP) koji se prije zvao *Lightweight Render Pipeline* (LRP).

Dio grafa sjenčara kojim je generirana prijašnja tekstura prikazuje Slika 3.4. Omjer frekvencija uzorkovanja dviju tekstura postavljen je na 3 : 4. Kao maska za miješanje korišten je jednostavni šum (engl. *Simple nosie*), koji također dolazi kao jedan od čvorova u grafu. Vrijednostima šuma podignut je kontrast kako teksture ne bi bile previše mutne. Na kraju se koristi funkcija linearne interpolacije između dviju tekstura, a vrijednosti šuma koriste se kao parametar.

Identični graf napravljen je i za mape normala kako bi i oblik površine imao više varijacije te izgledao prirodniji s teksturama.



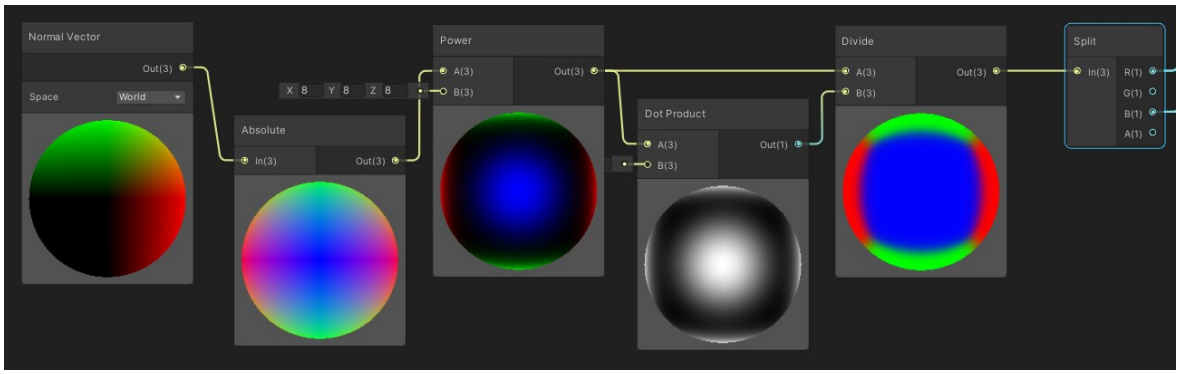
Slika 3.4 Miješanje tekstura korištenjem šuma u grafu sjenčara

3.2. Triplanarni sjenčar

S obzirom da će se špilje generirati proceduralno, ne može se koristiti UV mapiranje, već je potrebna metoda koja će se prilagoditi svakom obliku dinamički generirane špilje. U svom prošlom radu u tu svrhu koristio sam triplanarnu projekciju (engl. *Triplanar projection*). [6] Ideja je projicirati ponavljajuću teksturu na temelju globalnih koordinata, a ne lokalnih. Tekstura se projicira iz triju okomitih ravnina pa otud dolazi naziv triplanarna. Kako bi se spriječilo rastezanje teksture na prijelazima, vrijednosti se interpoliraju na temelju nagiba normale.

Triplanarni sjenčar u ovom radu napravljen je po uzoru na referencu [14]. Ideja je projicirati na površinu piksele teksture čija ravnina ima normalu najsličniju normali

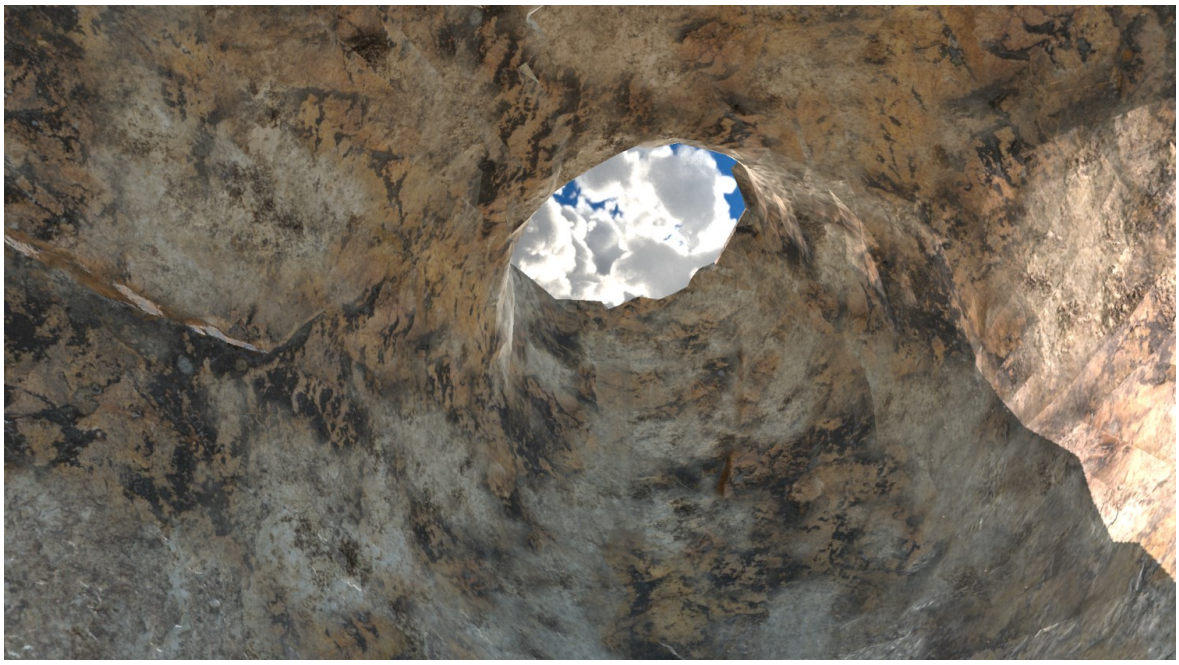
površine. Na prijelaznim dijelovima će se teksture interpolirati, ali širina prijelaznog područja se može regulirati. Slika 3.5 prikazuje niz čvorova kojima se stvara maska smjerova na temelju koje će se projicirati teksture.



Slika 3.5 Stvaranje maske za određivanje strane svijeta

Prvo se uzimaju globalne koordinate, koje mogu imati pozitivne i negativne vrijednosti. S obzirom da želimo istu teksturu i na prednjoj i na stražnjoj strani terena, primjenjuje se apsolutna funkcija na koordinate. Zatim se koristi potenciranje kako bi se svaka strana izolirala od ostalih kako ne bi bilo mutnih prijelaza. Na kraju se te normale podijele svojim amplitudama kako bi ostala samo maska smjera. RGB kanali se tada razdvoje i koriste kao parametri za interpolaciju tekstura i normala.

Kada se navedeni triplanarni sjenčar primjeni na teren, dobiju se rezultati koje prikazuje Slika 3.6. Korisno je i spomenuti da za korištenje URP sjenčara u projektu, korisnik mora stvoriti novi *asset* pod nazivom `UniversalRenderPipelineAsset`. Zatim je potrebno postaviti *Scriptable Render Pipeline Settings* u grafičkim postavkama projekta na novostvorenu datoteku. U suprotnom će sjenčar dojaviti grešku. *Skybox* korišten u ovom radu preuzet je s reference [15].



Slika 3.6 Teksturirani teren

4. Neograničeni teren

U brojnim videoigrama u kojima se koristi proceduralno generiranje terena, ne stvori se čitav teren odjednom, već se podijeli na komade (engl. *Chunks*) koji se dinamički generiraju i prikazuju kada im se igrač dovoljno približi. Osim što ovo smanjuje opterećenje računala, jer ne mora stvarati podatke i objekte koje igrač ne može vidjeti, također omogućava generiranje beskonačno velikog terena.

