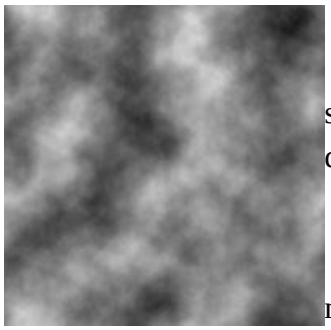
Proceduralno generisanje terena

Predlog projekta iz predmeta Numericki algoritami i numericki softver

Height Mape

Kod generacije terena pomocu Height Mape se uzmima da svaki piksel ima boju, ako se radi u RGB formatu za skorz beo piksel koji sadrzi mesavinu svih broja vrednost je jednaka 256x256x256 sto je i maksimalna moguca vrednost, a skroz crni ima vrednost 0x0x0 ovaj interaval se moze iskoristi tako sto se za svaku tacku generisanog terena koja odgovara poziciji piksela slike dodeli vrednost visine tog piksela. Naravno ovaj interval se moze skalirati da nam maksimalna visine ne bude 256x256x256. Skaliranje se obicno uvek vrsi na vrednosti od -1 do 1 koje se kasnije mnoze sa maximalnom visinom. -1 jer se predvidja da postoji neka voda ili zamisljena granica kojo ogranicava normalnu visinu, u radu ce to biti standardno 0 i na njoj ce biti voda.

Ovakva generacija terena preko slike nije uvek pogodna jer prelazi piksela nisu uvek blagi i ne prate ono sto bi se nazvalo prirodnim. Zato se prave specijalne slike koje se zovu HeightMape, a medju popularnijima je najcesca upotereba Perlin noisa za HeightMape.



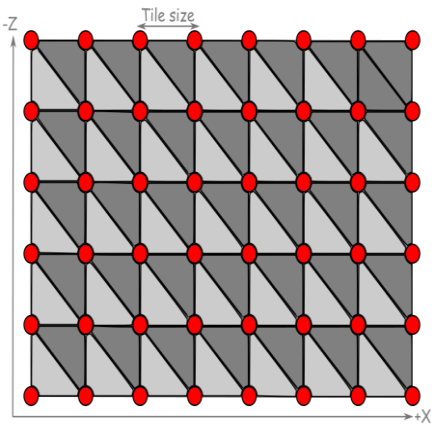
Primer Perlin noise. Bitne cinjenice su da su prelazi izmedju najtamnijeg piksela i najsvetlijeg blagi, ali sama funkcija nije po sebi glatka jer sadrzi puno manjih variranja vrednosti, ovo je bitno jer daje na prirodnosti izgleda generisanog terena.

Celo nase bavljenje HeightMapama i Perlin noisom jeste zato sto cemo mi proceduralno pokusati da izmitiramo ovu funkciju. To moze da se postigne na par nacina, dalje je opisan korisceni nacin.

