SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5364

Prikaz prirodnih okoliša

Tin Srnić

Zagreb, lipanj 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 9. ožujka 2018.

ZAVRŠNI ZADATAK br. 5364

Pristupnik: **Tin Srnić (0036491730)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Prikaz prirodnih okoliša**

Opis zadatka:

Proučiti različite elemente koji čine prirodni okoliš. Razraditi komponiranje pojedinih prirodnih elemenata kao što je vegetacija, voda i nebo kako bi ostvarili cjelokupan dojam prirodnog okoliša. Proučiti grafički standard OpenGL inačice 3.x na više. Razraditi aplikaciju koja će omogućiti realističan prikaz elemenata koji čine cjelokupan okoliš upotrebom proučenih tehnologija. Na različitim primjerima prikazati ostvarene rezultate. Načiniti ocjenu rezultata i implementiranih postupaka.
Izraditi odgovarajući programski proizvod. Koristiti programski jezik C++ i grafičko programsko sučelje OpenGL. 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.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 15. lipnja 2018.

Mentor:
Prof. dr. sc. Željka Mihajlović

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Siniša Srbljić

Dielovođa:
Doc. dr. sc. Tomislav Hrkać

Tablica sadržaja

1. Uvod

Glavni cilj računalne grafike je stvaranje i prikazivanje realističnih virtualnih scena. Postoje brojne tehnike koje se koriste pri stvaranju takvih scena. Ovaj rad se usredotočuje na vrste podataka i tehnika korištenih u programu za prikaz realističnih 3D scena u realnom vremenu.

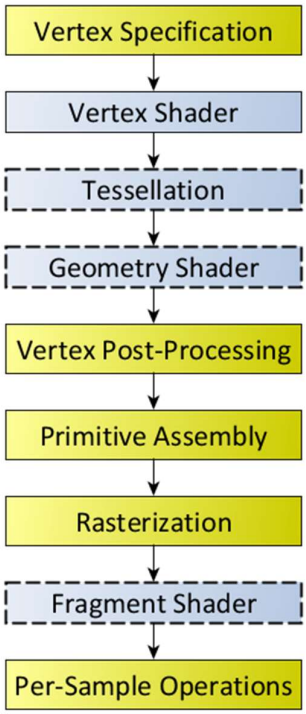
Obrađivat će se jednostavne tehnike koje se koriste za prikaz raznih elemenata koje se sreću u video igrama. Program koji je razvijen uz ovaj rad implementira tehnike koje su objašnjene u nastavku, no glavni cilj takvog programa je njegovo stalno poboljšavanje i razvijanje. Također će se obraditi način spremanja podataka i vrste podataka potrebne za prikaz raznih elemenata okoliša.

2. Programsko sučelje OpenGL

Open Graphics Library (OpenGL) je aplikacijsko programsko sučelje (API) koji služi za crtanje 3D i 2D objekata [1]. Namijenjen je da se koristi uz grafički procesor (GPU) kako bi se pomoću sklopovske podrške postiglo brže iscrtavanje na ekran.

2.1. Programi za sjenčanje

Cjevovod modernog OpenGLa se bazira na programima za sjenčanje (engl. shader) koji se pišu za izvođenje na GPU. U OpenGLu postoje 4 vrste sjenčara: sjenčar vrhova (engl. vertex shader), sjenčar teselacije (engl. tessellation shader), sjenčar geometrije (engl. geometry shader) i sjenčar fragmenata (engl. fragment shader). Ulaz u sjenčar vrhova zadaje korisnik, a izlaz iz jednog sjenčara je ulaz drugog kako je definirano standardom OpenGLovog cijevovoda [2].



Slika 1. Prikaz protočnog sustava OpenGLa

Na Slici 1. plave kućice su programibilni dijelovi u protočnom sustavu i odgovaraju svakom od četiri sjenčara koje je moguće koristiti. Izlaz sjenčara vrhova je ulaz sjenčara teselacije, izlaz sjenčara teselacije je ulaz sjenčara geometrije, a izlaz sjenčara geometrije

je ulaz sjenčara fragmenata.

Ovaj rad će ignorirati sjenčar teselacije i sjenčar geometrije jer se nisu koristili u programu, ali ću kratko spomenuti njihove moguće primjene.

Sjenčar vrhova je programibilna faza u cjevovodu za prikazivanje koja obrađuje pojedine vrhove poligona. U ovoj fazi se primjenjuju transformacije na vrhove koje uključuju translaciju, rotaciju, skaliranje i perspektivnu transformaciju.

Sjenčar teselacije služi da podijeli poligon na više manjih poligona temeljeno na podacima o vrhovima izvornog poligona. Proces uključuje podjelu poligona na više poligona, a zatim izračunavanje novih vršnih vrijednosti (položaj, boja, tekstura, itd.) za svaki od vrhova generiranih ovim procesom. Često se koristi za izglađivanje terena ili drugih objekata koje je poželjno zadati s manjim brojem vrhova te ih potom povećati.

Shader geometrije kao ulaz dobiva niz točaka slično kao i sjenčar vrhova, ali na mjestu dobivenih točaka crta zadanu geometriju odnosno zadanu vrstu poligona. Primjenjuje se u sustavima čestica gdje postoji puno čestica s jednostavnim oblikom. Moguće je svesti cijeli poligon na samo jednu točku oko koje sjenčar geometrije može nacrtati poligon.

Sjenčar fragmenata kao ulaz dobiva izlaz iz sjenčara vrhova, odnosno potencijalno iz sjenčara geometrije ukoliko se koristi. Tada za svaki poligon nacrtan u sceni iterira po njegovim fragmentima i ispunjava ih bojom.

2.2. Strukture podataka u OpenGLu

Kako bi se omogućio rad sa sjenčarima mora se inicijalizirati sjenčar program koji sadrži sve željene vrste sjenčara [3]. U programu su to uvijek sjenčar vrhova i sjenčar fragmenata. Kada je program inicijaliziran on dobiva svoj ID kojim se može postaviti kao trenutno aktivni sjenčar.

Kako bi se aktivnom sjenčaru poslali podaci, oni moraju biti zapakirani u objekt polja vrhova (engl. vertex array object, VAO).

VAO je objekt u koji je moguće pohraniti podatke o 3D modelima. Svaki VAO ima niz lista atributa (engl. attribute list) koji se koriste za pohranjivanje različitih vrsta podataka o 3D modelu. Broj mogućih lista ovisi o arhitekturi računala, no najčešće je ograničen na 16.

Ti se skupovi podataka pohranjuju kao objekti međuspremnika vrhova (engl vertex buffer object, VBO).
VBO je podatkovni međuspremnik koji sadrži podatke o 3D modelu. Svaki VBO pohranjen je u jednoj atributnoj listi VAOa te kada se pristupa podacima VAOa u sjenčare se učitavaju podaci svih VBOa vezanih na taj VAO.

Objekt međuspremnika slika (engl. Frame buffer object, FBO) je struktura podataka koja se može shvaćati kao dinamična slika. Dinamična je jer je namijenjena za često pisanje i brisanje, drugim riječima to je spremnik u koji možemo crtati pomoću OpenGLa bez da crtamo na glavni ekran. Korisna je za neke efekte koje je jedino moguće postići višestrukim crtanjem scene poput efekata vode, sjena ili sličnog. Također nužna je za ostvarenje bilo kakvih naknadnih vizualnih učinaka (engl. post-processing) poput zamagljivanja ekrana ili povećanja kontrasta.

3. Elementi scene

3.1. Entiteti

Entitet je veoma širok pojam, ali u alatu je to osnovna jedinica za prikaz u prostoru koji je prikazan perspektivnim matricama, odnosno prostor koji ima dubinu. Sam prikaz entiteta se svodi na osvjetljenje, bacanje sjena i druge tehnike koje su objašnjene kasnije. Budući da su strukture podataka veoma slične i dijele ih gotovo sve vrste objekata, u ovom poglavlju će na konkretnom primjeru entiteta biti objašnjeno koje vrste podataka se koriste i kako su relevantne.