4.1. Podjela terena na komade

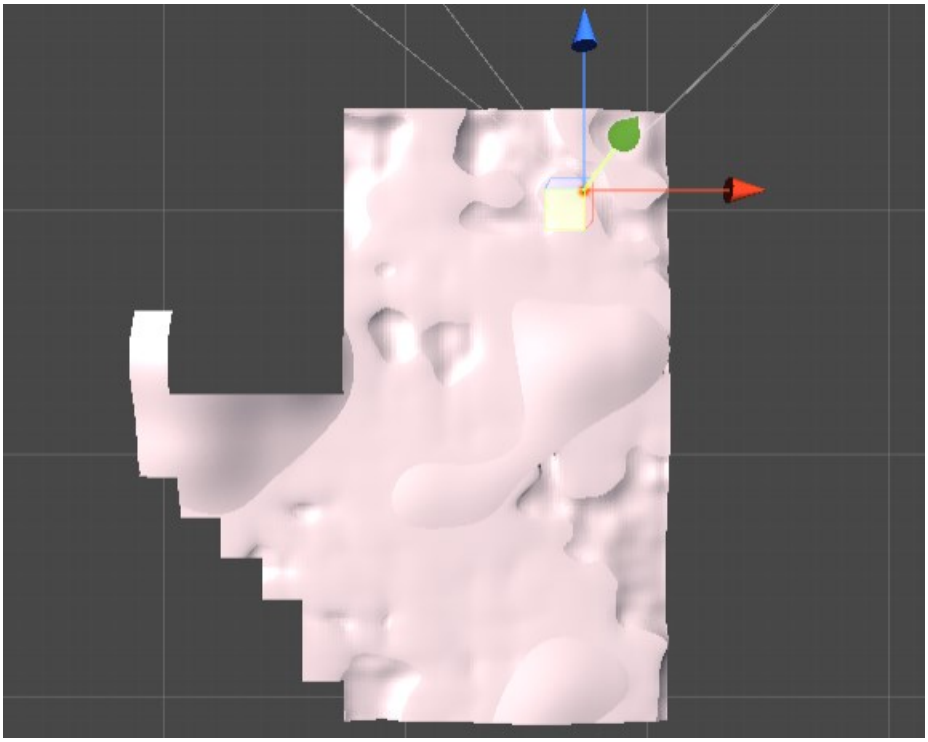
Veličina komada je proizvoljna, ali cilj je pronaći dobar omjer između količine generiranih i količine potrebnih podataka. Ako su komadi premaleni, prečesto će nestajati i nastajati novi te se neće moći dobro iskoristiti paralelizam, u slučaju da se koristi. Ako su preveliki, onda će patiti performanse zbog naglih skokova u količini računanja.

Ideja dinamičkog stvaranja je uvijek biti korak ispred igrača i u pozadini pripremati podatke koje će kasnije zatrebati. Maksimalnu veličinu komada ograničavaju broj dretvi na sjenčaru računanja ako se koristi te maksimalni broj vrhova u mreži (engl. *Mesh*). Sjenčar računanja ima 1024 dretve pa maksimalne dimezije 3D komada mogu biti 8 x 16 x 8. Prema uobičajenim postavkama, indeksi vrhova u mreži se u alatu Unity pohranjuju u 16-bitnom cijelom broju pa je maksimalni broj vrhova 65536.

Svaki komad ima svoju početnu točku koja može biti određena globalnim koordinatama ili koordinatama komada. Npr. ako je veličina komada 32 x 32, i promatra se komad čija početna točka se nalazi na koordinatama (32, 64), njegove koordinate komada bile bi (1, 2), to jest globalne koordinate podijeljene s dimenzijama komada. Za svaku točku u prostoru mogu se izračunati koordinate relativno nekom komadu. Tako bi točka (34, 58) relativno prijašnjem komadu imala koordinate (2, -6). Lokalne koordinate su bitne jer se na temelju njih može odrediti kojim točkama u komadu treba promijeniti gustoću pri kopanju tunela oko Perlinovog crva. Za svaku točku u komadu provjerava se nalazi li se unutar radijusa od centra trenutnog segmenta.

U kôdu su određene dvije udaljenosti: udaljenost za stvaranje te udaljenost za uklanjanje komada. Generator terena na temelju pozicije igrača određuje koji bi sve komadi trebali

biti generirani, a zatim se provjerava lista postojećih komada i dodaju oni koji nedostaju. Komadi izvan radijusa za uklanjanje se brišu, te će igračev povratak na to područje izazvati ponovno generiranje. Radijus za uklanjanje očekivano veći od onoga za stvaranje komada. Nisu jednaki, jer je za očekivati da će igrač često hodati između 2 ista komada pa bi bilo neefikasno da se kod svakog prijelaza preko granice komada ponovno briše i stvara teren. Kako bi se smanjilo računsko opterećenje, komadi se brišu i generiraju samo kada igrač prijeđe u novi komad. Pogled na dinamičko generiranje komada iz ptičje perspektive prikazuje Slika 4.1.



Slika 4.1 Komadi generirani za vrijeme igračeva kretanja

Komadi dolje lijevo su izvan zone stvaranja, ali unutar zone uklanjanja pa su zato još uvijek vidljivi. Kad bi se igrač vraćao nazad istim putem kojim je došao, neko vrijeme se ne bi ni stvarali ni uklanjali komadi, jer su stari još uvijek zapamćeni.

Svaki komad ima svoju poziciju, materijal, mrežu i druge atribute potrebne da se prikaže. Pri stvaranju novih komada jednostavno se stvori nova instanca razreda `Chunk`, čiji su mreže i sudarač (engl. *Collider*) generirani na temelju podataka dobivenih šumom.

4.2. Perlinovi crvi u neograničenom terenu

Jedna od globalnih konstanta u kôdu je sjeme (engl. *Seed*). Na temelju sjemena može se garantirati da će izvor nasumičnih podataka vraćati isti izlaz za isti ulaz. Ovo svojstvo je bitno jer je nužno da svaki komad izgleda nepromijenjen neovisno koliko puta se generira i ukloni.

4.2.1. Generiranje crva

Perlinov crv generira se od glave prema naprijed pa je prvi korak proceduralnog generiranja pećina odrediti u kojim komadima će biti glava crva, a u kojima neće. U tu svrhu koristio sam Perlinov šum. Metoda `ChunkContainsWormHead` na temelju koordinata komada i sjemena računa vrijednost Perlinovog šuma te ju uspoređuje s graničnom vrijednosti. Ako je veća, tada se u tom komadu nalazi glava nekog crva. Za isto sjeme i isti komad, metoda će uvijek vratiti isti rezultat. Podešavanjem vrijednosti praga može se mijenjati učestalost špilja. U svakom komadu može se nalaziti glava od najviše jednog crva, ali može prolaziti proizvoljan broj crva koji započinju u drugim komadima.

Slika 4.2 prikazuje isječak kôda za generiranje Perlinovog crva. Koordinate glave u komadu se također određuju na temelju sjemena, tako da će isti crv uvijek počinjati na istom mjestu i imati istu orijentaciju. Za svaki segment određuje se radijus utjecaja na teren oko njega te se pohranjuje popis svih komada na koje utječe. Taj skup se kasnije koristi kod izgradnje komada. Novi smjer određuje se rotacijom oko dviju osi za kut određen šumom, a zatim se crv pomakne u tom smjeru za iznos duljine segmenta.

```
public void GenerateWorm(Vector3 headPosition, Vector3 startDirection, int segmentNumber)
{
    Vector3 currentSegmentPosition = headPosition;
    float splitThreshold = 1f;

    for (int i = 0; i < segmentNumber; i++)
    {
        int segmentRadius = GetSegmentRadius(i, headPosition, segmentRadiusNoiseScale, segmentRadiusNoiseOctaves);
        HashSet<Vector3Int> chunksAffectedBySegment = Utilities.GetAllChunksInRadius(
            currentSegmentPosition, segmentRadius, width, height, length);
        segmentChunks.Add(chunksAffectedBySegment);
        activeChunks.UnionWith(chunksAffectedBySegment);
        segmentPositions.Add(currentSegmentPosition);
        segmentRadiuses.Add(segmentRadius);

        float angleX = (Noise.GetNormalizedNoise(i, headPosition.x,
            directionNoiseScale, directionNoiseOctaves) + 1) / 2;
        float angleY = Noise.GetNormalizedNoise(i, headPosition.y,
            directionNoiseScale, directionNoiseOctaves);
        angleX *= startDirection.x > 0 ? 1 : -1;
        Vector3 direction = Quaternion.AngleAxis(90 * angleX, Vector3.right) *
                            Quaternion.AngleAxis(180 * angleY, Vector3.up) * startDirection;
        if(direction.y > 0)
        {
            direction.y = 0;
        }
```