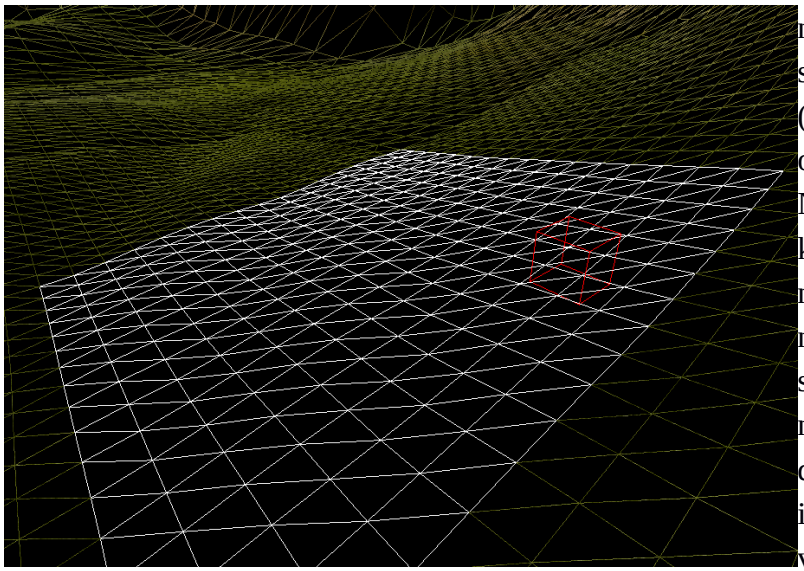
Proceduralna generacija

Pocetak naseg generisanja ce se zasnivati na nasumicnosti ili mozda bolje receno na pseudo nasumicnosti (pseudo jer cemo vrsiti vise korekcija na tome sto je nasumicno, a i nista nije stvarno nasumicno). Da bi objasnili pojam pseudo nasumicnosti prvo cemo se vratiti na neke osnove 3D prikaza i programiranja. Dakle mi imamo 3 ravni od kojih smo generisali nesto u ravnima X i Z ravanima, Y je visina i na njoj je 0. Principi generisanja se zasnivaju na trouglovima odnosno svako telo je generisano iz gomile malih trouglica.

Zasto trouglovi? Pa zato sto trouglove je nemoguce povezati drugacije, odnosno kad su tacke trougla povezane, povezane su na jedini moguci nacin. To nam je bitno kasnije za mapiranje istih tacaka jer jer se nase tacke preklapaju (u prevodu svaka dva susedna trougla imaju najmanje 2 iste tacke ovo kasnije ima uticaja i kod generacije visine).

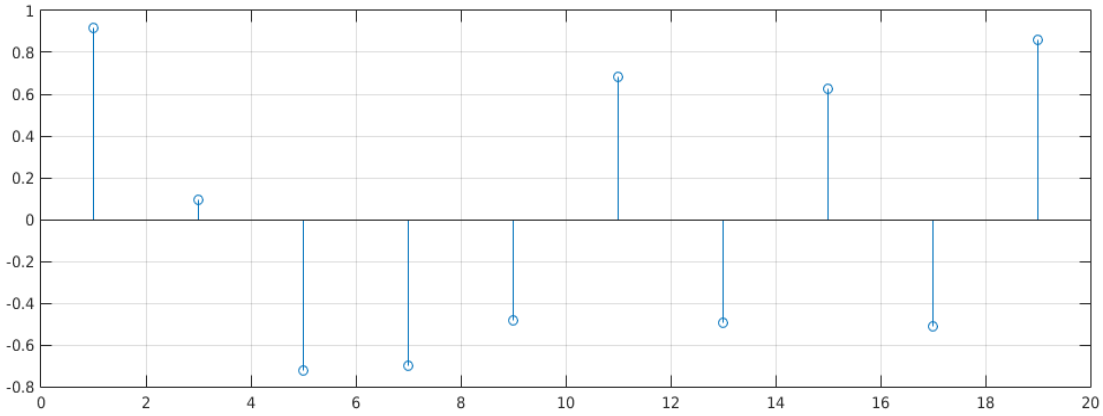


Primer koordinatnog sistema predstavljaju gore receno u praksi. Ovakava mreza se generise bez vecih problema sa jendakim korakom za svaku tacku, ali tu nailazimo na prvu stvar zbog koje