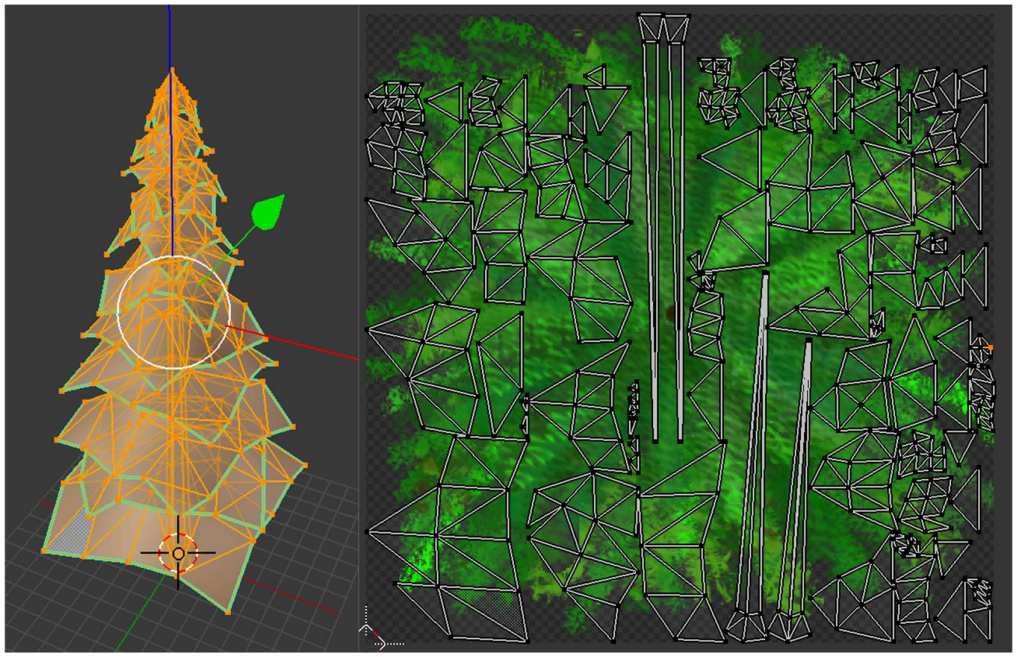
Instanca entiteta sadrži dvije glavne komponente: mrežu poligona (engl. mesh) i teksturu.

Mreža poligona je skupina podataka o vrhovima entiteta i uključuje tri različite vrste podataka koji su svi pohranjeni u jedan VAO. Podaci uključuju poziciju vrhova, vektor normale u vrhu i koordinate tekstura, svaki od kojih je pohranjen u vlastiti VBO.

Program je izrađen za rad s modelima koji su sačinjeni od niza trokuta od kojih svaki ima tri vrha, a svaki vrh ima tri prostorne koordinate x, y i z. Pozicije vrhova su glavna komponenta bilo kojeg sustava za crtanje jer se pomoću njih određuje gdje na ekranu će se objekt pojaviti. Dakle kako bi predstavili 3D model u računalu moramo pohraniti njegove vrhove. Stavljajući prostorne koordinate svakog vrha u VBO možemo konstruirati jednostavan prikaz entiteta.

Normale entiteta služe za izračune osvjetljenja nad modelom. Slično kao i pozicije, postoji uvijek jedna normala u svakom vrhu koja je predstavljena kao trodimenzionalni vektor. Stoga možemo primijeniti identičan pristup i u drugi VBO staviti x, y i z koordinate svake normale, pazeći pri tome da poredak vrhova odgovara onome u VBOu koji sadrži pozicije.

Zadnja vrsta podataka su koordinate tekstura. Ukoliko entitet želimo prikazati s teksturom, nije dovoljno imati samo sliku teksture već i podatke koji opisuju kako se tekstura treba primijeniti na entitet. Potrebno je svaki trokut koji čini entitet preslikati na dio slike.



Slika 2. (lijevo) prikaz modela, (desno) prikaz preslikavanja poligona na sliku

Zapravo ono što činimo je da preslikavamo koordinate x, y, z trokuta modela u novi koordinatni sustav slike (Slika 2.). Osi novog koordinatnog sustava se najčešće označavaju sa UV i u rasponu su [0, 1]x[0, 1]. Rezultat transformacije je dvodimenzionalni vektor kojeg možemo smjestiti u novi VBO.

Kada imamo sva tri VBOa spremna možemo konstruirati VAO koji će objediniti sve naše podatke. U VAO tada vežemo tri VBOa i možemo ga pozvati pomoću ID kojeg mu dodijeli OpenGL.

```
GLuint vertBufferID, uvBufferID, normBufferID;

glGenVertexArrays(1, &m_VaoID);
glBindVertexArray(m_VaoID);

glGenBuffers(1, &vertBufferID);
glGenBuffers(1, &uvBufferID);
glGenBuffers(1, &normBufferID);
glGenBuffers(1, &m_IboID);

StoreDataInAttribArray(vertBufferID, t_vertices, 3 * t_vertexCount * sizeof(GLfloat), VERTEX_LOCATION, 3);
StoreDataInAttribArray(uvBufferID, t_UVs, 2 * t_vertexCount * sizeof(GLfloat), UV_LOCATION, 2);
StoreDataInAttribArray(normBufferID, t_normals, 3 * t_vertexCount * sizeof(GLfloat), NORMAL_LOCATION, 3);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_IboID);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, t_indicesCount * sizeof(GLuint), t_indices, GL_STATIC_DRAW);

glBindVertexArray(0);
```

Izlist iz programskog koda 1. Konsturkcija VBO i VAO objekata

Teksture u OpenGLu mogu biti različitih tipova poput 1D, 2D ili 3D tekstura i drugih. OpenGL podržava i učitavanje različitih formata teksture poput rgb, bgr, rgba i drugih. Entitet koristi jednostavnu 2D teksturu. Pri inicijalizaciji teksture OpenGL joj pridjeljuje ID koji entitet kasnije može koristiti. Inicijalizacija teksture se svodi na zadavanje tipa teksture (2D teksture) , zadavanje formata teksture i predavanje bita tekstura OpenGLu.

```
glGenTextures(1, &gl_texID);

glBindTexture(GL_TEXTURE_2D, gl_texID);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_BGRA, GL_UNSIGNED_BYTE, bits);
```

Izlist iz programskog koda 2. Konstrukcija OpenGL teksture

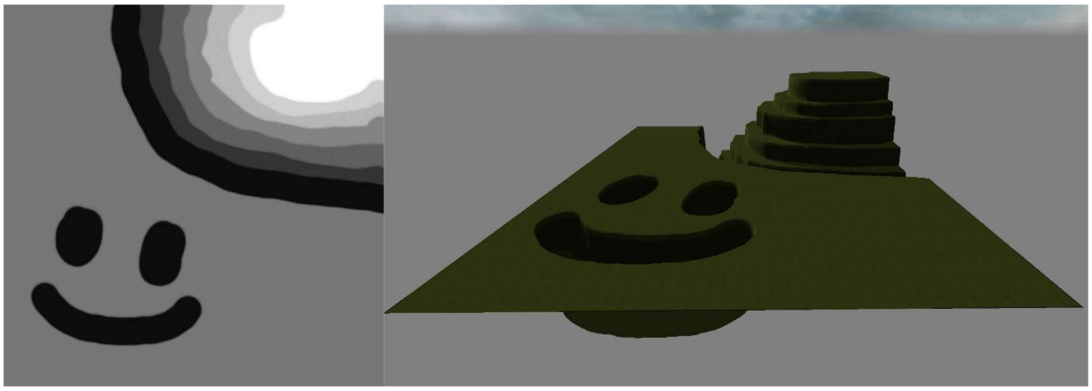
Kada je tekstura učitana u sjenčar njenim pikselima možemo pristupiti pomoću 2D UV koordinata.