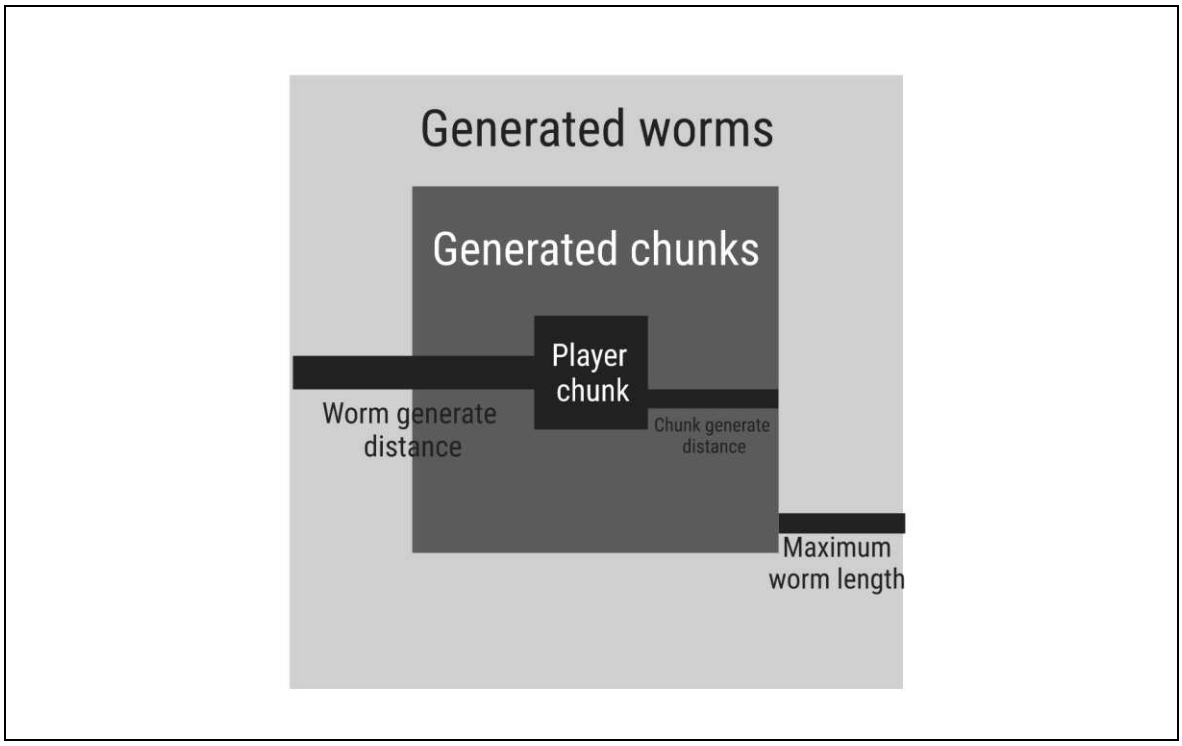
Slika 4.2 Dio kôda za generiranje Perlinovog crva

Sve rotacije određuju se u odnosu na početni smjer crva. Kako bi pećina uvijek išla prema dolje, dodana je provjera koja određuje hoće li se vektor smjera rotirati u pozitivnom ili negativnom smjerno ovisno o X komponenti početnog smjera. S obzirom da će Y komponenta početnog smjera uvijek biti manja ili jednaka nuli, a rotirati će se uvijek prema negativnoj Z osi u maksimalnom iznosu od 90 stupnjeva, crv će monotono ići prema dolje.

Dimenzije crva ograničene su konstantama. To je nužno jer je za generirane komade potrebno znati prolazi li kroz njih neki crv te ako prolazi, izgraditi odgovarajući špilju. Ako crv započinje u nekom komadu koji još nije generiran, a prolazi kroz komad koji se upravo generira, to je nužno detektirati te crva generirati. Kada bi crvi bili beskonačno dugi, tada bi bilo potrebno obaviti beskonačno mnogo provjera. S ograničenjima na broj, udaljenost i radijus segmenata, može se odrediti maksimalni radijus utjecaja svakog komada, a on odgovara izrazu:

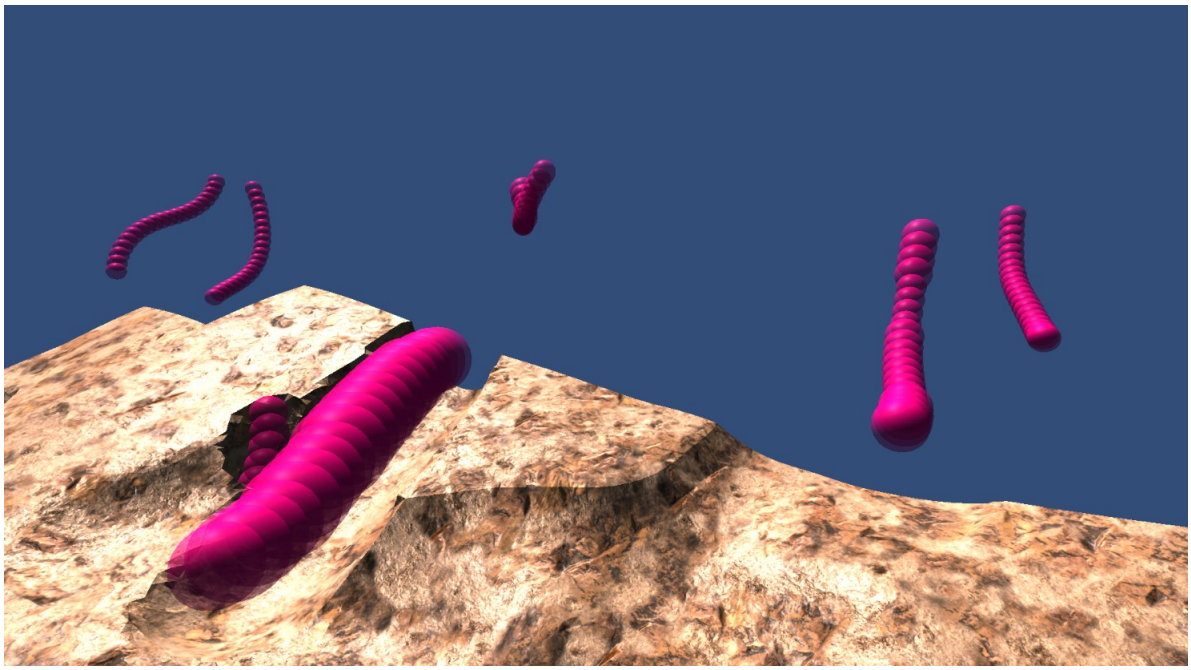
```
maxWormLength = maxWormSegmentLength * maxWormSegmentNumber +
maxWormSegmentRadius;
```

Kao što se novi komadi generiraju unutar radijusa za stvaranje, tako se i Perlinovi crvi generiraju unutar određenog radijusa, koji odgovara sumi maksimalne duljine crva te radijusa za dodavanje komada. Na ovaj način može se garantirati da će svi crvi koji prolaze kroz novi komad biti generirani prije samog komada. Za crve također postoji radijus za uklanjanje. Odnose radijusa za dodavanje komada i crva prikazuje Slika 4.3.



Slika 4.3 Graf odnosa duljine crva i stvaranja terena

Prikaz crva koji su generirani izvan radijusa za dodavanje komada prikazuje Slika 4.4. Crv koji doseže teren „kopa rupu“ kroz njega, a ostali crvi čekaju da se stvore novi komadi koji su pod njihovim utjecajem.



Slika 4.4 Crvi generirani izvan radijusa stvaranja komada

4.2.2. Generiranje pećine na temelju crva

Za svaki komad prvo se generiraju podatci gustoće na temelju šuma. Zatim se u listi postojećih pronađu špilje koje prolaze kroz taj komad. Za svaku se iterira po segmentima i provjerava utječe li segment na komad. Ako utječe, vrijednosti gustoće u radijusu oko segmenta postavljaju se na novu vrijednost, čime se stvara praznina u komadu. Tek nakon toga se provodi algoritam pokretne kocke. Isječak kôda koji stvara prazninu oko segmenta u komadu prikazuje Slika 4.5.

```
if (worm.SegmentAffectsChunk(l, chunkCoords))
{
    Vector3Int localCoords = Utilities.getLocalCoordsInsideChunk(chunkCoords, segment, width, height, length);
    int startX = Mathf.Max(0, localCoords.x - segmentRadius);
    int endX = Mathf.Min(width + 1, localCoords.x + segmentRadius);
    int startY = Mathf.Max(0, localCoords.y - segmentRadius);
    int endY = Mathf.Min(height + 1, localCoords.y + segmentRadius);
    int startZ = Mathf.Max(0, localCoords.z - segmentRadius);
    int endZ = Mathf.Min(length + 1, localCoords.z + segmentRadius);
    for (int i = startX; i < endX; i++)
    {
        for (int j = startY; j < endY; j++)
        {
            for (int k = startZ; k < endZ; k++)
            {
                float distance = Vector3.Magnitude(new Vector3(i, j, k) - localCoords);
                terrainMap[i, j, k] = Mathf.Lerp(Constants.AIR_DENSITY, terrainMap[i, j, k],
                    distance / segmentRadius);
            }
        }
    }
}
```

Slika 4.5 Isječak kôda koji stvara prazninu oko segmenta

Pronalaze se točke presjeka segmenta i odabranog komada terena te se njihova gustoća interpolira s gustoćom zraka na temelju udaljenosti.