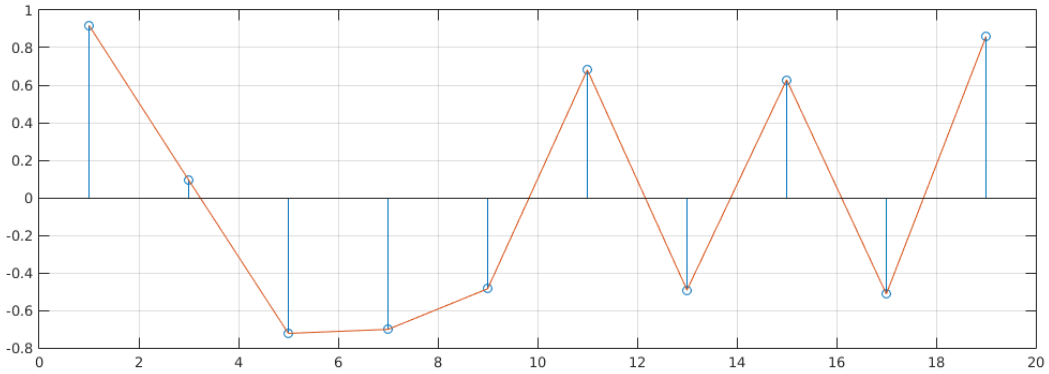
nije proces nasumican, svaki ovakam sistem se generise preko semena (seed), sto je jako bitno zbog optimizacije kasnije i renderovanja . Naime ako generisemo samo teren koji je u fokusu onda kad bi se vratili na pocetnu tacku bio bi izgenerisan neki skroz drugaciji teren sto nije ono sto zelimo. Te uvodimo u generisanje nasumicnih brojeva seme. Odnosno za dva ista broja nasumicni generator sa istim semenom uvek vraca istu vrednost. Primer igre koja koristi ovo je minecraft. Znaci kada je prvi put nasumicni (random) generator naisao na broj 3 i recimo izgenerisao broj 51 svaki sledeci put kad dobije broj 3 on mora da generise broj 51. Druga stvar koja nam smanjuje nasumicnost jeste nase ogranicenje intervala u kojem se generisu nasumicni brojevi. A treca stvar je to sto kod nasumicnog generisanja javlja se isti problem da dve susedne kordinate (recimo (x,y) i (x+1,y)) mogu da imaju ogroman skok (npr. -97, i 100) sto nije prirodno, te da bih to smanjio koristim u kodu osluskivanje okoline (ili ti neki filter) odnosno tacke okoline imaju uticaj na vrednost kordinate visine. Npr. uzimamo da samo okolne tacke uticu na dobijanje visi nase tacke tako da imamo 8 tacaka koje okruzuju nasu. Tacke u coskovima imaju recimo uticaj 1/16 , tacke iznad, levo, desno i ispod 1/8 , a nasa nasumicno generisana tacka (centralna) utice sa 1/4. Ako se saberu ove verovatnoce zbir tih uticaja bi trebao da bude 1 ukupno.