3.2. Teren

Generator terena podržava dva načina generiranja terena. Prvi je način korištenje crno-bijele slike koja predstavlja visinsku mapu, a drugi je proceduralno generiranje terena.

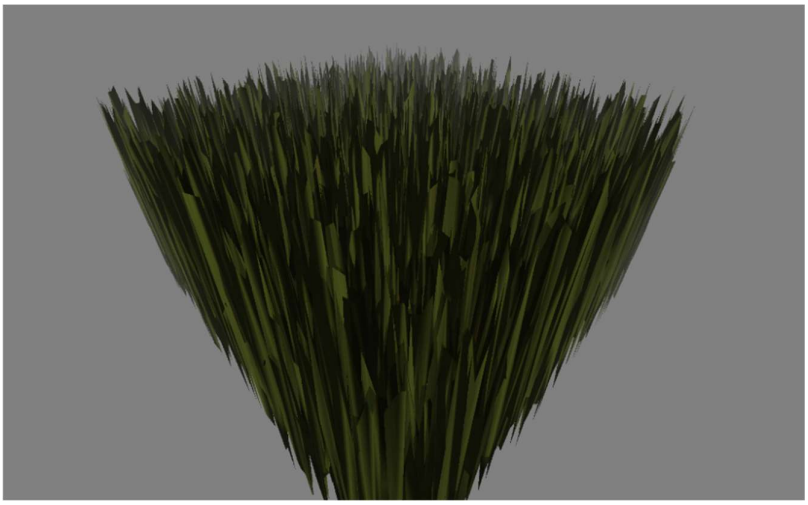
Visinska mapa je crno-bijela slika čiji pikseli predstavljaju željenu visinu terena. Mapa mora biti iste rezolucije kao željeni teren. Ukoliko se uzme slika u boji, u ovoj implementaciji uzima se samo crvena komponenta. Ovaj postupak generacije terena koristi se ako se želi postići teren sa veoma određenom elevacijom. Visinske mape najčešće se primjenjuju za imitaciju stvarnog terena, tako da se uzme topografska karta i pretvori se u visinsku mapu.

Primjerice, visinska mapa (Slika 3. lijevo) rezultira generiranim terenom prikazanima na Slici 3. desno.



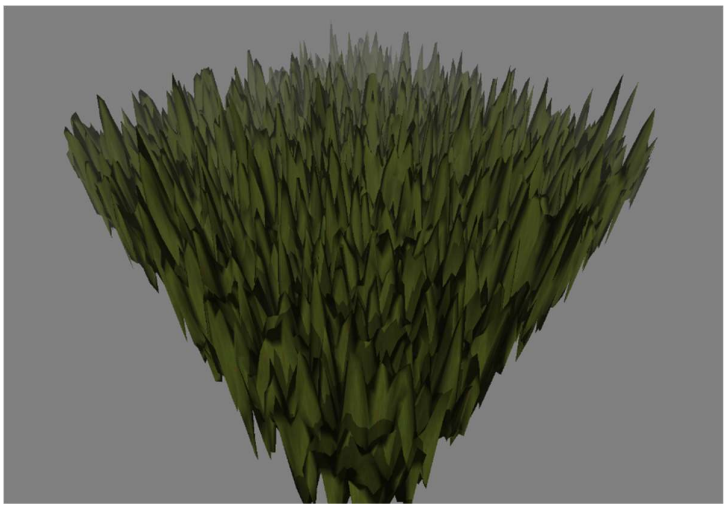
Slika 3. (lijevo) visinska mapa, (desno) generirani teren

Teren je također moguće generirati proceduralno. Proceduralna generacija znači da se na temelju nekog broja koji se naziva sjeme (engl. seed) generiraju visine terena koristeći generator nasumičnih brojeva. Bitna stvar za uočiti je da za isti x, z ulaz generator mora dati isti izlaz y, neovisno o tome koliko je puta pozvan. Generiranjem visina koristeći samo nasumični generator dobivamo rezultat prikazan na Slici 4.



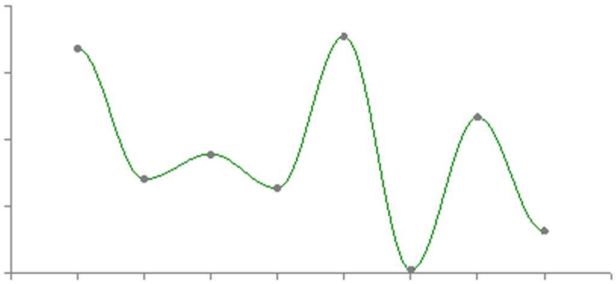
Slika 4. Teren generiran nasumičnim šumom

Očigledno se moraju koristiti tehnike zaglađivanja terena, u programu je implementiran algoritam temeljen na tehnici Perlinovog šuma (engl. Perlin noise) [4]. Prvi korak je izglađivanje visina. Ovo se postiže tako da se za svaki vrh za koji želimo saznati visinu gledaju visine svih susjednih vrhova. Te visine se potom zbrajaju tako da kutovi pridonose najmanje, stranice pridonose više, a najviše pridonosi sam vrh za kojeg želimo saznati visinu. Rezultat je prikazan na Slici 5.



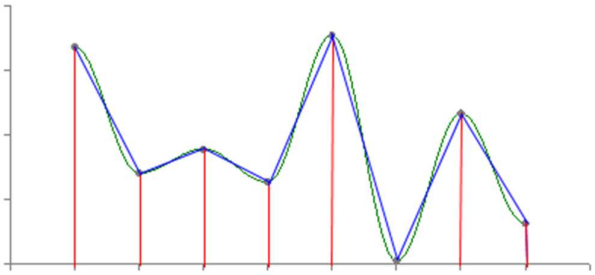
Slika 5. Teren generiran izglađivanjem visina

Sljedeći korak je korištenje interpolacije među vrhovima. U programu je korištena interpolaciju pomoću kosinusa kako bi se postiglo što bolje zaglađivanje. Interpolacija je prikazana na Slici 6.



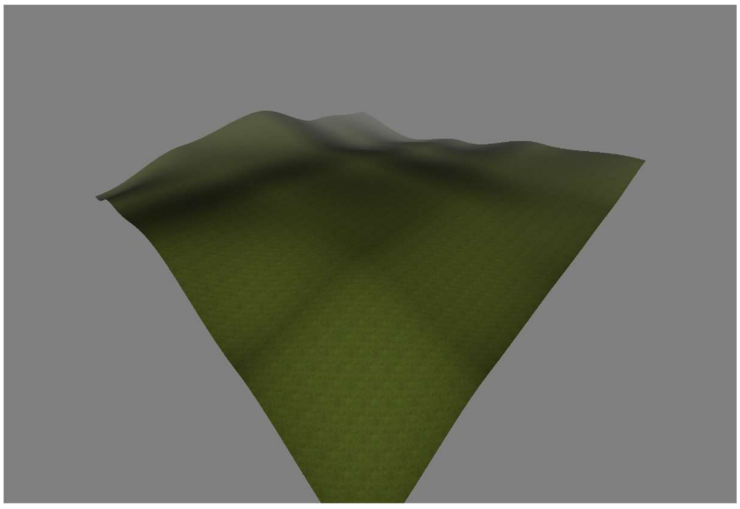
Slika 6. Prikaz interpolacije pomoću kosinusa

Bitno je napomenuti da ako pokušamo nad istim x, z podacima dobiti visine pomoću ove tehnike rezultat će biti isti teren kao da tehniku uopće nismo koristili. To je zato što uzimamo uzorke nad istim x, z pozicijama te efektivno dobivamo rezultat prikazan na slici 7.