Metode su napisane tako da rade s proizvoljnim veličinama, tako da je podržan slučaj gdje je radijus segmenta veći od veličine komada pa ih može obuhvatiti više od 4 u dvodimenzionalnom prostoru.

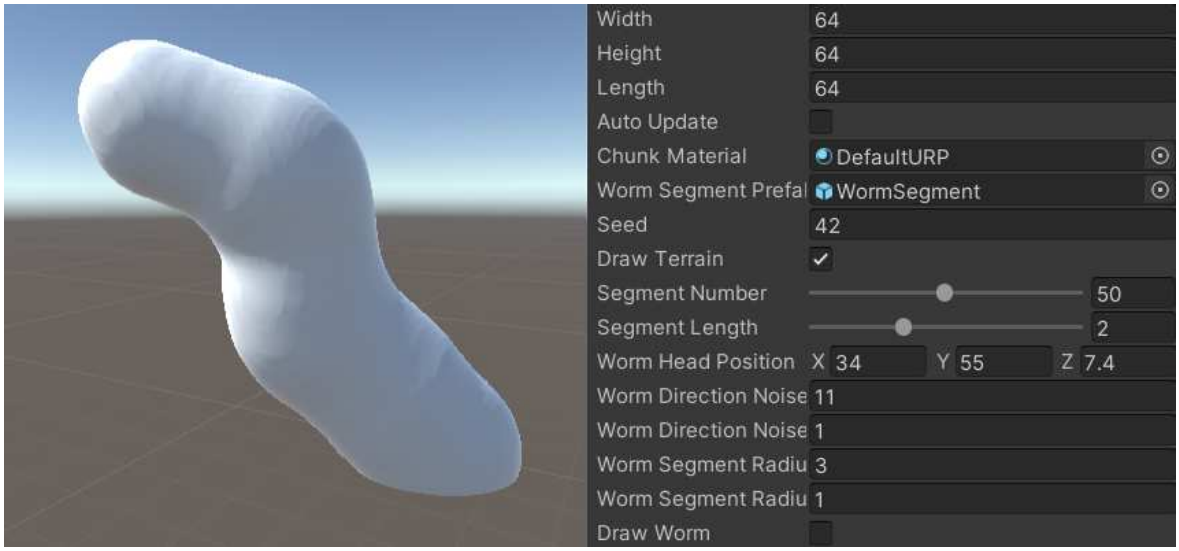
Kako bi se ubrzao proces pronalaska segmenata koji utječu na komad koji se generira, svaki crv u sebi pohranjuje skup svih komada na koje utječu njegovi segmenti, te za svaki segment također pohranjuje skup svih komada na koje samo on utječe. Pri generiranju komada provjeri se za svakog crva prolazi li on kroz komad, te u slučaju da prolazi traže se segmenti koji utječu na komad.

5. Uljepšavanje terena

Dosad prikazan dio projekta bio je samo osnovna implementacija. Sljedeći korak je uljepšavanje izgleda terena i špilja podešavanjem parametara generiranja te dodavanjem sjenčara, osvjetljenja i magle.

5.1. Izgled špilja

S obzirom da su špilje podzemne strukture, teško ih je vizualizirati. Radi lakše iteracije i vizualizacije oblika špilja, napravio sam scenu koju prikazuje Slika 5.1. U toj sceni generira se jedna špilja te se vizualizira na obrnuti način, kao teren u praznini umjesto praznina u terenu. U uređivaču mogu se podesiti parametri koji utječu na izgled i veličinu špilje. Dodani su parametri skale i broja oktava za šumove koji se koriste kod računanja smjera crva i radijusa segmenta. Promjena bilo kojeg parametra odmah se prikazuje na sceni što znatno olakšava pronalazak estetski zadovoljavajućih vrijednosti.

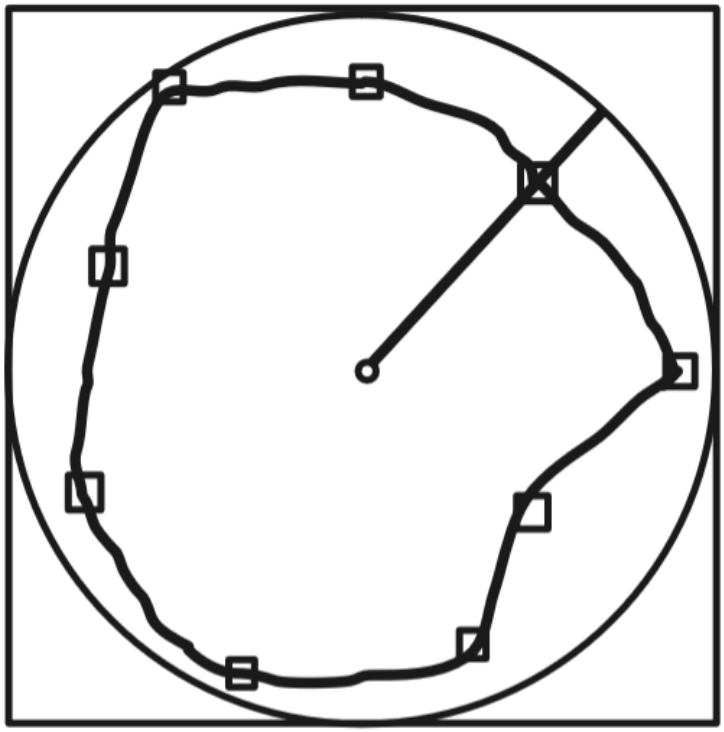


Slika 5.1 Scena za vizualizaciju špilje

5.1.1. Oblik segmenata

Špilje su dosad bile građene od segmenata koji su svi sfernog oblika. Iako su radijusi segmenta bili različiti, to nije doprinosilo dovoljno varijacije te su špilje izgledale nezanimljivo. Način na koji sam napravio da svaki segment bude drukčijeg oblika je

dodavanje varijabilnog radijusa za točke unutar segmenta. Sve točke koje leže na istom polu pravcu od središta segmenta prema vani, imaju isti radijus. Radijus će uvijek biti manji ili jednak maksimalnom radijusu segmenta, a razlika se određuje Perlinovim šumom. Iz tog razloga će promjene u radijusu biti glatke između susjednih točaka. Skicu navedene tehnike prikazuje Slika 5.2.



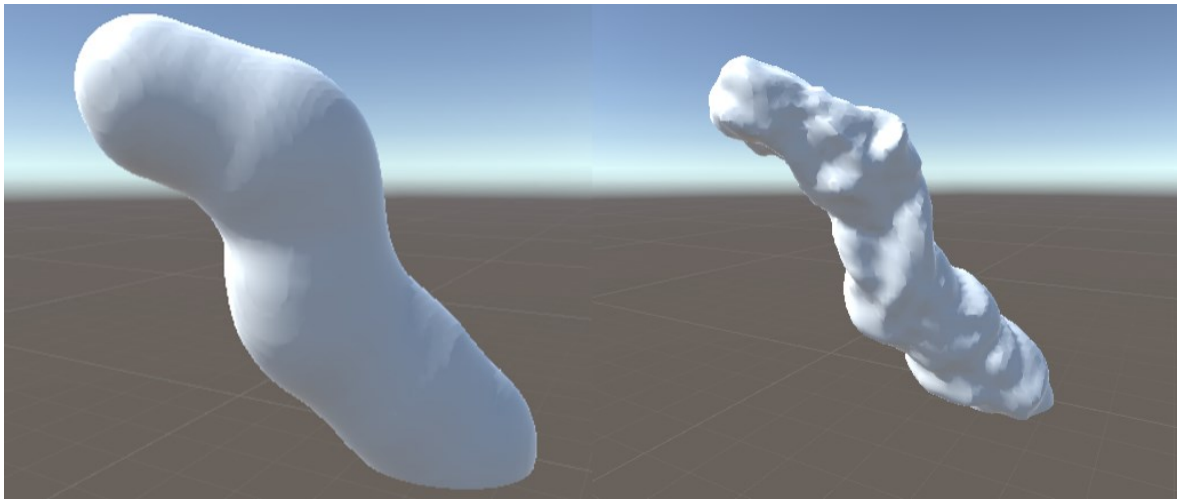
Slika 5.2 Skica određivanja oblika segmenta

Željeno svojstvo tehnike je da sve točke koje imaju isti normalizirani vektor smjera imaju isti radijus, ali samo unutar istog segmenta. U suprotnom bi svi segmenti imali isti oblik, jer bi im sve točke imale iste vektore. Iz tog razloga se osim komponenata normaliziranog vektora smjera, pri određivanju radijusa također koriste koordinate odgovarajućeg segmenta. Kôd implementacije prikazuje Slika 5.3.

```
Vector3 norm = (terrainPoint - segmentLocalCoords) * (1 / distance);
float radiusModification =
    1 - Mathf.Abs(Noise.GetNormalizedNoise(norm.x + norm.y + segmentLocation.x,
    norm.z + norm.y + segmentLocation.z, wormSegmentRadiusNoiseScale, 2)) / MAGIC_NUMBER;
float modifiedRadius = Mathf.Max(0.1f, segmentRadius * radiusModification);
float newValue = Mathf.Lerp(lerpDensity, currentDensity,
    Mathf.Clamp01(distance / modifiedRadius));
```

Slika 5.3 Implementacija varijabilnog oblika segmenta

Nakon podešavanja parametara, usporedbu starog i novog izgleda špilje prikazuje Slika 5.4. Zidovi špilje su puno grublji i prirodniji nego savršeno glatke sfere iz prijašnje implementacije.