Princip nasumicne generacije prikazani u 2D-u

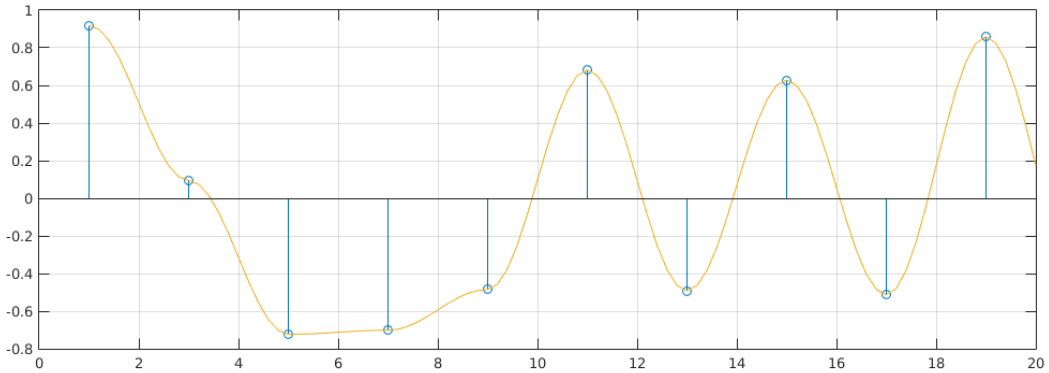


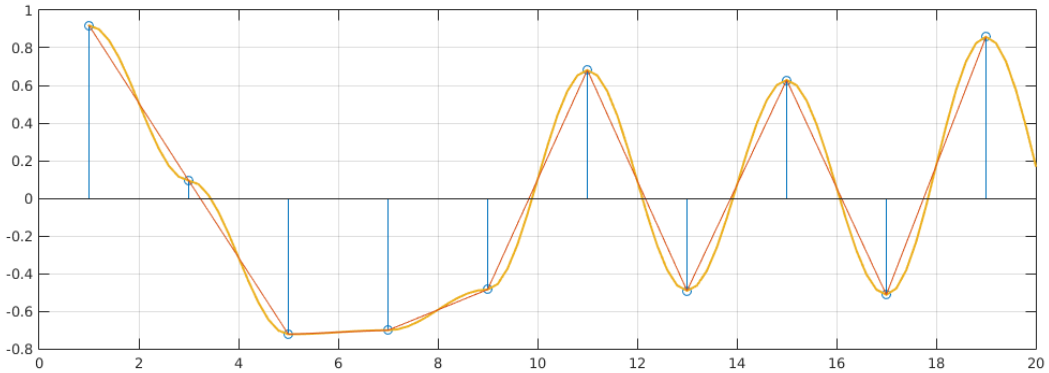
Prvo se generisu nasuminci projevi od -1 do 1 za svaku tacku.

U ovom primeru nije koriscenjo osluskivanje okoline, ali u kodu ce biti.
Sledi korak interpolacije. Ovaj korak se moze vrsiti na milion nacina. ispod je dat linearna.

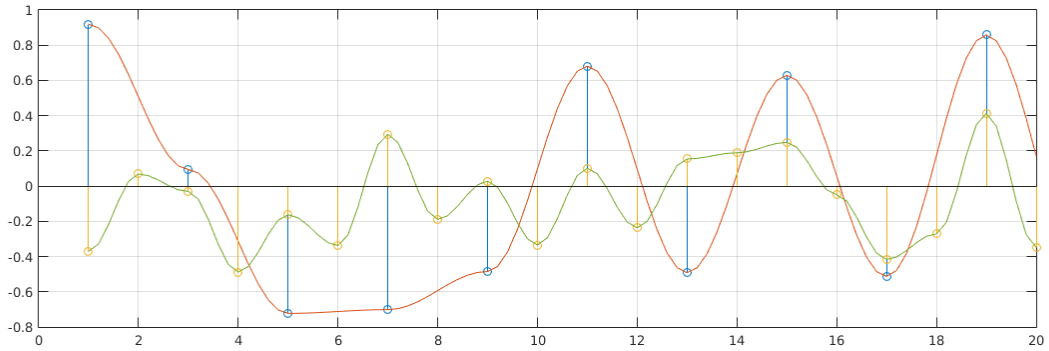


Kao sto se da primetiti ovo nije bas prirodno te cemo pokusati npr. sa kosinusnom iterpolacijom gde dobijamo i krivine. Sve interpolacije su kroz 2 tacke, moguce je implementirati i kroz vise tacaka.