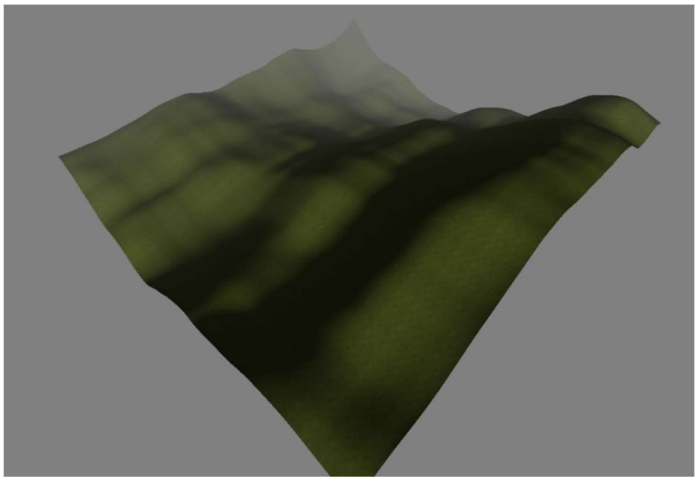
Slika 7. Greška pri korištenju interpolacije

Trebamo podijeliti x, z koordinate svake točke kako bi krivulje interpolacije došle do izražaja te rezultirale terenom prikazanim na Slici 8.



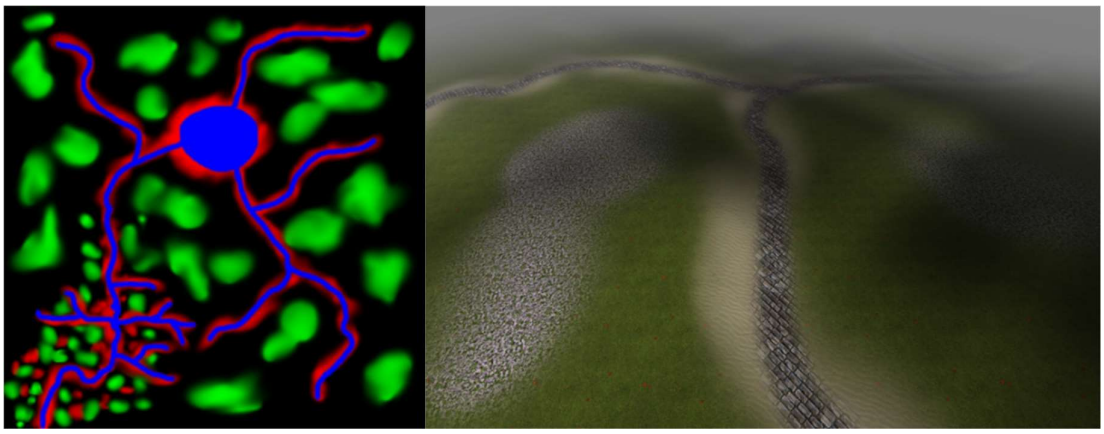
Slika 8. Teren generiran interpolacijom među vrhovima

Broj s kojim dijelimo x, z koordinate se može shvatiti kao frekvencija krivulje koju želimo dobiti. Veliki djelitelj znači jako ravan teren. Teren koji smo dobili mogao bi se smatrati preravnim što načelno nije poželjno. Tome možemo doskočiti tako da umjesto samo jedne krivulje niske frekvencije, koristimo više krivulja različitih frekvencija koje će različito doprinositi visini terena. Na primjer uzmemo krivulju frekvencije 32 koja doprinosi 0.7, krivulju frekvencije 16 koja doprinosi 0.2 i krivulju frekvencije 8 koja doprinosi 0.1. Time dobivamo teren koji je općenito ravan, ali ima male izbočine i nije toliko gladak (Slika 9.).



Slika 9. Konačni teren

Tehnika kojom se teksture primjenjuju na teren se temelji na teksturi koja se zove karta boja (engl blend map). Karta boja je tekstura koja se učita, ali se ne preslikava na teren, nego služi kao karta čije rgb komponente govore koju teksturu primijeniti gdje. U programu se zadaje karta boja i tekstura za svaku od komponenata .



Slika 10. (lijevo) karta boja, (desno) teren s primijenjenim teksturama

3.3. Kupola neba

Kupola neba (engl. skybox) je kocka na čije su stranice teksture primijenjene tako da izgleda kao veoma udaljeni prostor. Najčešće se koristi za prikaz neba ili okoliša u igrama do kojih igrač ne može doći. U alatu kupola neba podržava 4 različite teksture koje se mijenjaju kako vrijeme odmiče (Slika 11).

Slika 11. Prikaz kupole neba s različitim teksturama

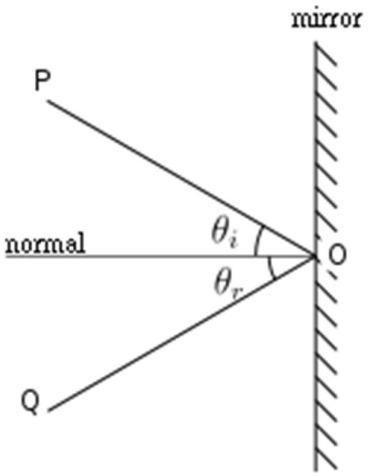
Tekstura koju kupola neba koristi je posebna vrsta teksture OpenGLa koja se zove mapa kocke (engl cube map). Slično kako pikselima 2D teksturama pristupamo pomoću 2D UV koordinata, bitovima mape kocke pristupamo pomoću 3D vektora. OpenGL pretpostavlja da je taj vektor u ishodištu i na temelju pravca kojeg vektor definira dohvaća piksel mape kocke.

3.4. Voda

Voda je zanimljiv element jer je njena mreža poligona veoma jednostavna, sačinjena je od samo dva trokuta koji čine kvadrat, ali njen prikaz je veoma složen.

Cilj prikaza vode je simulacija dva efekta, efekt zrcaljenja (slika 12.) i efekt refrakcije (slika 13.).

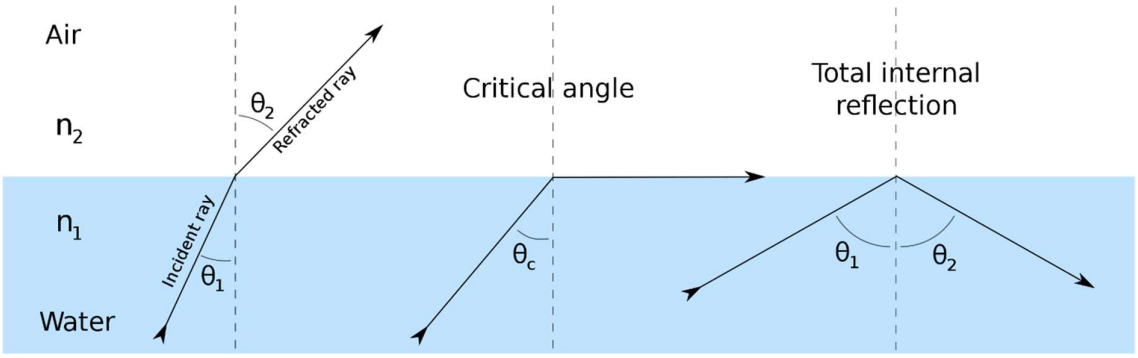
Zrcaljenje je odbijanje zrake svijetla od površine vode i reflektiranje oko normale u točki vode gdje je zraka pala.