Slika 5.4 Špilja prije i nakon dodavanja varijabilnog radijusa

Opisana tehnika ima problem koji prikazuje Slika 5.5. Zbog varijabilnog radijusa ponekad se može dogoditi da suma radijusa susjednih segmenata bude manja od udaljenosti njihovih središta što može rezultirati lebdećim elementima. Problem se može riješiti ograničavanjem maksimalne promjene radijusa ili smanjenjem udaljenosti segmenata.

Slika 5.5 Lebdeći segment koji nastaje zbog varijabilnog radijusa

5.1.2. Grananje špilja

Kako bi podržavala grananje, metoda generiranja crva napisana je kao rekurzivna funkcija. Svaka grana ponaša se kao novi crv koji ima svoj početni smjer, poziciju te maksimalni broj segmenata.

Poželjna svojstva funkcije koja određuje hoće li doći do grananja na nekom segmentu su:

- podesivost učestalosti grananja
- različite špilje se različito granaju
- grane su približno uniformno distribuirane, a ne grupirane oko jedne točke
- grane koje nastaju iz bliskih segmenata ne smiju imati isti početni smjer

Problem koji se pojavljuje pri korištenju Perlinovog šuma upravo leži u njegovom svojstvu glatkih prijelaza. Susjedne točke imaju slične vrijednosti. Kada bih koristio vrijednost šuma za određivanje hoće li doći do grananja, bilo bi puno grananja na jednom dijelu, a nimalo na drugom. Problem se može donekle riješiti korištenjem visokofrekventnog šuma, ali alternativno rješenje koje sam koristio bolje zadovoljava spomenuta svojstva. Programsku implementaciju grananja prikazuje Slika 5.6.

```
if (ShouldSplit(currentSegmentPosition, splitThreshold))
{
    splitThreshold = 1f;
    Vector3 splitDirection = GetSplitDirection(direction);
    Vector3 splitHeadPosition = currentSegmentPosition += segmentLength * splitDirection;
    int splitSegmentNumber = GetSplitLength(segmentNumber - i - 1);
    GenerateWorm(splitHeadPosition, splitDirection, splitSegmentNumber);
}
else
{
    splitThreshold -= segmentThresholdDecrement;
}
```

Slika 5.6 Isječak kôda za grananje

Postoji granična vrijednost koja se resetira na 1. Za svaki segment u kojem se nije dogodilo grananje, granična vrijednost se smanjuje za definirani iznos. Što je granična vrijednost manja, to je veća šansa za grananje. Logiku za provjeru pojave grananja prikazuje Slika 5.7. Na temelju pozicije segmenta računa se vrijednost Perlinovog šuma koja se uspoređuje s graničnom te u slučaju da je veća, događa se grananje. Nakon svakog grananja granica se postavlja nazad na 1. S obzirom da je velika većina vrijednosti šuma koncentrirana oko 0.5, očekivano je da će se dotad vrlo vjerojatno dogoditi grananje.

```
public bool ShouldSplit(Vector3 segmentPosition, float splitThreshold)
{
    float u = segmentPosition.x + segmentPosition.y;
    float v = segmentPosition.z + segmentPosition.y;
    int octaves = 3;
    float value = Noise.GetPerlinNoise(u, v, directionNoiseScale, octaves);
    return value > splitThreshold;
}
```

Slika 5.7 Pomoćna funkcija za određivanje pojave grananja

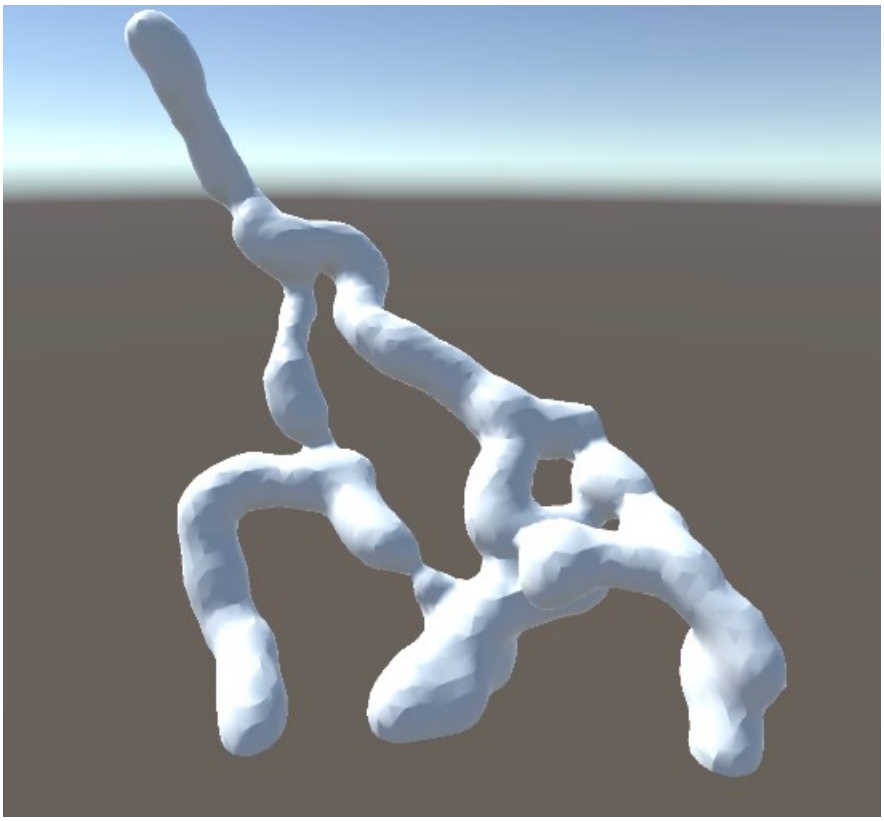
Učestalost grananja može se podesiti promjenom parametra `segmentThresholdDecrement`. Grananje se najčešće događa tek kad iznos granične vrijednosti padne oko srednje vrijednosti, ali za različite segmente granica će biti drukčija. S obzirom da se vrijednosti Perlinovog šuma nalaze u rasponu od 0 do 1, moguće je da dođe do grananja odmah na prvom segmentu, ali vjerojatnost je izrazito malena.

U slučaju pojave grananje potrebno je odrediti smjer u kojem će započeti nova grana. Kôd odgovarajuće metode prikazuje Slika 5.8.

```
public Vector3 GetSplitDirection(Vector3 originalDirection)
{
    Vector3 splitDirection = Vector3.Cross(Vector3.up, originalDirection);
    // half of the time flip the direction of split
    if(Noise.GetNormalizedNoise(splitDirection.x, splitDirection.z,
        splitDirectionNoiseScale) > 0) {
        splitDirection *= -1;
    }
    //make the split angle not 90 degrees
    float splitRotation = Noise.GetNormalizedNoise(splitDirection.z,
        splitDirection.x, splitDirectionNoiseScale);
    splitDirection = Quaternion.AngleAxis(30 * splitRotation, Vector3.up)
        * splitDirection;
    return splitDirection;
}
```