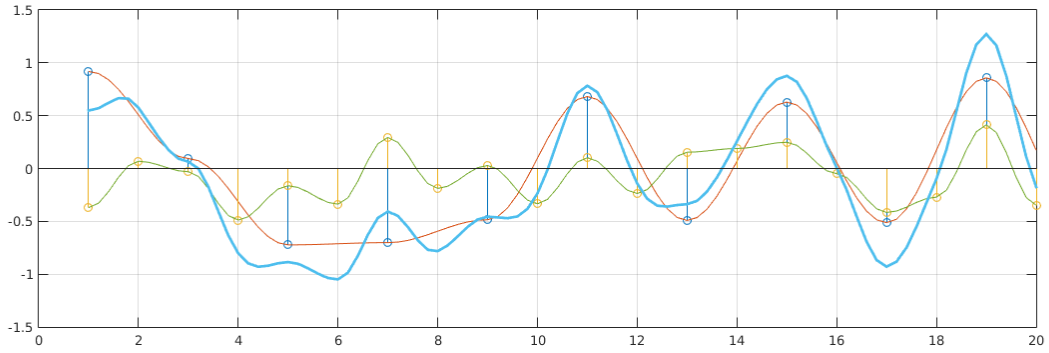


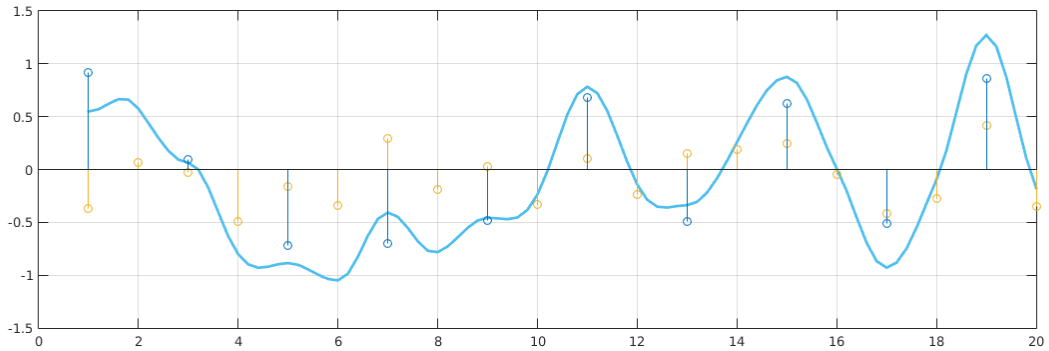


iznad data je bolja kosinusna iterpolacija i uporedo linearnom interpolacija. No idalje ovo nije bas prirodno te se pribegava sabiranju funkcija, da bi se dobile sitne promene funkcije koje su prirodne. To se vrsi tako sto sje frekvencija funkcije poveca ali se njena amplituda smanji.

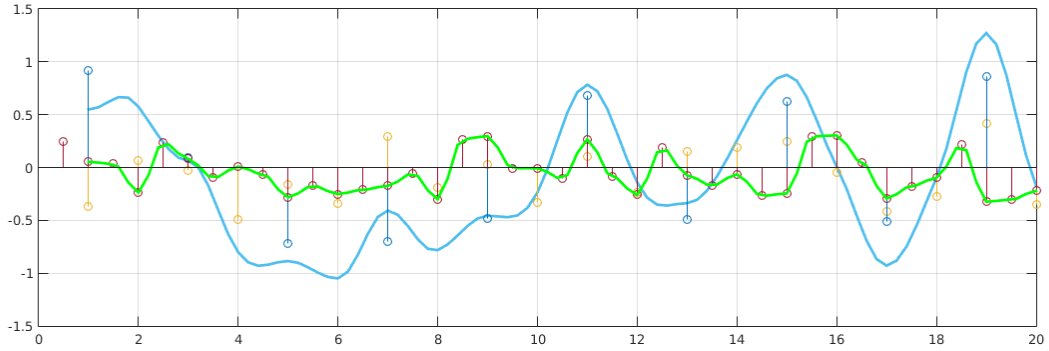


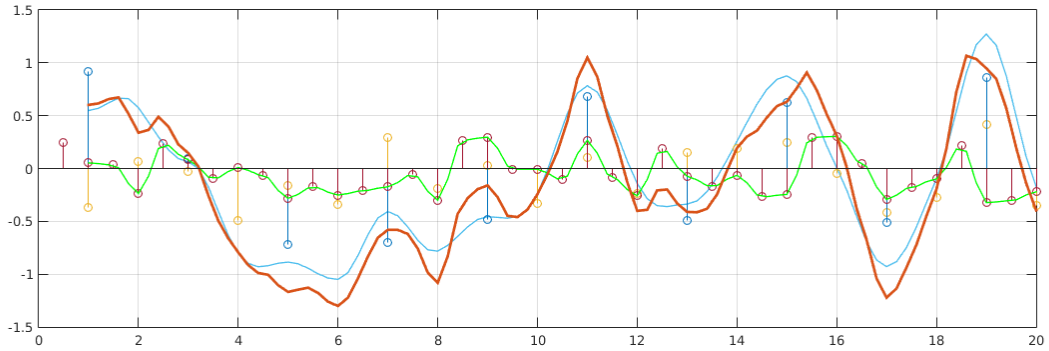
Gore je prikaz dotate funkcija sa duplo vecom frekvencijom i duplo manjom amplitudom. Ovo inace treba da su iste funkcije (na slici nisu). Dalje se saberu te dve funkcije i dobija se:





Vec se dobija malo realnija slika terena sada ovaj postupak se ponavlja dok nismo zadovoljni.