Slika 12. Reflektiranje zrake svijetla

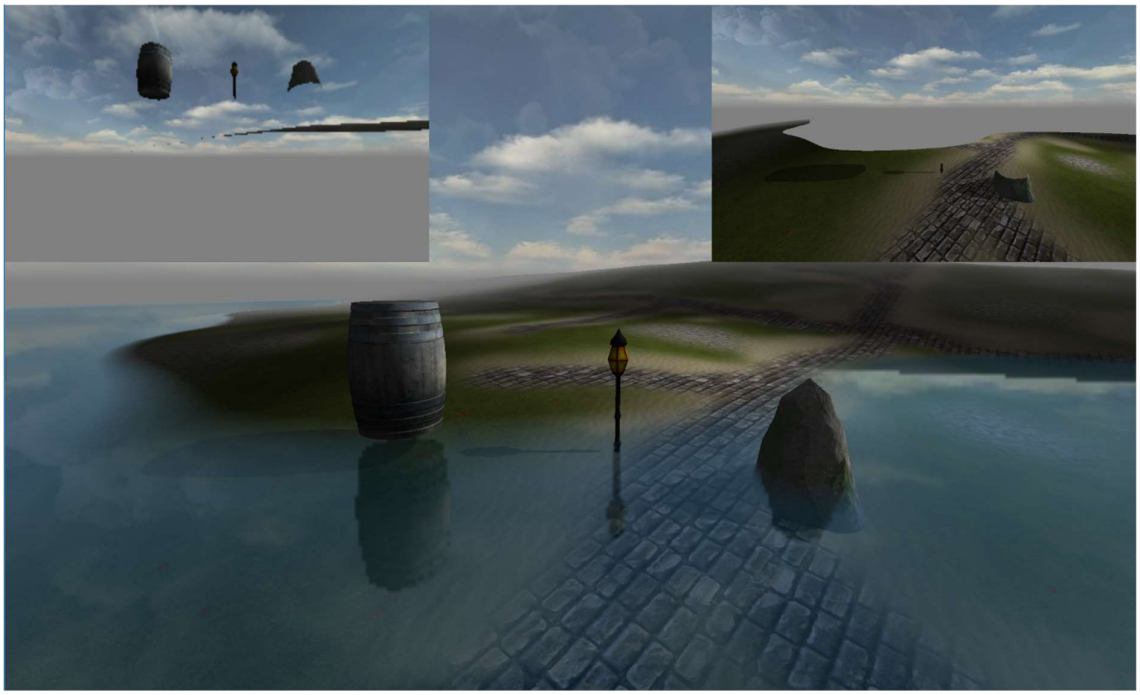
Efekt možemo postići tako da prije nego što nacrtamo scenu u FBO nacrtamo dio scene koji je iznad vode. Tako dobivamo teksturu svega što bi se trebalo zrcaliti u vodi (Slika 14. lijevo gore).

Refrakcija ili lom svjetlosti je skretanje zraka svjetlosti pri prijelazu iz jednoga sredstva u drugo zbog razlike u brzini širenja valova u različitim sredstvima.



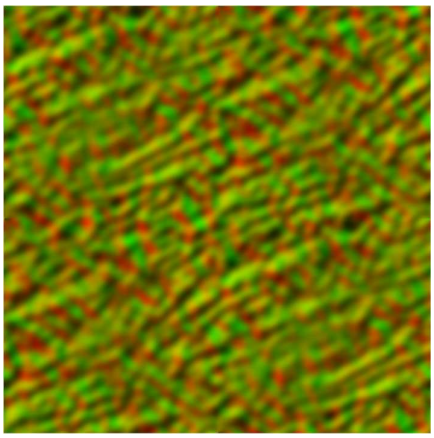
Slika 13. Refrakcija zrake svijetla

U ovom slučaju taj prijelaz je iz vode u zrak što znači da je potrebno promatrati sve ono što je ispod vode. Ponovno je potrebno napraviti FBO u kojeg se smješta tekstura svega ispod vode (Slika 14. desno gore).



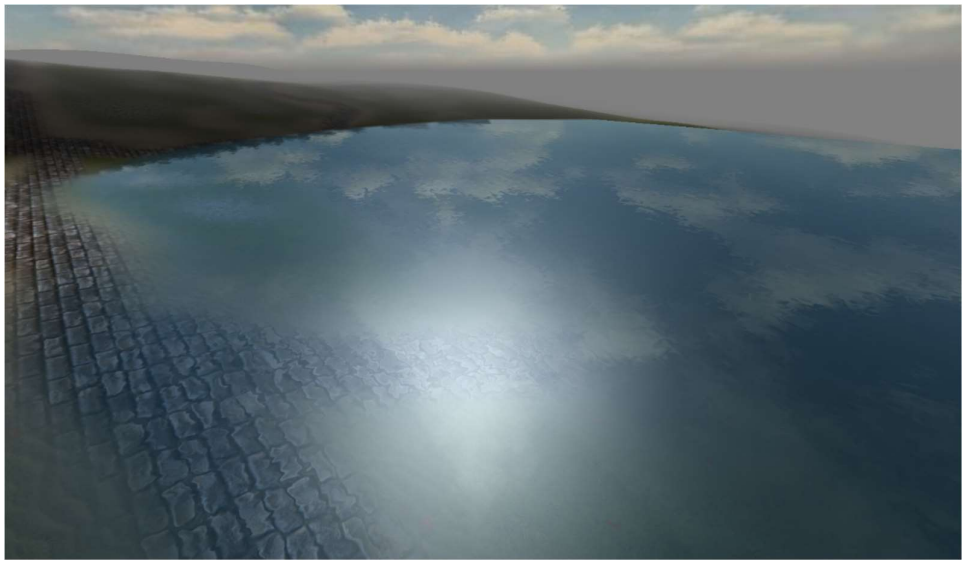
Slika 14. Prikaz vode, (lijevo gore) FBO refleksije, (desno gore) FBO refrakcije

Voda je zbog vjetra i drugih pojava često puna valova i izgleda dinamično, ovako mirna voda izgleda neprirodno. Zbog toga program pokušava simulirati valove koristeći posebnu teksturu koja se naziva DuDv mapa (Slika 15.).



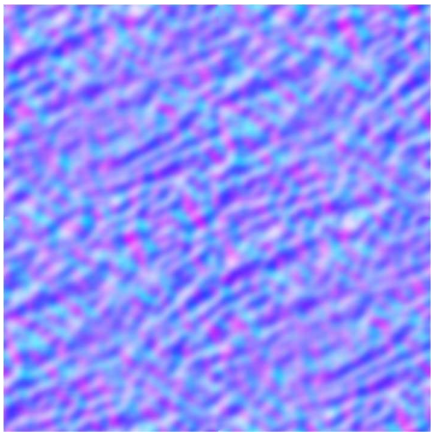
Slika 15. DuDv mapa valova

DuDv mape se koriste tako da pikseli mape služe za iskrivljavanje piksela druge teksture, u ovom slučaju teksture refleksije i refrakcije. Ako pritom koristimo različite dijelove teksture ovisno o vremenu, dobit ćemo efekt valova (Slika 16.).