Slika 5.8 Kôd za određivanje smjera grananja

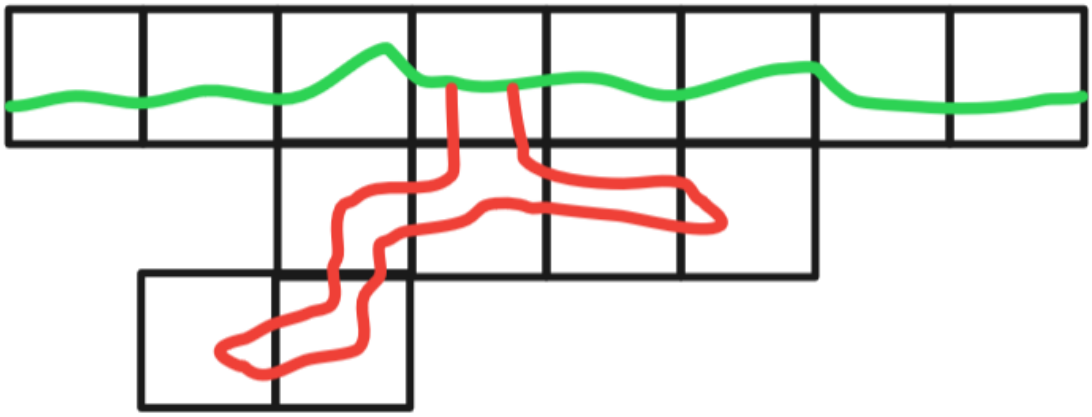
Smjer grananja početno se postavlja da odgovara normali ravnine koju određuju vektor smjera segmenta koji se grana i vektora osi Y. U 50% slučajeva taj vektor se obrne kako se grananje ne bi uvijek događalo u isto stranu. Na kraju se vektor rotira oko Y osi u rasponu [-30, 30] stupnjeva kako se grananje ne bi uvijek događalo pod kutom od 90 stupnjeva. Finalni rezultat prikazuje Slika 5.9.



Slika 5.9 Grananje špilje

5.1.3. Podzemni komadi

Kako bi se povećala maksimalna duljina i dubina špilja, dodani su podzemni komadi koji imaju negativnu Y koordinatu. Ulaz u špilju, odnosno glava crva, može nastati samo u površinskim komadima kojima je Y koordinata nula. Svi ti komadi moraju biti prikazani, ali podzemni komadi neće biti vidljivi jer su ispod površine. Njih je potrebno generirati jedino ako kroz komad prolazi špilja. U slučaju da ne prolazi, preskače se generiranje gustoća i provođenje algoritma pokretne kocke. U suprotnom, mapa gustoća inicijalizira se na 0, što odgovara gustoći terena. Ova optimizacija omogućava postojanje dubokih špilja bez da broj horizontalnih komada raste kvadratno za svaki vertikalni komad. Slika 5.10 prikazuje podzemne komade koji se generiraju te one koji se preskaču jer kroz njih ne prolazi špilja.



Slika 5.10 Vizualizacija podzemnih komada

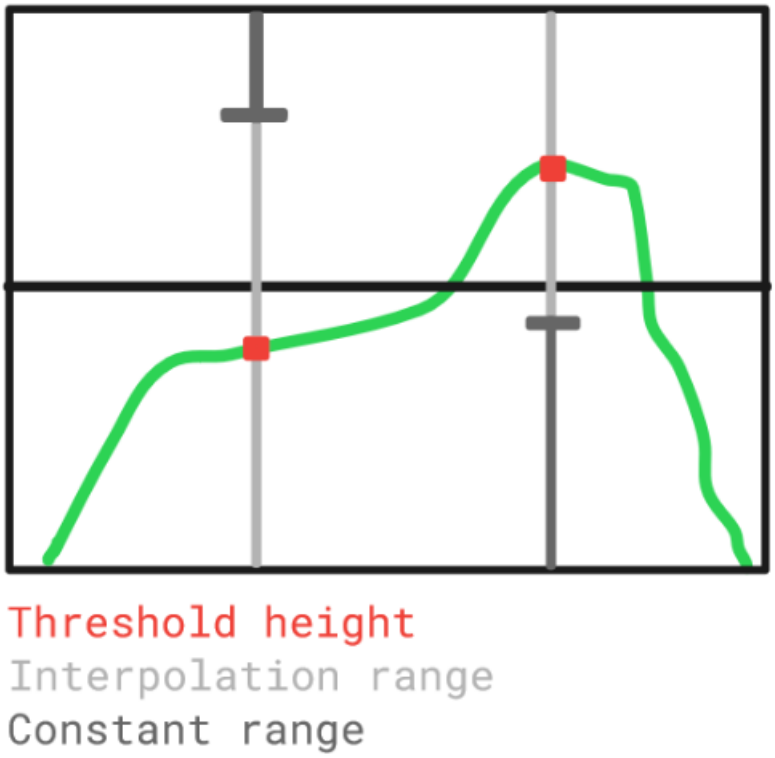
5.2. Izgled površine

Iako je fokus ovog rada na podzemnim strukturama, radi dočaravanja okoline u kojoj nastaju špilje, uređen je izgled površine.

5.2.1. Oblik površine

Zbog brzine računanja i jednostavnosti prikaza, površina terena određena je visinskom mapom. Informacije o visini koriste se pri interpolaciji vrijednosti gustoće u točkama površinskih komada te se dobivena gustoća predaje algoritmu pokretne kocke kao i inače.

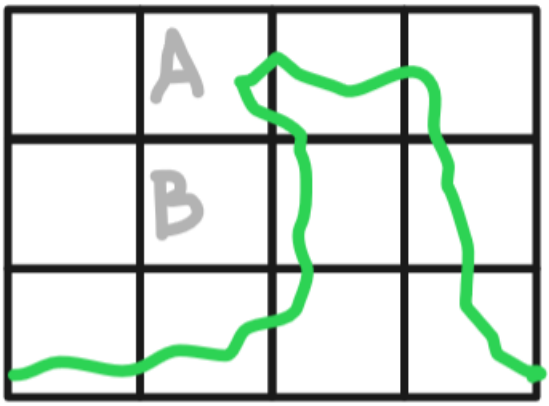
Kad bi se sve vrijednosti ispod granične visine postavile na gustoću terena, a sve iznad postavile na gustoću zraka, tada bi teren bio terasast jer bi algoritam pokretne kocke stavljao sve vrhove točno na polovinu stranice kocke. Iz tog razloga potrebno je interpolirati vrijednosti između minimalne i maksimalne tako da točka na graničnoj visini ima graničnu gustoću. Tehnika koju sam koristio svodi se na to da raspon u kojem se interpolira gustoća ispod i iznad granične visine budu jednako veliki, a sve točke izvan raspona imaju konstantnu gustoću koja odgovara ili terenu ili zraku. Ovom tehnikom dobije se glatki teren. Ideju prikazuje Slika 5.11.



Slika 5.11 Tehnika konverzije visinske mape u mapu gustoća

Prije stvaranja svakog novog vertikalnog komada površine prvo se provjeri je li u najvišem sloju prijašnjeg komada bio samo zrak. Ako je, tada iznad njega nema više terena pa novi vertikalni komad nije potrebno stvarati.

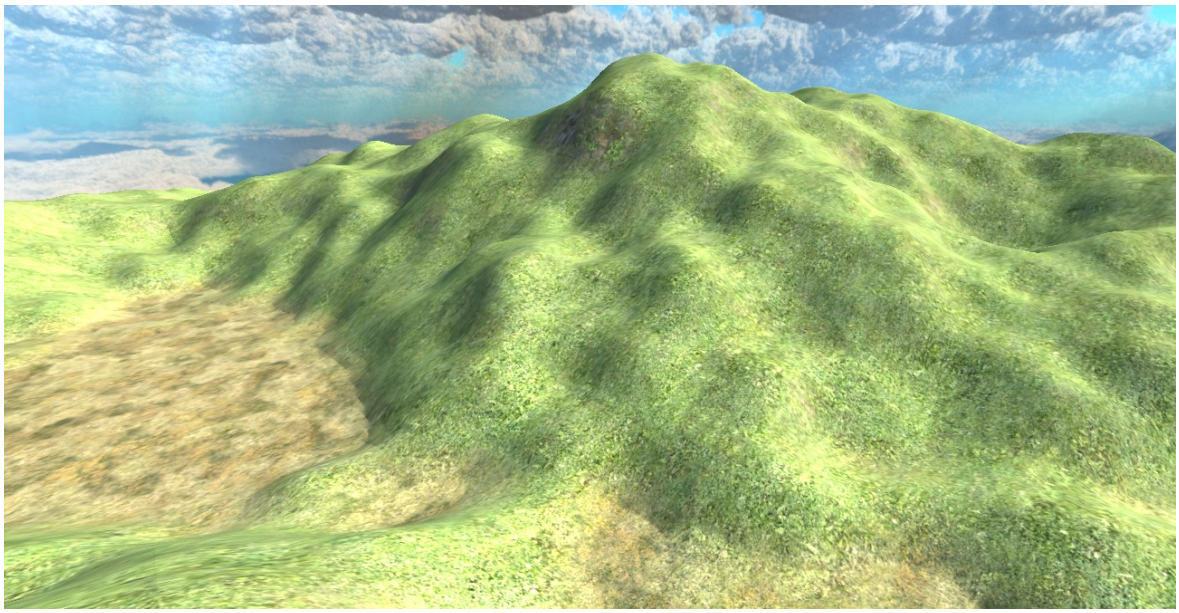
Ova optimizacija ne bi se mogla primjenjivati ako teren ne bi bio definiran visinskom mapom, jer bi mogli postojati prevjesi koji na većoj visini iz jednog komada prelaze u drugi, što prikazuje Slika 5.12. U komadu B nalazi se samo zrak pa ga nije potrebno generirati i prikazati, ali komad A bi trebalo stvoriti zbog prevjesa koji ulazi iz susjednog komada. Korištenjem visinske mape i spomenute optimizacije može se generirati teren s visokim ekstremnim vrijednostima bez da broj komada raste kvadratno.