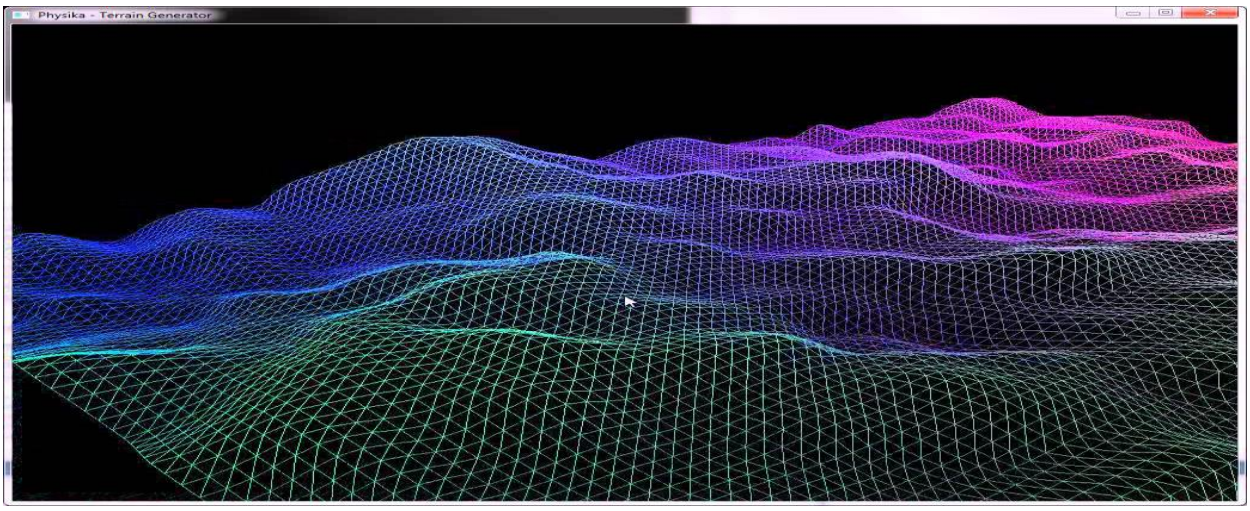


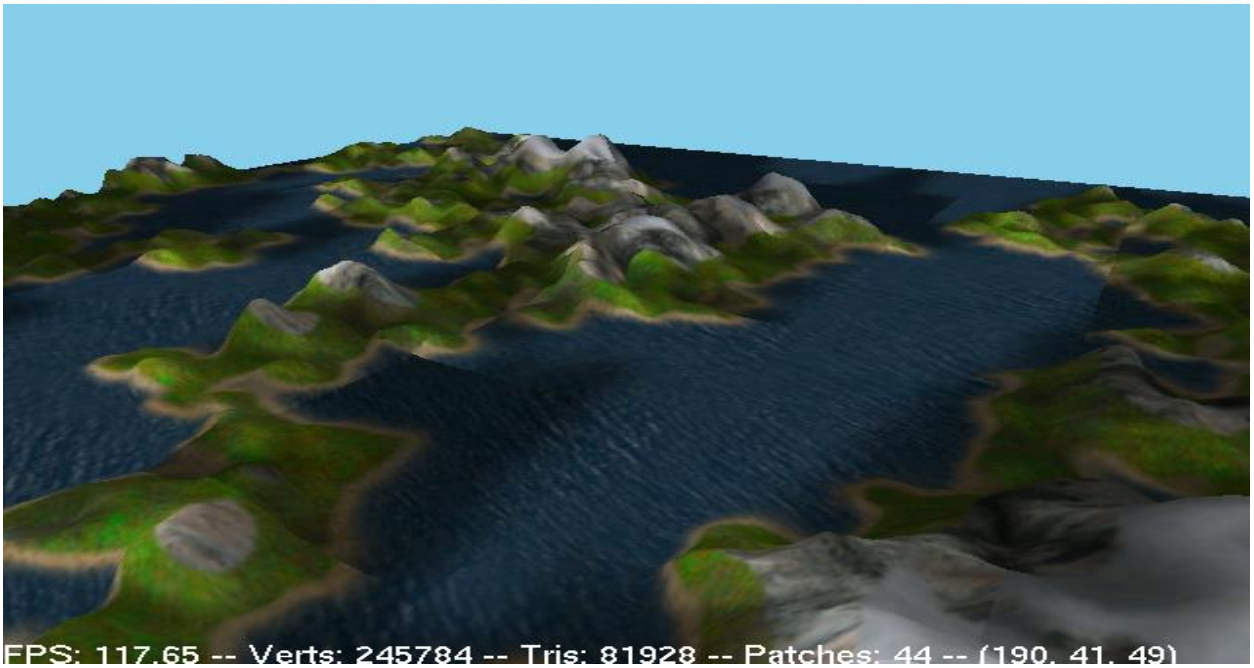


Uzeti u obzir da ovde nema osluskivanja okolike i da nije koriscena ista funkcija,no to je princip nasumicnog generisanja (Perlin noise) suma i proceduralnog terena
Bitno za izvuci odavde jeste to da povecanjem amplitude dobijamo detaljnost funkcije, ali ne zelimo da nam ta detaljnost kvari nas prvobitni oblik, te smanjujemo njen uticaj na globalni oblik tako sto joj smanjujemo amplitudu. Ove korake mozemo ponavljati vise puta.
Sada za razne nacine interpolacije konsultovati sledeci sajt odakle su uzete neke varijante interpolacije izmedju 2 tacke: https://codeplea.com/simple-interpolation

Specifikacija alata

Kod ce biti u jeziku java (Java) sa koriscenjem biblioteka LWJGL (Light Weight Java Game Library) koja koristi OpenGL API. Koristi se LWJGL 2.9.1 iako postoji novija verzija,zato sto ima najvise podrske na netu i usput ce se konsultovati tutorijali pravljenja game engine od scratch-a jednog programera. U kodu su implementirani i neke njegove metode za shadere (Vertex shader i Fragment shader - koji umnogome upotpunjuju izgle generisanog terena), takodje plan je da se ubaci i voda (sa njenim osobinama refrakcije i reflaksije ali tema ovog rada nece biti kako funkcionise ta voda i nece biti objasnjeno). Grafika ce se baziranti na njegovom kodu. On se takodje bavio malo i sa generacijom terena. Sav deo oko generacije terena bice obradjen u kodu i podrazumeva se u odbrani projekta. Link njegovog kanala:
https://www.youtube.com/channel/UCUkRj4qoT1bsWpE_C8lZYoQ
Takodje pored LWJGL biblioteke koristi se i Slick-Utils biblioteku kao i PNG-Decoder koji je potreban za implementaciju SkyBoxa.
Sto se tice menija koji je zadat u specifikaciji, napravice se napor da bude pokriveno sto vise parametara koji su bitnim: tipa izbor interpolacione metode (bice implementirana step, linear, cosine, smooth step...), izbor nacina generisanja terena (preko HeightMape ili proceduralno), izbor broja funkcija koje ce se sabirati i parametar povecanja frekvencije i parametar smanjenja amplitude no ne garantuje se da ce to biti vidljivo u real time-u. Shaderi su kompleksna stvar uopse za samu generaciju terena, veliki broj proracuna tacaka terena, vode, objekata, gde za svaki objekat mora da se uzmu pozicija izvora svetlosti i normale da bi se pravile senke, refleksije, refrakcije i uopste pozadinske interpolacije za boje i teksture koje vrsi GPU nisu nimalno naivne, zbog kojih mozda nece biti moguce vrsiti promene u real time-u, tu se jos dodaje ugao kamere sa kojim kamera gleda na nesto. Primer ocekivanog rezultata:


Physika - Terrain Generator


FPS: 117.65 -- Verts: 245784 -- Tris: 81928 -- Patches: 44 -- (190, 41, 49)