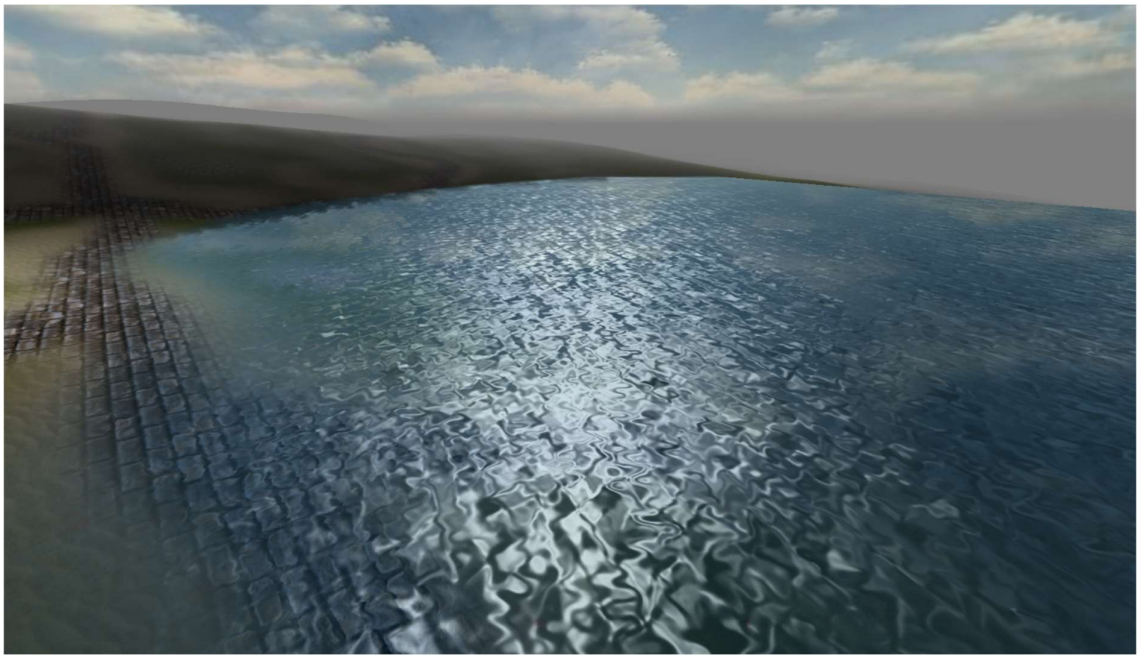
Slika 16. Prikaz vode korištenjem DuDv mape

Makar su refleksije malo iskrivljene, voda još uvijek ne izgleda prirodno. To je zato što DuDv tekstura sama po sebi nije dovoljna jer iako smo promijenili poziciju refleksije, svijetlost koja pada na vodu ne uračunava postojanje valova. Kako bi to popravili potrebna nam je mapa normala, pomoću koje za određenu točku na vodi zadajemo normalu (Slika 17.).



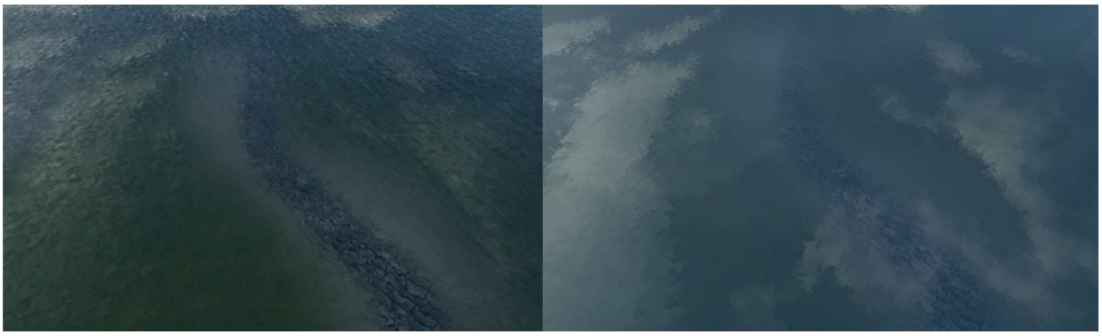
Slika 17. Mapa normala valova vode

Bitno je primijetiti da DuDv mapa i mapa normala izgledaju slično. Dvije teksture trebaju biti napravljene tako da si odgovaraju kako bi se producirali realistični efekti. Primijenivši obje teksture, konačan izgled vode prikazan je na Slici 18.



Slika 18. Prikaz vode s mapom normala

Zadnji efekt koji se koristi u programu je Fresnelov efekt. Efekt se pojavljuje kada gledamo vodu pod različitim kutovima. Ako se vodu gleda izravno odozgora (Slika 19.), tada će biti veoma prozirna. Ako se gleda iz niskog kuta (Slika 20.) tada će biti veoma reflektivna. Efekt se lako implementira tako da se dinamično mijenja prozirnost vode, ovisno o kutu gledanja.



Slika 19. visoki kut gledanja, (lijevo) s Fresnelovim efektom, (desno) bez efekta

Slika 20. niski kut gledanja, (lijevo) s Fresnelovim efektom, (desno) bez efekta

3.5. GUI elementi

Entiteti su osnovni elementi u prikazu pomoću perspektivne matrice, slično GUI elementi su osnovne jedinice u prikazu pomoću ortografske matrice. Iscrtavanje ovih elemenata je veoma jednostavno zbog toga što ne koriste osvjetljenje ili bilo koje druge 3D efekte.

Složenost u GUI elementima dolazi iz različitih grupacija elementa i tipova elemenata koji su podržani. U programu postoje gumbi, kučice za pisanje, grupe elemenata i prozori za prikaz entiteta. Kako se odnosi među elementima ne tiče iscrtavanja, neće biti obrađeno u ovom radu.

GUI elementi također imaju svoju mrežu poligona i teksturu. Mreža poligona je najčešće jednostavan kvadrat s iznimkom prozora za prikaz entiteta. Teksture koje koriste također su veoma jednostavne, svaki element podržava korištenje jedne teksture s iznimkom gumba koji koristi dvije, jednu za stanje kada je pritisnut i drugu za stanje kada je otpušten.

3.6. Tekst

Tekst je sačinjen od GUI elementa, svaki element je jedno slovo teksta. Posebnost teksta dolazi iz potrebnih podataka i teksture koju koristi.

Tekst se iscrtava na ekran tako da se odredi njegova pozicija i stvori pokazivač teksta na tom mjestu. Kada se iscrta jedan znak teksta pokazivač se pomiče za vrijednost koju specificira taj znak. Veličina znaka ne ukazuje na količinu pomaka pokazivača koji je potreban jer se inače ne bi mogli iscrtavati znakovi poput razmaka.

Svaki tekst je prikazan u određenom fontu, koji mora specificirati podatke za znakove koje podržava. Podaci o znakovima sadrže ASCII vrijednost znaka, koliko je znak širok i visok, koliko je potrebno pomaknuti pokazivač teksta nakon ispisa znaka i gdje u teksturi se nalazi tekstura tog znaka. Svi podaci nalaze se u datotekama s .fnt ekstenzijom. Za generaciju fontova korišten je program Hiero [5].

Za iscrtavanje običnog teksta dovoljna je tekstura prikazana na Slici 21.

Slika 21. Osnovna tekstura za tekst

Prilikom iscrtavanja običnog teksta znakovi će imati točno oblik kakav se može vidjeti na Slici 21. uz eventualnu promijenjenu boje. No ukoliko se tekst želi iscrtati s efektima poput obruba, sjene ili sličnog. Tekstura koja se koristi mora biti drugačija kako bi dobili dodatne informacije o znaku. Ono što je u teksturi potrebno je da se prozirnost znakova smanjuje što smo više udaljeni od središta znaka (Slika 22.).

Slika 22. Tekstura za tekst s prozirnošću

Ovo nam omogućuje da kasnije u sjenčaru iskoristimo tu informaciju o prozirnosti zajedno s nekim parametrima poput veličine obruba kako bi iscrtali tekst koji je prikazan na Slici 23. (dolje).

Slika 23. Prikaz teksta, (gore) običan tekst, (dolje) tekst s obrubom

3.7. Sustav čestica

Efekti poput vatre ili dima teški su za postići s dosad navedenim elementima, sustav čestica služi za iscrtavanje baš takvih pojava (Slika 24.).

Slika 24. (lijevo) vatra prikazana česticama, (desno) dim prikazan česticama

Položaj i gibanje sustava čestica u 3D prostoru kontrolira odašiljač (engl. emitter). Odašiljač djeluje kao izvor čestica, a njegov položaj u 3D prostoru određuje gdje su čestice generirane i kamo se kreću. Odašiljač sadrži skup parametara ponašanja čestica. Ovi parametri mogu uključivati brzinu stvaranja čestica, životni vijek čestica, početni vektor brzine čestica i drugo. Uobičajeno je da svi ili većina tih parametara budu ograničeno nasumični umjesto da koriste precizne numeričke vrijednosti.