Slika 5.12 Problem kod terena koji nije definiran visinskom mapom

5.2.2. Sjenčar površine

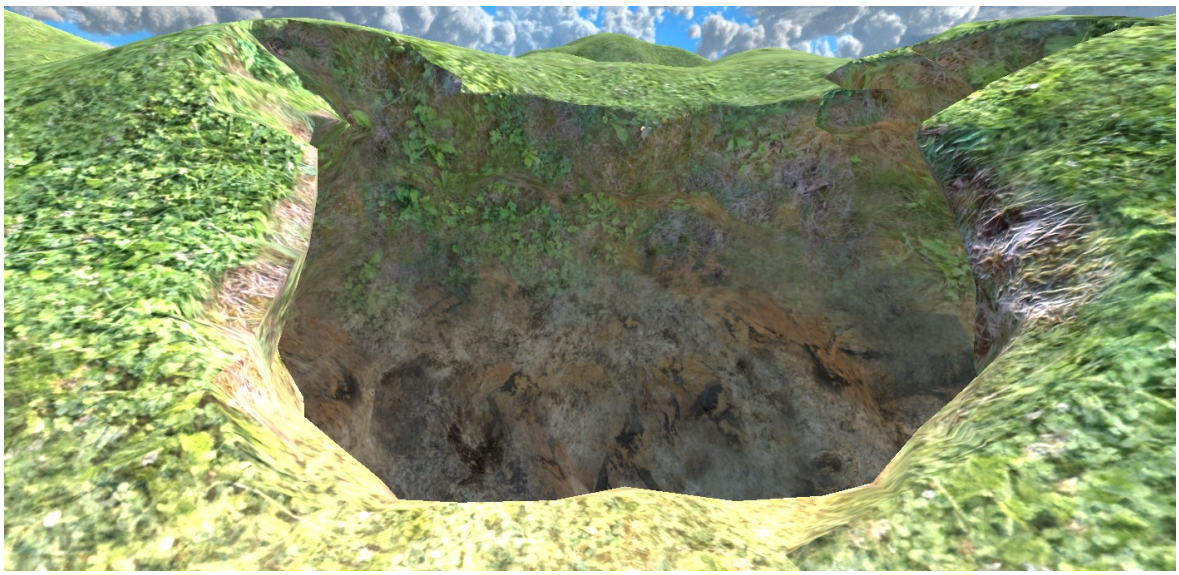
Za sjenčanje površine koristi se triplanarni sjenčar koji odozgo projicira teksturu trave, a sa strane projicira zemljaste teksture. Jedina modifikacija u odnosu na dosad korišteni sjenčar je to što se tekstura trave projicira samo odozgo dok se odozdo također projicira tekstura zemlje, kako trava ne bi rasla naopako s plafona. Izgled površine prikazuje Slika 5.13. Za razbijanje monotonije korištena je ista tehnika miješanja tekstura kao i u sjenčaru za špilje.



Slika 5.13 Teksturirana površina

Na prijelazu između površine i podzemlja interpoliraju se teksture zemlje i stijena kako bi prijelaz u špilju bio glađi. Interpolaciju tekstura na ulazu u špilju prikazuje Slika 5.14. S obzirom da su špilje često mokre, unutarnje površine su svjetlucave, a taj efekt ostvaren je

podešavanjem faktora glatkoće (engl. *Smoothness*). Teksture trave koje su korištene u radu preuzete su s reference [16].



Slika 5.14 Interpolirane teksture na ulazu u špilju

5.2.3. Osvjetljenje i magla

Špilje su prirodno većinom mračne, jer su zatvoreni prostori ispod zemlje. S obzirom da ovaj projekt ne koristi tehniku praćenja zrake, intenzivnije sjene mogu se ostvariti smanjivanjem ambijentalne komponente osvjetljenja. Osim realizma, mrak u špiljama dodaje efekt misterije i obogaćuje iskustvo istraživanja.

Jedan od problema koji se javio pri podešavanju osvjetljenja prikazuje Slika 5.15. A alatu Unity postoji ograničenje na udaljenost do koje se računaju sjene koje stvara interakcija svjetla i mreža objekata. S obzirom da špilje ponekad idu daleko u ravnoj liniji, korisnik može vidjeti svjetlo na kraju tunela koje nestane kad mu se približi.

Slika 5.15 Problem s osvjetljenjem

Opisani problem lako se da zakrpati povećanjem maksimalne udaljenosti sjena, iako to samo smanjuje vjerojatnost da će se problem primijetiti, a ne rješava ga u jezgri. S obzirom da projekt koristi *Universal Render Pipeline*, potrebno je zadati odgovarajući cjevovod u postavkama kvalitete te preko njega podešavati daljinu sjene.

Jednostavna tehnika koja je korištena u starijim videoigrama kako bi se prekrila ograničena udaljenost prikaza je korištenje magle (engl. *Fog*). U ovom projektu komadi se generiraju dinamički pa korisnik može lako primijetiti da neke stvari u daljini nedostaju. Srećom, u alatu Unity jednostavno je dodati efekt magle, jer već postoji ugrađen u postavkama rasvjete okoliša (engl. *Environment lighting*). Rezultate opisanih promjena prikazuje Slika 5.16.

Slika 5.16 Mračna i maglovita špilja

6. Višedretvena implementacija

Dosad su se svi izračuni izvršavali na glavnoj dretvi. To znači da kada se počnu generirati komadi terena i Perlinovi crvi, obrada korisnikovih unosa se zaledi dok ne završe svi izračuni. Projekt je izrađen na laptopu s procesorom Intel i5 9300H, 8 GB radne memorije te grafičkom karticom Nvidia GeForce GTX 1650. Rezultat je jako isprekidano izvršavanje programa koje dezorijentira korisnika i čini iskustvo korištenja alata manje ugodnim.

Glavni prioritet obrade trebali bi biti korisnikovi unosi. Generiranje komada može se dogoditi okvir (engl. *Frame*) prije ili kasnije te je puno manje primjetno nego kašnjenje s izračunima pokreta igrača. Idealno rješenje bilo bi da glavnoj dretvi druga dretva dojavi kada su izračuni gotovi pa se tek tada stvore novi komadi, a u međuvremenu bi se obrađivali unosi korisnika. U alatu Unity takvo rješenje može se implementirati vrlo elegantno korištenjem metoda za baratanje dretvama koje pruža biblioteka `System.Threading`.

Slijed događaja kod korištenja više dretvi prikazuje Slika 6.1. preuzeta iz videozapisa na referenci [17].

MapGenerator:

```
void RequestMapData(Action<MapData> callback) {
    // start MapData thread
}

void MapDataThread(Action<MapData> callback) {
    // get mapData
    // add mapData + callback to queue
}

void Update() {
    if (queue.Count > 0) {
        // call callback with mapData
    }
}
```

EndlessTerrain:

```
void Start() {
    RequestMapData(OnMapDataReceived);
}

void OnMapDataReceived(MapData mapData) {
    // do stuff with mapData
}
```

Slika 6.1 Korištenje paralelne dretve za izračun podataka o mapi [17]

Iz glavne dretve pozove se metoda kojom se zahtijevaju podatci o nekom komadu. Kao argument predaje se položaj komada i preostali potrebni parametri te povratna funkcija (engl. *Callback function*). Stvori se nova dretva na kojoj se računaju potrebni podatci, a za

to vrijeme glavna dretva obrađuje korisnikove unose. Kada paralelna dretva završi s izračunom, dodaje dobivene rezultate i povratnu funkciju u red (engl. *Queue*). U metodi `Update` na glavnoj dretvi provjerava se ima li ičega u redu te se poziva povratna funkcija s dobivenim podatcima. Razlog zašto se koristi red umjesto da se odmah pozove povratna funkcija je to što bi se inače pozvana funkcija također izvršila na paralelnoj dretvi. Nužno je da se izvrši na glavnoj dretvi jer se samo na njoj smiju stvarati nove instance objekata. Važno je napomenuti da se red treba zaključati pri dodavanju novog elementa, koristeći ključnu riječ `lock`, jer inače dolazi do sudaranja kada više dretava pokušava istovremeno dodati element.