Petlja ažuriranja sustava čestica koja se izvodi za svaki korak animacije može se razdvojiti u dvije različite faze, ažuriranje parametara i fazu iscrtavanja.

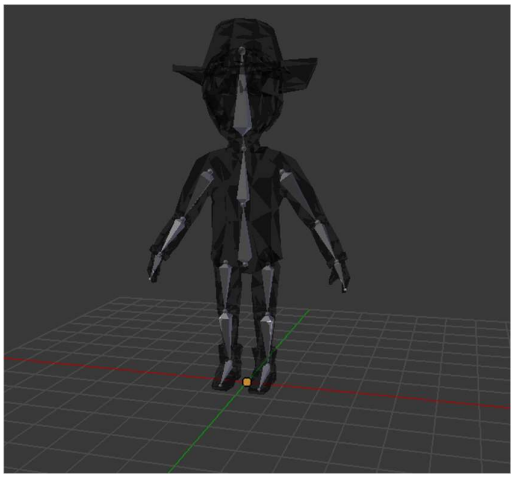
Čestice su objekti koji pamte svoj položaj, brzinu i vijek života na temelju kojih se ažurira

njihov položaj u prostoru. Iscrtavaju se kao kvadrati koji uvijek gledaju prema kameri.

3.8. Animirani modeli

Animirani modeli imaju sve podatke i funkcije entiteta, drugim riječima sadrže informacije o poziciji, normalama i koordinatama tekstura svakog pojedinog vrha modela. Ono što ih čini posebnima je što dodatno imaju i kostur pomoću kojih se model može animirati. Kostur je sačinjen od kostiju koje se mogu shvatiti kao struktura podataka koja sadrži vrhove modela na koje ta kost utječe te težine koje određuju jačinu tog utjecaja. Konkretno u implementaciji programa podaci nisu oformljeni tako da kost sadrži podatke o tome na koje vrhove utječe, nego vrhovi sadrže koje kosti na njih utječu. Ovo je napomenuto jer je bitno za optimizacijske svrhe, jednostavnom promjenom načina spremanja podataka možemo ubrzati postavljanje poze i time i animiranje modela.

Kosti su složene u hijerarhiju sa točno jednom korijenskom odnosno vršnom kosti. Pomoću informacija o tome koja kost je čije dijete ili roditelj možemo dobiti efekt kojim pomicanjem kosti nadlaktice pomičemo kosti ruke.



Slika 25. Prikaz modela i njegovog kostura

Sama animacija objekta je zapravo skup poza u određenim trenucima, a program interpolira među tim pozama te postavlja trenutnu pozu objekta. Poza objekata u određenom trenutku naziva se ključni okvir ili kadar (engl. keyframe) i zapravo je skup transformacija nad svakom pojedinom kosti.

Kako bi animiranje bilo moguće, interpoliranje između poza objekta mora biti moguće.

Poza nije ništa drugo nego matrica transformacije koja je sastavljena od transformacije pozicije, skaliranja i rotacije. Interpolacija među pozama bi bila jednostavna ukoliko bi zanemarili rotacijski dio matrice, no to nije moguće. Interpolacija između rotacijskih matrica nije lako izvediva, a da se pritom dobije željeni rezultat. Stoga dio transformacije koji se tiče rotacije mora biti zapisan u drugom obliku, konkretno u obliku kvaterniona. Kvaternioni su široka tema koja neće biti obrađena u ovom radu, no ono što je bitno je da su oni matematički objekt veoma sličan 4D vektoru pomoću kojeg možemo zapisivati rotacije 3D objekata. Također kvaternioni nude način za interpolaciju pomoću algoritma koji se naziva SLERP (engl. Spherical linear interpolation).

```
Quaternion Quaternion::Slerp(const Quaternion& start, const Quaternion& end, const double t) {
    Quaternion v0 = Normalize(start);
    Quaternion v1 = Normalize(end);

    float dot = Dot(v0, v1);
    const double DOT_THRESHOLD = 0.9995;
    if (fabs(dot) > DOT_THRESHOLD) {
        Quaternion result = v0 + (float)t * (v1 - v0);
        return Normalize(result);
    }

    if (dot < 0.0f) {
        v1 = -1 * v1;
        dot = -dot;
    }

    if (dot < -1) dot = -1.0f;
    else if (dot > 1) dot = 1.0f;

    double theta_0 = acos(dot);  // theta_0 = angle between input vectors
    double theta = theta_0 * t;    // theta = angle between v0 and result

    Quaternion v2 = v1 - (v0 * dot);
    v2 = Normalize(v2);              // { v0, v2 } is now an orthonormal basis

    return v0 * (float)cos(theta) + v2 * (float)sin(theta);
}
```

Izlist programskog koda 3. Implementacija SLERP algoritma

Uz poze i način interpolacije među njima, zadatak animiranja je jednostavan. Kako vrijeme teče biraju se dvije poze između kojih se interpolira i finalna poza se postavlja na modelu (Slika 26.).

Slika 26. Dva kadra animacije modela

4. Tehnike prikaza

4.1. Atlas tekstura

Atlas tekstura je tehnika u kojoj jedna slika sadrži više manjih tekstura (Slika 27.). Ova tehnika se koristi kako bi se postiglo ubrzanje pri učitavanju tekstura. Kako bi se teksture primijenile OpenGLu je potrebno reći koja se tekstura koristi, a ako su sve teksture u jednoj velikoj slici potrebno je samo navesti tu jednu veliku sliku kao izvor tekstura. To zahtjeva prilagođavanje sustava na taj način rada, jer sad umjesto samo osnovnih podataka teksture također moraju imati neku vrstu identifikatora koji govori gdje je njihova tekstura u atlasu.

Slika 27. Primjer atlasa tekstura

4.2. Model osvjetljenja

Model osvjetljenja koji se koristi je Phongov model i primjenjuje se na entitete, teren, animirane modele i vodu [6]. Phongov model se sastoji od tri komponente: ambijente, difuzne i zrcalne (engl. specular). Ukupna boja vrha se računa na sljedeći način:

$$\text{boja} = \text{ambijent} + \text{boja\_vrha} * \text{difuzna} + \text{zrcalna}$$

Ambijentna komponenta je sveprisutna u sceni i jednako doprinosi svim elementima.

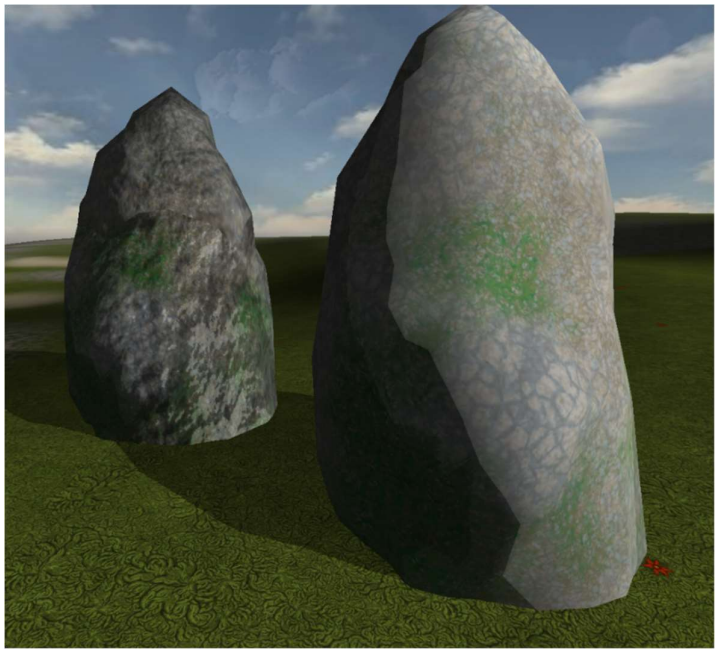
Difuzna komponenta ovisi o tome koliko je vrh izložen izvoru svjetlosti. Računa se tako da se normala vrha skalarno pomnoži sa vektorom od vrha do izvorišta svjetlosti.