Čak i nakon dodavanja više dretava, još uvijek su u programu postojali nagli skokovi u količini računanja, jer bi se svi komadi koji su dodani u red stvorili u jednom okviru. Iz tog razloga kôd je izmijenjen tako da se u svakom okviru stvori najviše jedan komad. Efekt je primjetan, jer se komadi pojavljuju na ekranu slijedno jedan po jedan, ali puno je ugodnije iskustvo korištenja alata jer nema naglog zastajkivanja.

Izvorni kôd projekta može se pronaći na referenci [18], a snimka izvođenja projekta na referenci [19].

Zaključak

Na početku ovog rada opisani su različiti tipovi špiljskih sustava te njihove karakteristike. Prikazani su različiti tipovi prolaza u špiljama i oblici špiljskih sustava. Odabrani tip špilje koji se istraživao u ostatku rada su krške špilje nastale otapanjem mekih stijena protokom vode, čiji prolazi su pretežno freatični, a sustavi razgranatog oblika. Duljina generiranih špilja je u desecima ili stotinu metara, a širine nekoliko metara.

Među metodama proceduralnog generiranja špilja spomenute su metoda 3D Perlinovog šuma, metoda grebenastog multifraktalnog šuma te metoda Perlinovog crva koja je korištena u ostatku rada jer lijepo imitira probijanje tunela vode kroz zemlju.

U svrhu uljepšavanja špilje i povećanja realizma opisana i implementirana je tehnika miješanja tekstura te triplanarna projekcija. Slične tehnike opisane su i za pridavanje tekstura površini, a za izradu sjenčara korišten je graf sjenčara u alatu Unity.

Iznesena je te implementirana ideja podjele terena na komade te dinamičko generiranje i uklanjanje neograničenog broja komada ovisno o poziciji igrača. Detaljno je prikazana interakcija Perlinovih crva s komadima terena.

U ostatku rada opisane su tehnike korištene za uljepšavanje terena i podešavanje parametara za oblih i veličinu špiljskih sustava. Za lakšu vizualizaciju oblika špilje izrađena je scena u kojoj se promjene nad parametrima automatski prikazuju u obliku špilje. Opisana je tehnika korištena kako bi segmenti špilje imali konkavne oblike.

Naposlijetku, dodani su površinski komadi koji predstavljaju planine. Objašnjeno je korištenje efekta magle i ambijentalnog osvjetljena kako bi špilje dobile mračan i misteriozan izgled. Spomenute su osnove višedretvene implementacije kojom se ostvaruje glatko izvođenje programa bez naglog zastajkivanja.

Moguća poboljšanja rada uključuju dodavanje podrške za dinamičko uređivanje terena od strane korisnika, definiranje površinskih komada koji nisu visinske mape, poboljšavanje sudarača u špiljama kako igrač ne bi prolazio kroz zidove, dodavanje podrške za praćenje zrake zbog realističnih sjena u špiljama te optimizacija algoritama korištenjem paralelizma.

Literatura

[1] Lava tube, https://en.wikipedia.org/wiki/Lava_tube; pristupljeno 21. travnja 2022.

[2] Karst landforms, https://www.gsi.ie/en-ie/programmes-and-projects/groundwater/activities/understanding-irish-karst/karst-landforms/Pages/default.aspx; pristupljeno 21. travnja 2022.

[3] Hang Sơn Đoòng, https://en.wikipedia.org/wiki/Hang_S%C6%A1n_%C4%90o%C3%B2ng; pristupljeno 21. travnja 2022.

[4] Types of Cave Passages: Advanced illustrated guide, https://startcaving.com/caving-guides/passages; pristupljeno 24.4.2022.

[5] Speleothem, https://en.wikipedia.org/wiki/Speleothem; pristupljeno 24.4.2022.

[6] Danijel Bajlo, Proceduralno generiranje terena algoritmom pokretne kocke (2020), str. 7, http://www.zemris.fer.hr/predmeti/irg/Zavrsni/20_Bajlo/Final_0036505511_41.pdf; pristupljeno 24.4.2022.

[7] Game developer magazine, Creator of worlds, str. 21, https://ubm-twvideo01.s3.amazonaws.com/o1/vault/GD_Mag_Archives/GDM_April_2011.pdf; pristupljeno 24.4.2022.

[8] What is ridged multifractal noise?, https://it-qa.com/what-is-ridged-multifractal-noise/; pristupljeno 24.4.2022.

[9] Auburn, FastNoise Lite, https://github.com/Auburn/FastNoiseLite; pristupljeno 25.3.2022.

[10] Perlin worms, http://libnoise.sourceforge.net/examples/worms/; pristupljeno 24.4.2022.

[11] Jenny Pederson, Stalagmite and Stalactite (Mnemonic) - How To Tell Apart, https://www.pinterest.com/pin/259238522270509076/; pristupljeno 24.4.2022.

[12] Ben Cloward, World-Aligned Textures - UE4 Materials 101 - Episode 22 (2020), https://youtu.be/pXOknekvmwE; pristupljeno 24.4.2022.

[13] Texture Haven, Rock Textures - 4K (2020), https://assetstore.unity.com/packages/2d/textures-materials/rock-textures-4k-179128; pristupljeno 17.4.2022.

[14] Ben Cloward, Triplanar Projection - Shader Graph Basics - Episode 28 (2021), https://youtu.be/sjpszGetM40; pristupljeno 17.4.2022.

[15] Avionix, Skybox series free, https://assetstore.unity.com/packages/2d/textures-materials/sky/skybox-series-free-103633; pristupljeno 18.4.2022.

[16] VIS Games, VIS – PBR Grass Textures, https://assetstore.unity.com/packages/2d/textures-materials/floors/vis-pbr-grass-textures-198071; pristupljeno 30.5.2022.

[17] Sebastian Lague, Procedural Landmass Generation (E08: Threading), https://www.youtube.com/watch?v=f0m73RsBik4; pristupljeno 20.6.2022.

[18] Danijel Bajlo, Masters project, https://gitlab.com/dBajlo/masters-project; pristupljeno 20.6.2022.

[19] Danijel Bajlo, Procedural Cave generation, https://www.youtube.com/watch?v=WlLKdd6jOBk; pristupljeno 22.6.2022.

[20] List of longest caves, https://en.wikipedia.org/wiki/List_of_longest_caves; 23.6.2022.

[21] List of deepest caves, https://en.wikipedia.org/wiki/List_of_deepest_caves; 23.6.2022.

Sažetak

Proceduralno generiranje špiljskog sustava

Ovaj rad opisuje primjenu metode Perlinovog crva u svrhu dinamičkog generiranja krških špiljskih sustava u neograničenom terenu podijeljenom na komade. Opisane su različite vrste i karakteristike špilja te navedene alternativne metode generiranja. Miješanje i projekcija tekstura na teren implementirani su koristeći graf sjenčara u sklopu programskog alata Unity. Opisana je tehnika dinamičkog generiranja komada terena te njihova interakcija s Perlinovim crvima. Ilustrirana je metoda korištena za dobivanje konkavnih segmenata u špiljama te algoritam za određivanje karakteristika grananja špilje. Spomenuto je osvjetljenje i efekt magle koji doprinose stvaranju atmosfere u špiljama te prekrivaju ograničeni doseg generiranih komada. Na kraju su veliki izračuni izdvojeni na druge dretve kako bi se ostvarilo glatko izvođenje programa na glavnoj dretvi.

Ključne riječi: špilje, pećine, proceduralno generiranje, Unity, Perlinov crv, krški reljef, graf sjenčara, algoritam pokretne kocke

Summary

Procedural Generation of Cave System

This thesis describes the application of Perlin worm method for the purpose of dynamically generating karst cave systems as a part of a boundless terrain divided into chunks. Different types of caves and their characteristics are examined along with alternative methods of generation. The mixing and projection of textures onto the terrain were implemented using shader graph built into Unity game engine. The dynamic generation of terrain chunks and their interaction with Perlin worms is described. The thesis illustrates the method used for generating concave cave segments and the algorithm for determining cave branching characteristics. Lighting and fog effects are used to breathe atmosphere into the caves while also covering up render distance limitations. Finally, big calculations were sent to separate threads for smooth execution of the program on the main thread.

Keywords: caves, procedural generation, Unity, Perlin worm, karst relief, shader graph, marching cubes algorithm