Zrcalna komponenta ovisi o očištu i reflektiranoj zraci svjetla od vrha. Predstavlja odbljesak predmeta i računa se tako da se zraka svjetlosti koja dolazi u vrh reflektira oko njegove normala te se tako dobije vektor odbljeska. Vektor odbljeska se tada skalarno množi s vektorom od vrha do očišta.

Program podržava i mijenjanje faktora Phogovog modela kroz postavljanje vrijednosti u teksturi objekta. Faktori utječu na to koliko će objekt zrcaliti svijetla i koliko dobro upija svjetlost.

4.3. Mape normala

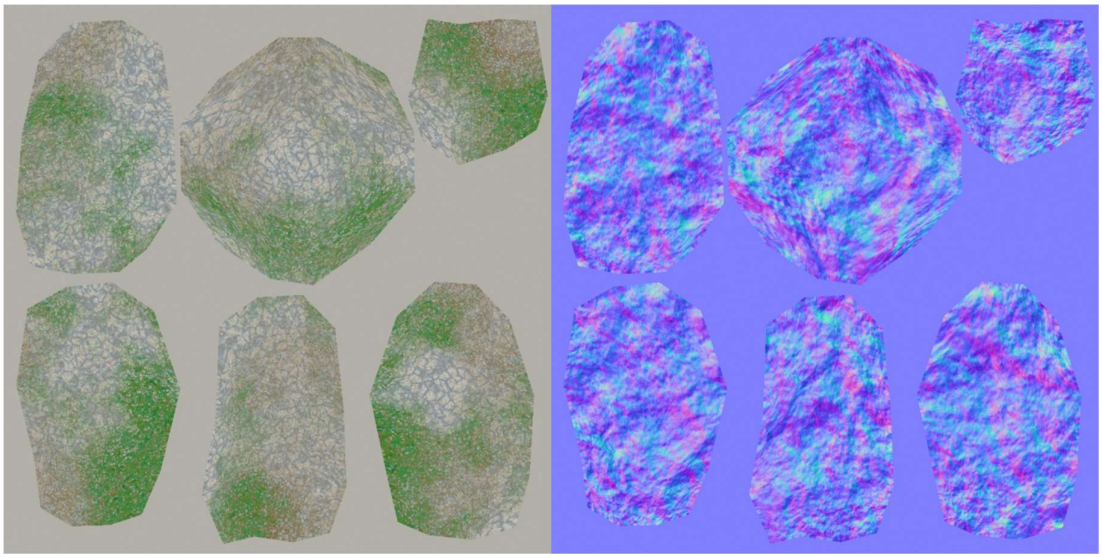
Mape normala već su spomenute kod iscrtavanja vode, ali su također koristan alat za prikazivanje modela (Slika 28.). Umjesto da model učinimo jako kompleksnim sa puno udubljenja kako bi dobili dinamičan raspored svijetlih i tamnih mjesta, možemo rabiti mapu normala.



Slika 28. Primjer korištenja mape normala, (lijevo) s mapom, (desno) bez mape

Normale objekata zamijenjene su normalama koje specificira tekstura mape. Bitno je da

tekstura objekta bude napravljena zajedno s mapom normala, jer moraju biti u paru kako bi efekt bio realističan. Par teksture i mape prikazan je na Slici 29.



Slika 29. (lijevo) tekstura objekta, (desno) mapa normala objekta

4.4. Sjene

Sjene se postižu korištenjem mape sjena (engl. Shadow map). Mapa sjena je FBO tekstura u koju se crtaju svi objekti koji trebaju bacati sjenu. Prilikom iscrtavanja scene, dubina objekta se uspoređuje s dubinom na mapi sjena. Ako je dubina objekta veća nego dubina na mapi tada je objekt skriven drugim objektom i on treba biti u sjeni.

Problem koji se javlja pri uspoređivanju dubine objekta i dubine mape je što su u mapu objekti crtani u drugom koordinatnom sustavu. Kako bi mapa radila ispravno u nju je objekte nužno crtati iz smjera svjetla koje je univerzalno za cijelu scenu. To će poredati objekte od najbližeg do najdaljeg od svjetla. Budući da mapa koristi drugačiji koordinatni sustav, prije usporedbe objekte je nužno transformirati u koordinatni sustav mape.

5. Konačni rezultati

Rezultat rada je program za prikaz prirodnih okoliša. Program nudi mogućnost učitavanja modela i tekstura te njihov prikaz na željenom položaju. Podržava se generiranje terena, prikaz vode i nudi se rudimentarni sustav čestica. Na Slici 30. može se vidjeti da program radi na 30 sličica u sekundi dok prikazuje 40 000 vrhova entiteta, teren veličine 16 000 vrhova, 250 čestica te prikazuje vodu i sjene objekata. Korišteni modeli su preuzeti s interneta u standardnim formatima [7].

Slika 30. Prikaz scene u programu

Računalo koje je služilo za testiranje rada je Lenovo Thinkpad T460s, s ugrađenim grafičkim procesorom Intel HD Graphics 520.

6. Zaključak

Računalna grafika je široko područje koje se konstantno razvija. Iz ovog kratkog pregleda tehnika i elemenata scene možemo zaključiti da su alati koji se razvijaju u svrhu objedinjavanja svih tehnika veoma složeni i rigorozno testirani. Postoje mnoge prepreke pri izradi takvog alata, od optimizacije, do samog snalaženja u kodu koji stalno raste. Također je bitno uočiti da makar su neke od tehnika složene korisniku bi se trebale predstavljati jednostavno i sažeto.

Zbog širene koje pružaju takvi alati lako je razumjeti da stvaraju dobru podlogu za daljnje istraživanje i povezivanje sve više sustava.

7. Literatura

[1] M. Čupić i Ž. Mihajlović, Interaktivna računalna grafika kroz primjere u OpenGL-u, Zagreb: Sveučilište u Zagrebu, 2016.

[2] Learn OpenGL https://learnopengl.com/ Svibanj 2018.

[3] OpenGL Step by Step - OpenGL Development, ogldev.atspace.co.uk. Svibanj 2018.

[4] http://flafla2.github.io/2014/08/09/perlinnoise.html Svibanj 2018.

[5] Hiero program https://github.com/libgdx/libgdx/wiki/Hiero

[6] I. Pandžić, T. Pejša, K. Matković, H. Benko, A. Čereković i M. Matijašević, Virtualna Okruženja: Interaktivna 3D grafika i njene primjene, Zagreb: Element, 2011.

[7] GitHub modeli https://github.com/TheThinMatrix Svibanj 2018.

Prikaz prirodnih okoliša

Sažetak

Ovaj završni rad obrađuje temu prikaza prirodnih okoliša. Prvi dio rada fokusira se na OpenGL i njegove strukture podataka kako bi se pružao kratak uvod u temu. Nakon toga se obrađuju svi elementi scene koji su uključeni u prikaz pomoću izrađenog programa. Konačno se obrađuju neke tehnike za prikaz poput Phongovog modela osvjetljenja, mape sjena i drugih.

Ključne riječi: mapa sjena, Phongov model osvjetljenja, OpenGL

Natural Environments Rendering

Abstract

This final paper deals with the topic of natural environments. The first part of the paper focuses on OpenGL and its data structures to get a brief introduction to the topic. Afterwards, all the scene elements included in the display of the environment. Finally, the paper deals with the techniques of displaying environments such as the Phong model of illumination, shadow map and others are discussed

Keywords: shadow mapping, Phong lighting model, OpenGL